

Tivoli Workload Scheduler LoadLeveler



Using and Administering

Version 3 Release 4

Tivoli Workload Scheduler LoadLeveler



Using and Administering

Version 3 Release 4

Note

Before using this information and the product it supports, read the information in "Notices" on page 689.

Seventh Edition (October 2006)

This edition applies to version 3, release 4 of IBM Tivoli Workload Scheduler LoadLeveler (product numbers 5765-E69 and 5724-I23) and to all subsequent releases and modifications until otherwise indicated in new editions. This edition replaces SA22-7881-05. Significant changes or additions to the text and illustrations are indicated by a vertical line (|) to the left of the change.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this publication, or you can send your comments to the address:

International Business Machines Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+845+432-9405

FAX (Other Countries):

Your International Access Code +1+845+432-9405

IBMLink™ (United States customers only): IBMUSM10(MHVRCFS)

Internet e-mail: mhvrcfs@us.ibm.com

If you want a reply, be sure to include your name, address, and telephone or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this book
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

©Copyright 1986, 1987, 1988, 1989, 1990, 1991 by the Condor Design Team.

©Copyright International Business Machines Corporation 1986, 2006. All rights reserved. US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures ix

Tables xi

About this book xiii

Who should use this book xiii

Conventions and terminology used in this book xiii

Prerequisite and related information xiv

IBM System Blue Gene Solution documentation xv

Using LookAt to look up message explanations xv

How to send your comments xv

Summary of changes xvii

Part 1. Overview of TWS LoadLeveler concepts and operation 1

Chapter 1. What is LoadLeveler? 3

LoadLeveler basics 3

LoadLeveler: A network job management and
scheduling system 4

Job definition 4

Machine definition 5

How LoadLeveler schedules jobs 7

How LoadLeveler daemons process jobs 7

The master daemon 8

The Schedd daemon 9

The startd daemon 10

The negotiator daemon 12

The kbdd daemon 13

The gsmonitor daemon 13

The LoadLeveler job cycle 14

LoadLeveler job states 18

Consumable resources 21

Consumable resources and AIX Workload

Manager 22

Overview of reservations 23

Fair share scheduling overview 25

Chapter 2. Getting a quick start using the default configuration 27

What you need to know before you begin 27

Using the default configuration files 27

LoadLeveler for Linux quick start 28

Quick installation 28

Quick configuration 28

Quick verification 28

Post-installation considerations 29

Starting LoadLeveler 29

Location of directories following installation 30

Chapter 3. What operating systems are supported by LoadLeveler? 33

AIX and Linux compatibility 33

Restrictions for LoadLeveler for Linux 34

Features not supported in Linux 34

Restrictions for LoadLeveler AIX and Linux

mixed clusters 35

Part 2. Configuring and managing the TWS LoadLeveler environment . 37

Chapter 4. Configuring the LoadLeveler environment 39

Modifying a configuration file 40

Defining LoadLeveler administrators 41

Defining a LoadLeveler cluster 41

Choosing a scheduler 42

Setting negotiator characteristics and policies 43

Specifying alternate central managers 44

Defining network characteristics 45

Specifying file and directory locations 45

Configuring recording activity and log files 46

Setting up file system monitoring 50

Defining LoadLeveler machine characteristics 51

Defining job classes that a LoadLeveler machine
will accept 51

Specifying how many jobs a machine can run 52

Defining security mechanisms 52

Configuring LoadLeveler to use cluster security
services 53

Defining usage policies for consumable resources 56

Enabling support for bulk data transfer and rCxt
blocks 57

Gathering job accounting data 57

Collecting job resource data on serial and parallel
jobs 58

Collecting job resource data based on machines 58

Collecting job resource data based on events 59

Collecting job resource information based on user
accounts 59

Collecting the accounting information and
storing it into files 60

Producing accounting reports 60

Correlating AIX and LoadLeveler accounting
records 61

64-bit support for accounting functions 61

Example: Setting up job accounting files 61

Managing job status through control expressions 63

How control expressions affect jobs 63

Tracking job processes 64

Querying multiple LoadLeveler clusters 65

Handling switch-table errors 66

Providing additional job-processing controls through
installation exits 66

Controlling the central manager scheduling cycle 67

Handling DCE security credentials 68

Handling an AFS token	69
Filtering a job script	70
Writing prolog and epilog programs	70
Using your own mail program	75

Chapter 5. Defining LoadLeveler resources to administer 77

Steps for modifying an administration file	77
Defining machines	78
Planning considerations for defining machines	79
Machine stanza format and keyword summary	79
Examples: Machine stanzas	80
Defining adapters	80
Configuring dynamic adapters	81
Configuring InfiniBand adapters	81
Adapter stanza format and keyword summary	82
Examples: Adapter stanzas	82
Defining classes	83
Using limit keywords	83
Allowing users to use a class	86
Class stanza format and keyword summary	86
Examples: class stanzas	86
Defining user substanzas in class stanzas	88
Examples: substanzas	88
Defining users	91
User stanza format and keyword summary	91
Examples: User stanzas	91
Defining groups	92
Group stanza format and keyword summary	92
Examples: Group stanzas	92
Defining clusters	93
Cluster stanza format and keyword summary	93
Examples: Cluster stanzas	94

Chapter 6. Performing additional administrator tasks 95

Setting up the environment for parallel jobs	96
Scheduling considerations for parallel jobs	96
Steps for reducing job launch overhead for parallel jobs	96
Steps for allowing users to submit interactive POE jobs	97
Setting up a class for parallel jobs	98
Setting up a parallel master node	99
Configuring LoadLeveler to support MPICH jobs	99
Configuring LoadLeveler to support MVAPICH jobs	100
Configuring LoadLeveler to support MPICH-GM jobs	100
Using the BACKFILL scheduler	101
Tips for using the BACKFILL scheduler	104
Example: BACKFILL scheduling	104
Using an external scheduler	105
Replacing the default LoadLeveler scheduling algorithm with an external scheduler	106
Customizing the configuration file to define an external scheduler	107
Steps for getting information about the LoadLeveler cluster, its machines, and jobs	108
Assigning resources and dispatching jobs	112

Example: Changing scheduler types	115
Preempting and resuming jobs	116
Overview of preemption.	116
Planning to preempt jobs	117
Steps for configuring a scheduler to preempt jobs	120
Configuring LoadLeveler to support reservations	121
Steps for configuring reservations in a LoadLeveler cluster	121
Collecting accounting data for reservations	126
Steps for integrating LoadLeveler with AIX Workload Manager	127
LoadLeveler support for checkpointing jobs	129
Checkpoint keyword summary	129
Planning considerations for checkpointing jobs	130
AIX checkpoint and restart limitations	131
Naming checkpoint files and directories	134
Removing old checkpoint files.	135
LoadLeveler scheduling affinity support	136
Configuring LoadLeveler to use scheduling affinity	136
Linux CPU affinity support.	137
Assigning Linux CPU affinity to application processes	138
LoadLeveler multicluster support.	139
Configuring a LoadLeveler multicluster	140
Steps for configuring a LoadLeveler multicluster	141
Steps for securing communications within a LoadLeveler multicluster	143
LoadLeveler Blue Gene support	143
Configuring LoadLeveler Blue Gene support	145
Blue Gene advance reservation support	147
Blue Gene fair share scheduling support	147
Blue Gene heterogeneous memory support	147
Using fair share scheduling.	148
Fair share scheduling keywords	148
Reconfiguring fair share scheduling keywords	150
Example: three groups share a LoadLeveler cluster.	151
Example: two thousand students share a LoadLeveler cluster	152
Querying Information about fair share scheduling	153
Resetting fair share scheduling	153
Saving historic data	153
Restoring saved historic data	153
Procedure for recovering a job spool.	154

Chapter 7. Using LoadLeveler's GUI to perform administrator tasks 155

Job-related administrative actions.	155
Machine-related administrative actions	158

Part 3. Submitting and managing TWS LoadLeveler jobs 163

Chapter 8. Building and submitting jobs 165

Building a job command file	165
---------------------------------------	-----

Using multiple steps in a job command file	166
Examples: Job command files	167
Editing job command files	171
Defining resources for a job step	171
Working with coscheduled job steps.	171
Submitting coscheduled job steps.	171
Determining priority for coscheduled job steps	171
Supporting preemption of coscheduled job steps	172
Coscheduled job steps and commands and APIs	172
Termination of coscheduled steps.	173
Using bulk data transfer.	173
Preparing a job for checkpoint/restart	174
Preparing a job for preemption	177
Submitting a job command file	177
Submitting a job using a submit-only machine	178
Working with parallel jobs	178
Scheduler support for parallel jobs	178
Step for controlling whether LoadLeveler copies	
environment variables to all executing nodes	179
Ensuring that parallel jobs in a cluster run on	
the correct levels of PE and LoadLeveler	
software	179
Task-assignment considerations	180
Submitting jobs that use striping	182
Running interactive POE jobs	188
Running MPICH, MVAPICH, and MPICH-GM	
jobs	188
Examples: Building parallel job command files	191
Obtaining status of parallel jobs	196
Obtaining allocated host names	197
Working with reservations	197
Understanding the reservation life cycle	198
Creating new reservations	200
Submitting jobs to run under a reservation	202
Removing bound jobs from the reservation	203
Querying existing reservations	204
Modifying existing reservations	204
Canceling existing reservations	205
Submitting jobs requesting scheduling affinity	205
Submitting and monitoring jobs in a LoadLeveler	
multicluster	205
Steps for submitting jobs in a LoadLeveler	
multicluster environment	206
Submitting and monitoring Blue Gene jobs	208

Chapter 9. Managing submitted jobs 211

Querying the status of a job	211
Working with machines	211
Displaying currently available resources	212
Setting and changing the priority of a job	212
Example: How does a job's priority affect	
dispatching order?.	213
Placing and releasing a hold on a job	213
Canceling a job.	214
Checkpointing a job	214

Chapter 10. Example: Using commands to build, submit, and manage jobs. 215

Chapter 11. Using LoadLeveler's GUI to build, submit, and manage jobs . . . 217

Building jobs	217
Editing the job command file	229
Submitting a job command file	230
Displaying and refreshing job status.	231
Sorting the Jobs window	232
Changing the priority of your jobs	232
Placing a job on hold.	233
Releasing the hold on a job.	233
Canceling a job.	233
Modifying consumable resources and other job	
attributes	233
Taking a checkpoint	234
Adding a job to a reservation	234
Removing a job from a reservation	234
Displaying and refreshing machine status	235
Sorting the Machines window	236
Finding the location of the central manager	237
Finding the location of the public scheduling	
machines.	237
Finding the type of scheduler in use.	237
Specifying which jobs appear in the Jobs window	237
Specifying which machines appear in Machines	
window	238
Saving LoadLeveler messages in a file	238

Part 4. TWS LoadLeveler interfaces reference 241

Chapter 12. Configuration file reference 243

Configuration file syntax	243
Numerical and alphabetical constants	244
Mathematical operators	244
64-bit support for configuration file keywords	
and expressions	244
Configuration file keyword descriptions	244
User-defined keywords	293
LoadLeveler variables	294
Variables to use for setting dates	299
Variables to use for setting times	299

Chapter 13. Administration file reference 301

Administration file structure and syntax	301
Stanza characteristics	303
Syntax for limit keywords	304
64-bit support for administration file keywords	305
Administration file keyword descriptions	306

Chapter 14. Job command file reference 333

Job command file syntax	333
Serial job command file	333
Parallel job command file	334
Syntax for limit keywords	334
64-bit support for job command file keywords	335

Job command file keyword descriptions	336
Job command file variables	372
Run-time environment variables	374
Job command file examples	375

Chapter 15. Graphical user interface (GUI) reference. 377

Starting the GUI	377
Specifying GUI options	377
The LoadLeveler main window	378
Getting help using the GUI.	379
Differences between LoadLeveler’s GUI and other graphical user interfaces.	379
GUI typographic conventions	379
64-bit support for the GUI	380
Customizing the GUI.	380
Syntax of an Xloadl file	380
Modifying windows and buttons	380
Creating your own pull-down menus	381
Customizing fields on the Jobs window and the Machines window.	382
Modifying help panels	383

Chapter 16. Commands 385

llactmrg - Collect machine history files	387
llbind - Bind job steps to a reservation	389
llcancel - Cancel a submitted job	392
llchres - Change attributes of a reservation	395
llckpt - Checkpoint a running job step	400
llclass - Query class information	403
llclusterauth - Generates public and private keys	408
llctl - Control LoadLeveler daemons.	409
lldbconvert - Job migration utility	414
llextrPD - Extract data from an RSCT peer domain	415
llfavorjob - Reorder system queue by job	419
llfavoruser - Reorder system queue by user	421
llfs - Fair share scheduling queries and operations	422
llhold - Hold or release a submitted job	426
llinit - Initialize machines in the LoadLeveler cluster.	429
llmkres - Make a reservation	431
llmodify - Change attributes of a submitted job step	435
llmovejob - Move a single idle job from the local cluster to another cluster	440
llmovespool - Move job records	442
llpreempt - Preempt a submitted job step	444
llprio - Change the user priority of submitted job steps	447
llq - Query job status.	449
llqres - Query a reservation.	468
llrmres - Cancel a reservation	474
llrunscheduler - Run the central manager’s scheduling algorithm.	476
llstatus - Query machine status	477
llsubmit - Submit a job	494
llsummary - Return job resource information for accounting	496

Chapter 17. Application programming interfaces (APIs) 503

64-bit support for the LoadLeveler APIs	505
AIX APIs	505
Linux APIs	506
Accounting API	506
GetHistory subroutine	507
llacctval user exit	509
Checkpointing API	510
ckpt subroutine.	511
ll_ckpt subroutine	512
ll_init_ckpt subroutine	515
ll_set_ckpt_callbacks subroutine	517
ll_unset_ckpt_callbacks subroutine	518
Configuration API.	519
ll_config_changed subroutine	520
ll_read_config subroutine	521
Data access API	522
Using the data access API	522
Understanding the LoadLeveler data access object model.	523
Understanding the Blue Gene object model	524
Understanding the Class object model	524
Understanding the Cluster object model	525
Understanding the Fairshare object model.	525
Understanding the Job object model.	525
Understanding the Machine object model	527
Understanding the MCluster object model.	528
Understanding the Reservations object model	528
Understanding the Wlmstat object model	529
ll_deallocate subroutine	530
ll_free_objs subroutine	531
ll_get_data subroutine	532
ll_get_objs subroutine	574
ll_next_obj subroutine	577
ll_query subroutine	578
ll_reset_request subroutine	579
ll_set_request subroutine	580
Examples of using the data access API	584
Error handling API	590
ll_error subroutine.	591
Fair share scheduling API	592
ll_fair_share subroutine	593
Query API	595
ll_free_jobs subroutine	596
ll_free_nodes subroutine.	597
ll_get_jobs subroutine	598
ll_get_nodes subroutine	599
Reservation API	600
ll_bind subroutine.	601
ll_change_reservation subroutine	604
ll_init_reservation_param subroutine	608
ll_make_reservation subroutine	609
ll_remove_reservation subroutine.	612
Submit API	614
llfree_job_info subroutine	615
llsubmit subroutine	616
monitor_program user exit	618
Workload management API	619
ll_cluster subroutine	620
ll_cluster_auth subroutine	622

ll_control subroutine	624
ll_modify subroutine	628
ll_move_job subroutine	632
ll_move_spool subroutine	634
ll_preempt subroutine	637
ll_preempt_jobs subroutine	639
ll_run_scheduler subroutine	642
ll_start_job subroutine	643
ll_start_job_ext subroutine	645
ll_terminate_job subroutine	649

Appendix A. Troubleshooting

LoadLeveler 651

Frequently asked questions	651
Why won't LoadLeveler start?	651
Why won't my job run?	652
Why won't my parallel job run?	654
Why won't my checkpointed job restart?	655
Why won't my submit-only job run?	656
Why won't my job run on a cluster with both AIX and Linux machines?	656
Does my Linux machine support CPU affinity? Why does a job stay in the Pending (or Starting) state?	656
What happens to running jobs when a machine goes down?	657
What happens if the central manager isn't operating?	658
How do I recover resources allocated by a Schedd machine?	660
Why can't I find a core file on Linux?	660
Why am I seeing inconsistencies in my llfs output?	661
What happens if errors are found in my configuration or administration file?	661
Other questions	661
Troubleshooting in a multicluster environment	663
How do I determine if I am in a multicluster environment?	663
How do I determine how my multicluster environment is defined and what are the inbound and outbound hosts defined for each cluster?	663
Why is my multicluster environment not enabled?	663

How do I find log messages from my multicluster defined installation exits?	663
Why won't my remote job be submitted or moved?	664
Why did the CLUSTER_REMOTE_JOB_FILTER not update the job with all of the statements I defined?	665
How do I find my remote job?	665
Why won't my remote job run?	665
Why does llq -X all show no jobs running when there are jobs running?	666
Helpful hints	666
Scaling considerations	666
Hints for running jobs	666
Hints for using machines	669
History files and Schedd	670
Getting help from IBM	670

Appendix B. Sample command output 673

llclass -l command output listing	673
llq -l command output listing	674
llq -l command output listing for a Blue Gene enabled system	676
llq -l -x command output listing	676
llstatus -l command output listing	678
llstatus -l -b command output listing	679
llsummary -l -x command output listing	681

Appendix C. LoadLeveler port usage 685

Accessibility features for TWS

LoadLeveler 687

Accessibility features	687
Keyboard navigation	687
IBM and accessibility	687

Notices 689

Trademarks	691
----------------------	-----

Glossary 693

Index 697

Figures

1. Example of a LoadLeveler cluster	3	29. MPICH job command file - sample 2	193
2. LoadLeveler job steps	5	30. MPICH-GM job command file - sample 1	194
3. Multiple roles of machines	6	31. MPICH-GM job command file - sample 2	195
4. High-level job flow	14	32. MVAPICH job command file - sample 1	195
5. Job is submitted to LoadLeveler.	15	33. MVAPICH job command file - sample 2	196
6. LoadLeveler authorizes the job	16	34. Using LOADL_PROCESSOR_LIST in a shell script	197
7. LoadLeveler prepares to run the job	16	35. Building a job command file	215
8. LoadLeveler starts the job.	17	36. LoadLeveler build a job window	218
9. LoadLeveler completes the job	17	37. Format of administration file stanzas	302
10. How control expressions affect jobs	64	38. Format of administration file stanzas	302
11. Format of a machine stanza	80	39. Sample administration file stanzas	302
12. Format of an adapter stanza	82	40. Sample administration file stanza with user stanzas	303
13. Format of a class stanza	86	41. Serial job command file	334
14. Format of a user substanza	88	42. Main window of the LoadLeveler GUI	378
15. Format of a user stanza	91	43. Creating a new pull-down menu	382
16. Format of a group stanza	92	44. TWS LoadLeveler Blue Gene object model	524
17. Format of a cluster stanza.	93	45. TWS LoadLeveler Class object model	525
18. Multicluster Example	94	46. TWS LoadLeveler Cluster object model	525
19. Job command file with multiple steps	166	47. TWS LoadLeveler Fairshare object model	525
20. Job command file with multiple steps and one executable	167	48. TWS LoadLeveler Job object model	527
21. Job command file with varying input statements	167	49. TWS LoadLeveler Machine object model	528
22. Using LoadLeveler variables in a job command file	169	50. TWS LoadLeveler MCluster object model	528
23. Job command file used as the executable	170	51. TWS LoadLeveler Reservations object model	528
24. Striping over multiple networks	185	52. TWS LoadLeveler Wlmstat object model	529
25. Striping over a single network	186	53. Obtaining machine, job, and cluster information with the data access API	584
26. POE job command file – multiple tasks per node	192	54. Extracting job accounting information from a history file	588
27. POE sample job command file – invoking POE twice	192	55. When the primary central manager is unavailable	659
28. MPICH job command file - sample 1	193	56. Multiple central managers	659

Tables

1. Summary of typographic conventions	xiv	35. Multicluster support related topics	139
2. IBM System Blue Gene Solution documentation	xv	36. Subtasks for configuring a LoadLeveler multicluster	140
3. Topics in the TWS LoadLeveler overview	1	37. Blue Gene subtasks and associated instructions	145
4. Major topics in TWS LoadLeveler: Using and Administering	1	38. Blue Gene related topics and associated information	145
5. LoadLeveler daemons	7	39. Blue Gene configuring subtasks and associated instructions	145
6. startd determines whether its own state permits a new job to run	11	40. Learning about building and submitting jobs	163
7. Job state descriptions and abbreviations	18	41. Roadmap of user tasks for building and submitting jobs	165
8. Location and description of product directories following installation	30	42. Standard files for the five job steps	168
9. Location and description of directories for submit-only LoadLeveler	31	43. Checkpoint configurations	175
10. Roadmap of tasks for TWS LoadLeveler administrators	37	44. Parallel keywords supported by the LL_DEFAULT, BACKFILL, and API schedulers	179
11. Roadmap of administrator tasks related to using or modifying the LoadLeveler configuration file.	39	45. Valid combinations of task assignment keywords are listed in each column	180
12. Roadmap for defining LoadLeveler cluster characteristics	41	46. node and total_tasks	181
13. Default locations for all of the files and directories	45	47. Blocking	181
14. Log control statements	46	48. Unlimited blocking	182
15. Roadmap of configuration tasks for securing LoadLeveler operations	53	49. Roadmap of tasks for reservation owners and users	198
16. Roadmap of tasks for gathering job accounting data	58	50. Reservation states, abbreviations, and usage notes	198
17. Collecting account data - modifying the configuration file.	61	51. Instructions for submitting a job to run under a reservation.	203
18. Roadmap of administrator tasks accomplished through installation exits	66	52. Submitting and monitoring jobs in a LoadLeveler multicluster.	205
19. Roadmap of tasks for modifying the LoadLeveler administration file	77	53. Roadmap of user tasks for managing submitted jobs	211
20. Types of limit keywords	83	54. How LoadLeveler handles job priorities	213
21. Enforcing job step limits	84	55. User tasks available through the GUI	217
22. Setting limits	85	56. GUI fields and input	219
23. Roadmap of additional administrator tasks	95	57. Nodes dialog box	223
24. Roadmap of BACKFILL scheduler tasks	103	58. Network dialog box fields	224
25. Roadmap of tasks for using an external scheduler	106	59. Build a job dialog box fields	225
26. Effect of LoadLeveler keywords under an external scheduler	106	60. Limits dialog box fields	227
27. Roadmap of tasks for using preemption	116	61. Checkpointing dialog box fields	228
28. Preemption methods for which LoadLeveler automatically resumes preempted jobs	119	62. Blue Gene job fields	228
29. Preemption methods for which administrator or user intervention is required	119	63. Modifying the job command file with the Edit pull-down menu	230
30. Roadmap of reservation tasks for administrators	121	64. Modifying the job command file with the Tools pull-down menu	230
31. Roadmap of tasks for checkpointing jobs	129	65. Saving and submitting information	230
32. Deciding where to define the directory for staging executables	130	66. Sorting the jobs window	232
33. Linux platforms for CPU affinity	138	67. Sorting the machines window	236
34. Multicluster support subtasks and associated instructions	139	68. Specifying which jobs appear in the Jobs window	237
		69. Specifying which machines appear in Machines window	238
		70. Learning about LoadLeveler interfaces	241
		71. Configuration subtasks	243
		72. Administration file subtasks	301

73. Notes on 64-bit support for administration file keywords	305	90. FAIRSHARE specifications for ll_get_data subroutine	541
74. Summary of possible values set for the env_copy keyword in the administration file	313	91. JOBS specifications for ll_get_data subroutine	542
75. Sample user and group settings for the max_reservations keyword	322	92. MACHINES specifications for ll_get_data subroutine	566
76. Job command file subtasks	333	93. MCLUSTERS specifications for ll_get_data subroutine	571
77. Notes on 64-bit support for job command file keywords	335	94. RESERVATIONS specifications for ll_get_data subroutine	572
78. Example of a selection table.	379	95. WLMSTAT specifications for ll_get_data subroutine	573
79. Decision table	379	96. query_daemon summary.	574
80. Decision table actions.	380	97. query_flags summary.	580
81. Window identifiers in the Xloadl file	381	98. object_filter value related to the query flags value	581
82. Resource variables for all the windows and the buttons	381	99. enum LL_reservation_data type	605
83. Modifying help panels	383	100. How nodes should be arranged in the node list	647
84. LoadLeveler command summary	385	101. Why your job might not be running	652
85. llmodify options and keywords	438	102. Why your job might not be running	655
86. LoadLeveler API summary	503	103. Troubleshooting running jobs when a machine goes down	657
87. BLUE_GENE specifications for ll_get_data subroutine	533	104. LoadLeveler default port usage	685
88. CLASSES specifications for ll_get_data subroutine	537		
89. CLUSTER specifications for ll_get_data subroutine	540		

About this book

IBM® Tivoli® Workload Scheduler (TWS) LoadLeveler® provides various ways of scheduling and managing applications for best performance and most efficient use of resources. LoadLeveler manages both serial and parallel jobs over a cluster of machines or servers, which may be desktop workstations, dedicated servers, or parallel machines. This book describes how to configure and administer this LoadLeveler cluster environment, and to submit and manage jobs that run on machines in the cluster.

Note: IBM LoadLeveler has been rebranded and is now referred to as “Tivoli Workload Scheduler LoadLeveler” (short name: TWS LoadLeveler). This change is intended to extend the existing Tivoli Workload Scheduler functionality to better address the cluster and Grid computing environments and to increase the availability of LoadLeveler.

Who should use this book

This book is intended for two separate audiences:

- Personnel who are responsible for installing, configuring and managing the LoadLeveler cluster environment. These people are called LoadLeveler administrators. LoadLeveler administrative tasks include:
 - Setting up configuration and administration files
 - Maintaining the LoadLeveler product
 - Setting up the distributed environment for allocating batch jobs
- Users who submit and manage serial and parallel jobs to run in the LoadLeveler cluster.

Both LoadLeveler administrators and general users should be experienced with the UNIX® commands. Administrators also should be familiar with:

- Cluster system management techniques such as SMIT, as it is used in the AIX® environment
- Networking and NFS or AFS® protocols

Conventions and terminology used in this book

Throughout the TWS LoadLeveler product documentation:

- TWS LoadLeveler for Linux Multiplatform includes:
 - IBM System servers with AMD Opteron or Intel® EM64T processors
 - IBM System x™ servers
 - IBM BladeCenter® Intel processor-based servers
 - IBM Cluster 1350™

Note: IBM Tivoli Workload Scheduler LoadLeveler is supported when running Linux on non-IBM Intel-based and AMD hardware servers.

Supported hardware includes:

- Servers with Intel 32-bit and Intel Extended Memory 64 Technology (EM64T)
 - Servers with Advanced Micro Devices (AMD) 64-bit technology
- Note that in this document:

- LoadLeveler is also referred to as Tivoli Workload Scheduler LoadLeveler and TWS LoadLeveler.
- Switch_Network_Interface_For_HPS is also referred to as HPS or High Performance Switch.

Table 1 describes the typographic conventions used in this book.

Table 1. Summary of typographic conventions

Typographic	Usage
Bold	<ul style="list-style-type: none"> • Bold words or characters represent system elements that you must use literally, such as commands, flags, and path names. • Bold words also indicate the first use of a term included in the glossary.
<i>Italic</i>	<ul style="list-style-type: none"> • <i>Italic</i> words or characters represent variable values that you must supply. • <i>Italics</i> are also used for book titles and for general emphasis in text.
Constant width	Examples and information that the system displays appear in constant width typeface.
[]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.
	A vertical bar separates items in a list of choices. (In other words, it means “or.”)
< >	Angle brackets (less-than and greater-than) enclose the name of a key on the keyboard. For example, <Enter> refers to the key on your terminal or workstation that is labeled with the word Enter.
...	An ellipsis indicates that you can repeat the preceding item one or more times.
<Ctrl-x>	The notation <Ctrl-x> indicates a control character sequence. For example, <Ctrl-c> means that you hold down the control key while pressing <c>.
\	The continuation character is used in coding examples in this book for formatting purposes.

Prerequisite and related information

The Tivoli Workload Scheduler LoadLeveler publications are:

- *Installation Guide*, GI10-0763
- *Using and Administering*, SA22-7881
- *Diagnosis and Messages Guide*, GA22-7882

To access all TWS LoadLeveler documentation, refer to the **IBM Cluster Information Center**, which contains the most recent TWS LoadLeveler documentation in PDF and HTML formats. This Web site is located at:

<http://publib.boulder.ibm.com/infocenter/clresctr/vrx/index.jsp>

A **LoadLeveler Documentation Updates** file also is maintained on this Web site. The **LoadLeveler Documentation Updates** file contains updates to the TWS LoadLeveler documentation. These updates include documentation corrections and clarifications that were discovered after the TWS LoadLeveler books were published.

Both the current TWS LoadLeveler books and earlier versions of the library are also available in PDF format from the IBM Publications Center Web site located at: <http://www.ibm.com/shop/publications/order>

To easily locate a book in the IBM Publications Center, supply the book's publication number. The publication number for each of the TWS LoadLeveler books is listed after the book title in the preceding list.

IBM System Blue Gene Solution documentation

Table 2 lists the IBM System Blue Gene[®] Solution publications that are available from the IBM Redbooks[™] Web site at the following URLs:

Table 2. IBM System Blue Gene Solution documentation

Publication Name	URL
<i>Blue Gene/L: System Administration</i>	http://www.redbooks.ibm.com/abstracts/sg247178.html
<i>Blue Gene/L: Hardware Overview and Planning</i>	http://www.redbooks.ibm.com/abstracts/sg246796.html
<i>Blue Gene/L: Application Development</i>	http://www.redbooks.ibm.com/abstracts/sg247179.html
<i>Unfolding the IBM eServer Blue Gene Solution</i>	http://www.redbooks.ibm.com/abstracts/sg246686.html

Using LookAt to look up message explanations

LookAt is an online facility that lets you look up explanations for most of the IBM messages you encounter, as well as for some system abends and codes. You can use LookAt from the following locations to find IBM message explanations for Clusters for AIX and Linux[®]:

- The Internet. You can access IBM message explanations directly from the LookAt Web site:
<http://www.ibm.com/eserver/zseries/zos/bkserv/lookat/>
- Your wireless handheld device. You can use the LookAt Mobile Edition with a handheld device that has wireless access and an Internet browser (for example, Internet Explorer for Pocket PCs, Blazer, or Eudora for Palm OS, or Opera for Linux handheld devices). Link to the LookAt Mobile Edition from the LookAt Web site.

How to send your comments

Your feedback is important in helping us to produce accurate, high-quality information. If you have any comments about this book or any other TWS LoadLeveler documentation:

- Send your comments by e-mail to: mhvrcfs@us.ibm.com
Include the book title and order number, and, if applicable, the specific location of the information you have comments on (for example, a page number or a table number).
- Fill out one of the forms at the back of this book and return it by mail, by fax, or by giving it to an IBM representative.

To contact the IBM cluster development organization, send your comments by e-mail to: cluster@us.ibm.com

Summary of changes

The following sections summarize changes to the IBM Tivoli Workload Scheduler (TWS) LoadLeveler product and TWS LoadLeveler library for each new release or major service update for a given product version. Within each book in the library, a vertical line to the left of text and illustrations indicates technical changes or additions made to the previous edition of the book.

Changes to TWS LoadLeveler for this release or update include:

- **New information:**

- Enhanced job spool recovery has been added:
 - The **llmovespool** command has been added to move the job records from the spool of one managing Schedd to another managing Schedd in the local cluster. For more information, see “llmovespool - Move job records” on page 442.
 - The Move Spool menu item has been added to the Machines window **Admin** pull-down menu in the graphical user interface (GUI) to allow you to move job spool records from the specified spool directory to the Schedd daemon that is running on the selected machine.
- A new communication scheme has been introduced in TWS LoadLeveler to provide reliable and fast job launch performance and improved scalability.
- Multiple top-dog support has been incorporated into the TWS LoadLeveler BACKFILL scheduler, which allows one or more top dogs to be allocated a future start time. The TWS LoadLeveler scheduler will determine that there are insufficient resources for some jobs to start, and for those jobs, it will compute the earliest future start times. The resources will be allocated to the top-dog jobs and remaining jobs in the queue will be backfilled so that top-dog jobs are not delayed.

For more information, see “Using the BACKFILL scheduler” on page 101.

- Coscheduling function has been added to:
 - Enable TWS LoadLeveler to coschedule multiple job steps while placing minimal restrictions on the type and number of resource requirements that can be specified within the coscheduled job steps
 - Allow coscheduled job steps to take advantage of the current state-of-the-art job scheduling paradigms, such as top-dog reservation, BACKFILL, preemption, and reservations, which are presently supported in the TWS LoadLeveler product

For more information, see “Working with coscheduled job steps” on page 171.

- The following TWS LoadLeveler Blue Gene support has been enhanced, including:
 - Advance reservations has been extended to include the reservation of Blue Gene resources. For more information, see “Blue Gene advance reservation support” on page 147.
 - Fair share scheduling has been extended to include Blue Gene resources. The Blue Gene resource usage is an additional set of information that is made available by TWS LoadLeveler, which can be used to influence job priorities. For more information, see “Blue Gene fair share scheduling support” on page 147.

- TWS LoadLeveler now allows users to request Blue Gene compute nodes (C-nodes) with a certain amount of memory in the same way as TWS LoadLeveler currently allows for non-Blue Gene nodes. For more information, see “Blue Gene heterogeneous memory support” on page 147.
- Preemption support for Blue Gene jobs has been enabled, so Blue Gene jobs now have the same preemption support as non-Blue Gene jobs.
- The TWS LoadLeveler product has been extended to include support for InfiniBand adapters in SUSE Linux Enterprise Server (SLES) 9 on POWER TWS LoadLeveler clusters. InfiniBand adapters on any other platform are not supported. This support does not place new constraints on the submit nodes or where the central manager runs. Preemption is not supported on TWS LoadLeveler for Linux when running in an InfiniBand cluster environment. For more information, see “Configuring InfiniBand adapters” on page 81.
- Support for SLES 10 has been added.

Note: SLES 10 is not supported on the Myrinet switch.

- MPICH job management has been integrated with TWS LoadLeveler. Operations allowed for Parallel Environment (PE) parallel jobs are supported for MPICH parallel jobs. For more information, see “Configuring LoadLeveler to support MPICH jobs” on page 99.
- The **smt** keyword has been introduced in the TWS LoadLeveler job command file to enable or disable dynamic simultaneous multithreading (SMT). This keyword can also be set in the job class stanza in the TWS LoadLeveler administration file so that all parallel jobs submitted in that class will run with SMT on or off by default.
For more information, see the **smt** keyword in the Chapter 14, “Job command file reference,” on page 333.
- The restriction on the maximum number of reservations allowed in a cluster has been removed.
- OpenAFS 1.4.1 is now supported on the RedHat Enterprise Linux 3.0 and RedHat Enterprise Linux 4.0 distributions on the x86 and x86_64 platforms. If you are using TWS LoadLeveler with OpenAFS 1.2.11 on RedHat Enterprise Linux 3.0 you will need to upgrade to OpenAFS 1.4.1 when you upgrade to this version of TWS LoadLeveler.
- In this release of TWS LoadLeveler, we are offering you an opportunity to preview our first steps toward a Web-based user interface with sample programs. Your input is invaluable in our efforts to revitalize and modernize our user interface. For more information, see the README file for the sample in the **samples/llwebui** subdirectory of the release directory.
- A new topic has been added to help you find information about quickly starting TWS LoadLeveler using the default configuration. For more information, see Chapter 2, “Getting a quick start using the default configuration,” on page 27.
- A new topic has been added to help you find TWS LoadLeveler port usage information. For more information, see Appendix C, “LoadLeveler port usage,” on page 685.

- **Changed information:**

- Release 3 of TWS LoadLeveler no longer supports switches other than the SP Switch2 and the High Performance Switch.
- In previous releases of TWS LoadLeveler, when a job is submitted it automatically gets a **requirements** statement defined for the Arch and Opsy

LoadLeveler reserved words. The values assigned are determined by the operating system and node where the submission parsing was done. For example:

```
Requirements: (Arch == "R6000") && (OpSys == "AIX52")
```

In this release of TWS LoadLeveler, this automatic definition of the **requirements** keyword has been removed. Administrators still need to use the **lldbconvert** utility to convert jobs from TWS LoadLeveler 3.3 to the TWS LoadLeveler 3.4 format. See “lldbconvert - Job migration utility” on page 414 for more information.

- This is the last release that will provide the following functions:
 - The scheduling of parallel jobs with the default scheduler (**SCHEDULER_TYPE=LL_DEFAULT**)
 - The following API functions: **ll_get_nodes**, **ll_get_jobs**, and **ll_start_job**
- **Deleted information:**

The following function is no longer supported and the information has been removed:

- Gang scheduler
- Network queuing system (NQS)
- SP Switch (csss, css0, and css1)
- Distributed Computing Environment (DCE)
- Parallel Job API
- Parallel System Support Programs (PSSP)

Part 1. Overview of TWS LoadLeveler concepts and operation

Setting up IBM Tivoli Workload Scheduler (TWS) LoadLeveler involves defining machines, users, jobs, and how they interact, in such a way that TWS LoadLeveler is able to run jobs quickly and efficiently. If you are unfamiliar with the TWS LoadLeveler product, consider reading one or more of the introductory topics listed in Table 3.

Table 3. Topics in the TWS LoadLeveler overview

To learn about:	Read the following:
TWS LoadLeveler interfaces, operations, and the lifecycle of a job	Chapter 1, "What is LoadLeveler?," on page 3
Using the default configuration for getting a quick start	Chapter 2, "Getting a quick start using the default configuration," on page 27
Specific products and features that are required for or available through the TWS LoadLeveler environment	Chapter 3, "What operating systems are supported by LoadLeveler?," on page 33

Once you have a basic understanding of the TWS LoadLeveler product and its interfaces, you can find more details in the topics listed in Table 4.

Table 4. Major topics in TWS LoadLeveler: Using and Administering

To learn about:	Read the following:
Performing administrator tasks	Part 2, "Configuring and managing the TWS LoadLeveler environment," on page 37
Performing general user tasks	Part 3, "Submitting and managing TWS LoadLeveler jobs," on page 163
Using TWS LoadLeveler interfaces	Part 4, "TWS LoadLeveler interfaces reference," on page 241

Chapter 1. What is LoadLeveler?

LoadLeveler is a job management system that allows users to run more jobs in less time by matching the jobs' processing needs with the available resources. LoadLeveler schedules jobs, and provides functions for building, submitting, and processing jobs quickly and efficiently in a dynamic environment.

Figure 1 shows the different environments to which LoadLeveler can schedule jobs. Together, these environments comprise the *LoadLeveler cluster*.

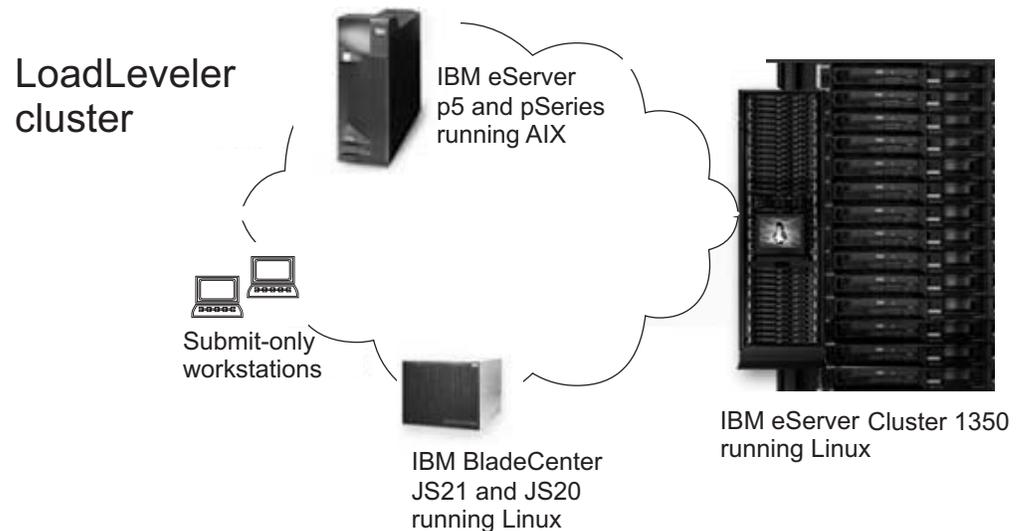


Figure 1. Example of a LoadLeveler cluster

As Figure 1 also illustrates, a LoadLeveler cluster can include *submit-only* machines, which allow users to have access to a limited number of LoadLeveler features.

Throughout all the topics, the terms *workstation*, *machine*, *node*, and *operating system instance (OSI)* refer to the machines in your cluster. In LoadLeveler, an OSI is treated as a single instance of an operating system image.

LoadLeveler basics

LoadLeveler has various types of interfaces that enable users to create and submit jobs and allow system administrators to configure the system and control running jobs. These interfaces include:

- Control files that define the elements, characteristics, and policies of LoadLeveler and the jobs it manages. These files are the configuration file, the administration file, and job command file.
- The command line interface, which gives you access to basic job and administrative functions.
- A graphical user interface (GUI), which provides system access similar to the command line interface. Experienced users and administrators may find the command line interface more efficient than the GUI for job and administrative functions.

LoadLeveler basics

- An application programming interface (API), which allows application programs written by users and administrators to interact with the LoadLeveler environment.

The commands, GUI, and APIs permit different levels of access to administrators and users. User access is typically restricted to submitting and managing individual jobs, while administrative access allows setting up system configurations, job scheduling, and accounting.

Using either the command line or the GUI, users create job command files that instruct the system on how to process information. Each job command file consists of keywords followed by the user defined association for that keyword. For example, the keyword **executable** tells LoadLeveler that you are about to define the name of a program you want to run. Therefore, **executable = longjob** tells LoadLeveler to run the program called *longjob*.

After creating the job command file, you invoke LoadLeveler commands to monitor and control the job as it moves through the system. LoadLeveler monitors each job as it moves through the system using process control daemons. However, the administrator maintains ultimate control over all LoadLeveler jobs by defining job classes that control how and when LoadLeveler will run a job.

In addition to setting up job classes, the administrator can also control how jobs move through the system by specifying the type of scheduler. LoadLeveler has several different scheduler options that start jobs using specific algorithms to balance job priority with available machine resources.

When LoadLeveler administrators are configuring clusters and when users are planning jobs, they need to be aware of the machine resources available in the cluster. These resources include items like the number of CPUs and the amount of memory available for each job. Because resource availability will vary over time, LoadLeveler defines them as consumable resources.

LoadLeveler: A network job management and scheduling system

A network job management and job scheduling system, such as LoadLeveler, is a software program that schedules and manages jobs that you submit to one or more machines under its control. LoadLeveler accepts jobs that users submit and reviews the job requirements. LoadLeveler then examines the machines under its control to determine which machines are best suited to run each job.

Job definition

LoadLeveler schedules your jobs on one or more machines for processing. The definition of a **job**, in this context, is a set of **job steps**. For each job step, you can specify a different executable (the executable is the part of the job that gets processed). You can use LoadLeveler to submit jobs which are made up of one or more job steps, where each job step depends upon the completion status of a previous job step. For example, Figure 2 on page 5 illustrates a stream of job steps:

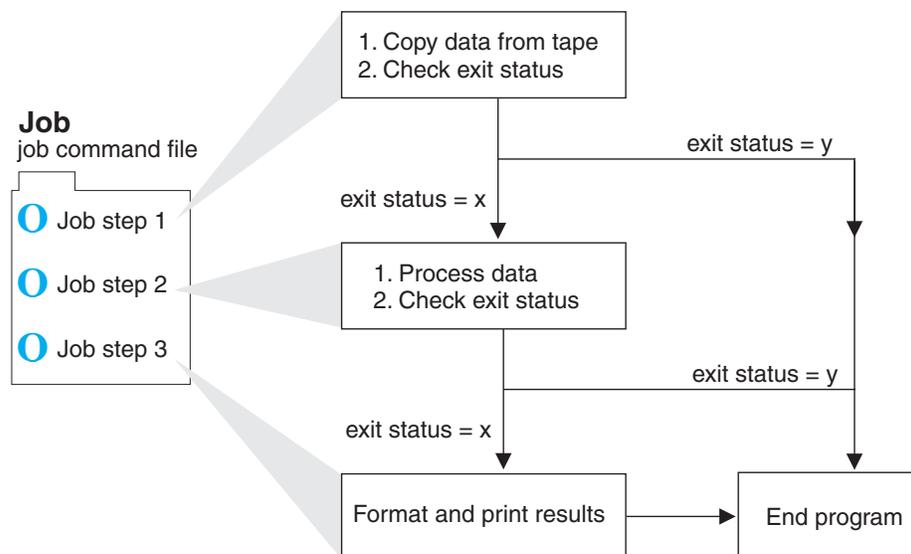


Figure 2. LoadLeveler job steps

Each of these job steps is defined in a single **job command file**. A job command file specifies the name of the job, as well as the job steps that you want to submit, and can contain other LoadLeveler statements.

LoadLeveler tries to execute each of your job steps on a machine that has enough resources to support executing and checkpointing each step. If your job command file has multiple job steps, the job steps will not necessarily run on the same machine, unless you explicitly request that they do.

You can submit batch jobs to LoadLeveler for scheduling. Batch jobs run in the background and generally do not require any input from the user. Batch jobs can either be **serial** or **parallel**. A serial job runs on a single machine. A parallel job is a program designed to execute as a number of individual, but related, processes on one or more of your system's nodes. When executed, these related processes can communicate with each other (through message passing or shared memory) to exchange data or synchronize their execution.

For parallel jobs, LoadLeveler interacts with Parallel Operating Environment (POE) to allocate nodes, assign tasks to nodes, and launch tasks.

Machine definition

For LoadLeveler to schedule a job on a machine, the machine must be a valid member of the LoadLeveler cluster. A cluster is the combination of all of the different types of machines that use LoadLeveler.

To make a machine a member of the LoadLeveler cluster, the administrator has to install the LoadLeveler software onto the machine and identify the central manager (described in "Roles of machines"). Once a machine becomes a valid member of the cluster, LoadLeveler can schedule jobs to it.

Roles of machines

Each machine in the LoadLeveler cluster performs one or more roles in scheduling jobs:

- **Scheduling Machine:** When a job is submitted, it gets placed in a queue managed by a scheduling machine. This machine contacts another machine that

LoadLeveler basics

serves as the central manager for the entire LoadLeveler cluster. This scheduling machine asks the central manager to find a machine that can run the job, and also keeps persistent information about the job. Some scheduling machines are known as *public scheduling machines*, meaning that any LoadLeveler user can access them. These machines schedule jobs submitted from submit-only machines:

- **Central Manager Machine:** The role of the central manager is to examine the job's requirements and find one or more machines in the LoadLeveler cluster that will run the job. Once it finds the machine(s), it notifies the scheduling machine.
- **Executing Machine:** The machine that runs the job is known as the executing machine.
- **Submitting Machine:** This type of machine is known as a *submit-only* machine. It participates in the LoadLeveler cluster on a limited basis. Although the name implies that users of these machines can only submit jobs, they can also query and cancel jobs. Users of these machines also have their own Graphical User Interface (GUI) that provides them with the submit-only subset of functions. The submit-only machine feature allows workstations that are not part of the LoadLeveler cluster to submit jobs to the cluster.

Keep in mind that one machine can assume multiple roles, as shown in Figure 3.

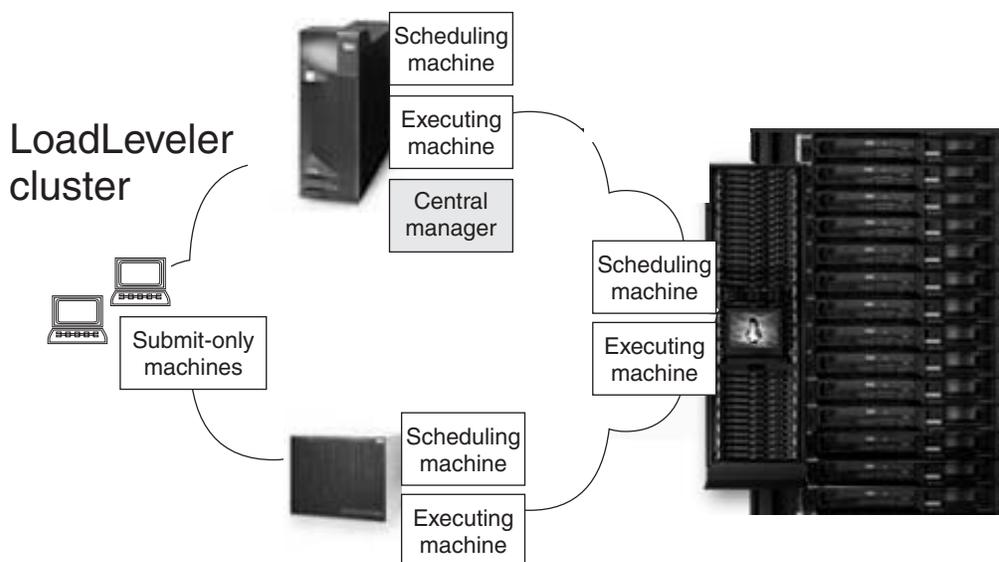


Figure 3. Multiple roles of machines

Machine availability

There may be times when some of the machines in the LoadLeveler cluster are not available to process jobs; for instance, when the owners of the machines have decided to make them unavailable. This ability of LoadLeveler to allow users to restrict the use of their machines provides flexibility and control over the resources.

Machine owners can make their personal workstations available to other LoadLeveler users in several ways. For example, you can specify that:

- The machine will always be available
- The machine will be available only between certain hours
- The machine will be available when the keyboard and mouse are not being used interactively.

Owners can also specify that their personal workstations never be made available to other LoadLeveler users.

How LoadLeveler schedules jobs

When a user submits a job, LoadLeveler examines the job command file to determine what resources the job will need. LoadLeveler determines which machine, or group of machines, is best suited to provide these resources, then LoadLeveler dispatches the job to the appropriate machines. To aid this process, LoadLeveler uses queues. A **job queue** is a list of jobs that are waiting to be processed. When a user submits a job to LoadLeveler, the job is entered into an internal database— which resides on one of the machines in the LoadLeveler cluster— until it is ready to be dispatched to run on another machine.

Once LoadLeveler examines a job to determine its required resources, the job is dispatched to a machine to be processed. A job can be dispatched to either one machine, or— in the case of parallel jobs— to multiple machines. Once the job reaches the executing machine, the job runs.

Jobs do not necessarily get dispatched to machines in the cluster on a first-come, first-serve basis. Instead, LoadLeveler examines the requirements and characteristics of the job and the availability of machines, and then determines the best time for the job to be dispatched.

LoadLeveler also uses **job classes** to schedule jobs to run on machines. A job class is a classification to which a job can belong. For example, short running jobs may belong to a job class called `short_jobs`. Similarly, jobs that are only allowed to run on the weekends may belong to a class called `weekend`. The system administrator can define these job classes and select the users that are authorized to submit jobs of these classes.

You can specify which types of jobs will run on a machine by specifying the types of job classes the machine will support. LoadLeveler also examines a job's **priority** to determine when to schedule the job on a machine. A priority of a job is used to determine its position among a list of all jobs waiting to be dispatched.

"The LoadLeveler job cycle" on page 14 describes job flow in the LoadLeveler environment in more detail.

How LoadLeveler daemons process jobs

LoadLeveler has its own set of daemons that control the processes moving jobs through the LoadLeveler cluster. The LoadLeveler daemons are programs that run continuously and control the processes that move jobs through the LoadLeveler cluster. A master daemon (**LoadL_master**) runs on all machines in the LoadLeveler cluster and manages other daemons.

Table 5 summarizes these daemons, which are described in further detail in topics immediately following the table.

Table 5. LoadLeveler daemons

Daemon	Description
LoadL_master	Referred to as the master daemon. Runs on all machines in the LoadLeveler cluster and manages other daemons.

LoadLeveler daemons

Table 5. LoadLeveler daemons (continued)

Daemon	Description
LoadL_schedd	Referred to as the Schedd daemon. Receives jobs from the llsubmit command and manages them on machines selected by the negotiator daemon (as defined by the administrator).
LoadL_startd	Referred to as the startd daemon. Monitors job and machine resources on local machines and forwards information to the negotiator daemon. The startd daemon spawns the starter process (LoadL_starter) which manages running jobs on the executing machine.
LoadL_negotiator	Referred to as the negotiator daemon. Monitors the status of each job and machine in the cluster. Responds to queries from llstatus and llq commands. Runs on the central manager machine.
LoadL_kbdd	Referred to as the keyboard daemon. Monitors keyboard and mouse activity.
LoadL_GSmonitor	Referred to as the gsmonitor daemon. Monitors for down machines based on the heartbeat responses of the MACHINE_UPDATE_INTERVAL time period.

The master daemon

The **master** daemon runs on every machine in the LoadLeveler cluster, except the submit-only machines. The real and effective user ID of this daemon must be root.

The master daemon determines whether to start any other daemons by checking the **START_DAEMONS** keyword in the global or local configuration file. If the keyword is set to **true**, the daemons are started. If the keyword is set to **false**, the master daemon terminates and generates a message.

The master daemon will not start on a Linux machine if **SEC_ENABLEMENT** is set to **CTSEC**. If the master daemon does not start, no other daemons will start.

On the machine designated as the central manager, the master runs the **negotiator** daemon. The master also controls the central manager backup function. The negotiator runs on either the primary or an alternate central manager. If a central manager failure is detected, one of the alternate central managers becomes the primary central manager by starting the negotiator.

The master daemon starts and if necessary, restarts all the LoadLeveler daemons that the machine it resides on is configured to run. As part of its startup procedure, this daemon executes the **.llrc** file (a dummy file is provided in the **bin** subdirectory of the release directory). You can use this script to customize your local configuration file, specifying what particular data is stored locally. This daemon also runs the **kbdd** daemon, which monitors keyboard and mouse activity.

When the master daemon detects a failure on one of the daemons that it is monitoring, it attempts to restart it. Because this daemon recognizes that certain situations may prevent a daemon from running, it limits its restart attempts to the number defined for the **RESTARTS_PER_HOUR** keyword in the configuration file. If this limit is exceeded, the master daemon forces all daemons including itself to exit.

When a daemon must be restarted, the master sends mail to the administrators identified by the **LOADL_ADMIN** keyword in the configuration file. The mail contains the name of the failing daemon, its termination status, and a section of the daemon's most recent log file. If the master aborts after exceeding **RESTARTS_PER_HOUR**, it will also send that mail before exiting.

The master daemon may perform the following actions in response to an **llctl** command:

- Kill all daemons and exit (**stop** keyword)
- Kill all daemons and execute a new master (**recycle** keyword)
- Rerun the **.llrc** file, reread the configuration files, stop or start daemons as appropriate for the new configuration files (**reconfig** keyword)
- Send drain request to startd and (**drain** keyword)
- Send flush request to startd and send result to caller (**flush** keyword)
- Send suspend request to startd and send result to caller (**suspend** keyword)
- Send resume request to startd and Schedd, and send result to caller (**resume** keyword)

The Schedd daemon

The **Schedd** daemon receives jobs sent by the **llsubmit** command and manages those jobs to machines selected by the negotiator daemon. The Schedd daemon is started, restarted, signalled, and stopped by the master daemon.

The Schedd daemon can be in any one of the following activity states:

Available	This machine is available to schedule jobs.
Drained	The Schedd machine accepts no more jobs. There are no jobs in starting or running state. Jobs in the Idle state are drained, meaning they will not get dispatched.
Draining	The Schedd daemon is being drained by the administrator but some jobs are still running. The state of the machine remains Draining until all running jobs complete. At that time, the machine status changes to Drained.
Down	The daemon is not running on this machine. The Schedd daemon enters this state when it has not reported its status to the negotiator. This can occur when the machine is actually down, or because there is a network failure.

The Schedd daemon performs the following functions:

- Assigns new job identifiers when requested by the job submission process (for example, by the **llsubmit** command).
- Receives new jobs from the **llsubmit** command. A new job is received as a *job object* for each job step. A job object is the data structure in memory containing all the information about a job step. The Schedd forwards the job object to the negotiator daemon as soon as it is received from the submit command.
- Maintains on disk copies of jobs submitted locally (on this machine) that are either waiting or running on a remote (different) machine. The central manager can use this information to reconstruct the job information in the event of a failure. This information is also used for accounting purposes.
- Responds to directives sent by the administrator through the negotiator daemon. The directives include:
 - Run a job.
 - Change the priority of a job.

LoadLeveler daemons

- Remove a job.
- Hold or release a job.
- Send information about all jobs.
- Sends job events to the negotiator daemon when:
 - Schedd is restarting.
 - A new series of job objects are arriving.
 - A job is started.
 - A job was rejected, completed, removed, or vacated. Schedd determines the status by examining the exit status returned by the startd.
- Communicates with the Parallel Operating Environment (POE) when you run an interactive POE job.
- Requests that a remote startd daemon end a job.
- Receives accounting information from startd.
- Receives requests for reservations.
- Collects resource usage data when jobs terminate and stores it as historic fair share data in the \$(SPOOL) directory.
- Sends historic fair share data to the central manager when it is updated or when the **Schedd** daemon is restarted.
- Maintains and stores records of historic CPU and Blue Gene utilization for users and groups known to the Schedd.
- Passes the historic CPU and Blue Gene utilization data to the central manager.

The startd daemon

The **startd** daemon monitors the status of each job, reservation, and machine in the cluster, and forwards this information to the negotiator daemon. The startd also receives and executes job requests originating from remote machines. The master daemon starts, restarts, signals, and stops the startd daemon.

Checkpoint/restart is not supported in LoadLeveler for Linux. If a checkpointed job is sent to a Linux node, the Linux node will reject the job.

The startd daemon can be in any one of the following states:

Busy	The maximum number of jobs are running on this machine as specified by the MAX_STARTERS configuration keyword.
Down	The daemon is not running on this machine. The startd daemon enters this state when it has not reported its status to the negotiator. This can occur when the machine is actually down, or because there is a network failure.
Drained	The startd machine will not accept any new jobs. No jobs are running when startd is in the drained state.
Draining	The startd daemon is being drained by the administrator, but some jobs are still running. The machine remains in the draining state until all of the running jobs have completed, at which time the machine status changes to drained. The startd daemon will not accept any new jobs while in the draining state.
Flush	Any running jobs have been vacated (terminated and returned to the queue to be redispached). The startd daemon will not accept any new jobs.
Idle	The machine is not running any jobs.
None	LoadLeveler is running on this machine, but no jobs can run here.

- Running** The machine is running one or more jobs and is capable of running more.
- Suspend** All LoadLeveler jobs running on this machine are stopped (cease processing), but remain in virtual memory. The startd daemon will not accept any new jobs.

The startd daemon performs these functions:

- Runs a time-out procedure that includes building a snapshot of the state of the machine that includes static and dynamic data. This time-out procedure is run at the following times:
 - After a job completes.
 - According to the definition of the **POLLING_FREQUENCY** keyword in the configuration file.
- Records the following information in LoadLeveler variables and sends the information to the negotiator.
 - State (of the startd daemon)
 - EnteredCurrentState
 - Memory
 - Disk
 - KeyboardIdle
 - Cpus
 - LoadAvg
 - Machine
 - Adapter
 - AvailableClasses
- Calculates the **SUSPEND**, **RESUME**, **CONTINUE**, and **VACATE** expressions through which you can manage job status.
- Receives job requests from the Schedd daemon to:
 - Start a job
 - Preempt or resume a job
 - Vacate a job
 - Cancel

When the Schedd daemon tells the startd daemon to start a job, the startd determines whether its own state permits a new job to run:

Table 6. startd determines whether its own state permits a new job to run

If:	Then this happens:
Yes, it can start a new job	The startd forks a starter process.
No, it cannot start a new job	The startd rejects the request for one of the following reasons: <ul style="list-style-type: none"> • Jobs have been suspended, flushed, or drained • The job limit set for the MAX_STARTERS keyword has been reached • There are not enough classes available for the designated job class

- Receives requests from the master (through the **llctl** command) to do one of the following:
 - Drain (**drain** keyword)
 - Flush (**flush** keyword)
 - Suspend (**suspend** keyword)
 - Resume (**resume** keyword)

LoadLeveler daemons

- For each request, startd marks its own new state, forwards its new state to the negotiator daemon, and then performs the appropriate action for any jobs that are active.
- Receives notification of keyboard and mouse activity from the kbdd daemon
- Periodically examines the process table for LoadLeveler jobs and accumulates resources consumed by those jobs. This resource data is used to determine if a job has exceeded its job limit and for recording in the history file.
- Send accounting information to Schedd.

The starter process

The startd daemon spawns a **starter** process after the Schedd daemon tells the startd daemon to start a job. The starter process manages all the processes associated with a job step. The starter process is responsible for running the job and reporting status back to the startd daemon.

The starter process performs these functions:

- Processes the prolog and epilog programs as defined by the **JOB_PROLOG** and **JOB_EPILOG** keywords in the configuration file. The job will not run if the prolog program exits with a return code other than zero.
- Handles authentication. This includes:
 - Authenticates AFS, if necessary
 - Verifies that the submitting user is *not* root
 - Verifies that the submitting user has access to the appropriate directories in the local file system.
- Runs the job by forking a child process that runs with the user ID and all groups of the submitting user. That child process creates a new process group of which it is the process group leader, and executes the user's program or a shell.

The starter process is responsible for detecting the termination of any process that it forks. To ensure that all processes associated with a job are terminated after the process forked by the starter terminates, process tracking must be enabled. To configure LoadLeveler for process tracking, see "Tracking job processes" on page 64.
- Responds to vacate and suspend orders from the startd.

The negotiator daemon

The **negotiator** daemon maintains status of each job and machine in the cluster and responds to queries from the **llstatus** and **llq** commands. The negotiator daemon runs on a single machine in the cluster (the central manager machine). This daemon is started, restarted, signalled, and stopped by the master daemon.

In a mixed cluster, the negotiator daemon must run on an AIX node.

The negotiator daemon receives status messages from each Schedd and startd daemon running in the cluster. The negotiator daemon tracks:

- Which Schedd daemons are running
- Which startd daemons are running, and the status of each startd machine.

If the negotiator does not receive an update from any machine within the time period defined by the **MACHINE_UPDATE_INTERVAL** keyword, then the negotiator assumes that the machine is down, and therefore the Schedd and startd daemons are also down.

The negotiator also maintains in its memory several queues and tables which determine where the job should run.

The negotiator performs the following functions:

- Receives and records job status changes from the Schedd daemon.
- Schedules jobs based on a variety of scheduling criteria and policy options. Once a job is selected, the negotiator contacts the Schedd that originally created the job.
- Handles requests to:
 - Set priorities
 - Query about jobs, machines, classes, and reservations
 - Change reservation attributes
 - Bind jobs to reservations
 - Remove a reservation
 - Remove a job
 - Hold or release a job
 - Favor or unfavor a user or a job.
- Receives notification of Schedd resets indicating that a Schedd has restarted.

The kbdd daemon

The kbdd daemon monitors keyboard and mouse activity. The kbdd daemon is spawned by the master daemon if the `X_RUNS_HERE` keyword in the configuration file is set to `true`.

The kbdd daemon notifies the startd daemon when it detects keyboard or mouse activity; however, kbdd is *not* interrupt driven. It sleeps for the number of seconds defined by the `POLLING_FREQUENCY` keyword in the LoadLeveler configuration file, and then determines if X events, in the form of mouse or keyboard activity, have occurred. For more information on the configuration file, see Chapter 5, “Defining LoadLeveler resources to administer,” on page 77.

The gsmonitor daemon

The gsmonitor daemon is not available in LoadLeveler for Linux.

The negotiator daemon monitors for down machines based on the heartbeat responses of the `MACHINE_UPDATE_INTERVAL` time period. If the negotiator has not received an update after two `MACHINE_UPDATE_INTERVAL` periods, then it marks the machine as down, and notifies the Schedd to remove any jobs running on that machine. The gsmonitor daemon (`LoadL_GSmonitor`) allows this cleanup to occur more reliably. The gsmonitor daemon uses the Group Services Application Programming Interface (GSAPI) to monitor machine availability on peer domains and to notify the negotiator quickly when a machine is no longer reachable.

| If the `GSMONITOR_DOMAIN` keyword was not specified in the LoadLeveler
| configuration file, then LoadLeveler will try to determine if the machine is running
| in a peer (cluster) domain. The gsmonitor must run in a peer domain. The
| gsmonitor will detect that it is running in an active peer domain, then it will use
| the RMC API to determine the node numbers and names of machines running in
| the cluster.

If the administrator sets up a LoadLeveler administration file that contains OSIs spanning several peer domains then a gsmonitor daemon must be started in each domain. A gsmonitor daemon can monitor only the OSIs contained in the domain within which it is running. The administrator specifies which OSIs run the

LoadLeveler daemons

gsmonitor daemon by specifying `GSMONITOR_RUNS_HERE=TRUE` in the local configuration file for that OSI. The default for `GSMONITOR_RUNS_HERE` is False.

The gsmonitor daemon should be run on one or two nodes in the peer domain. By running `LoadL_GSmonitor` on more than one node in a domain you will have a backup in case one of the nodes that the monitor is running on goes down. `LoadL_GSmonitor` subscribes to the Group Services system-defined host membership group, which is represented by the `HA_GS_HOST_MEMBERSHIP` Group Services keyword. This group monitors every configured node in the system partition and every node in the active peer domain.

Notes:

1. The Group Services routines need to be run as root, so the `LoadL_GSmonitor` executable must be owned by root and have the setuid permission bit enabled.
2. It will not cause a problem to run more than one `LoadL_GSmonitor` daemon per peer domain, this will just cause the negotiator to be notified by each running daemon.
3. For more information about the Group Services subsystem, see the *RSCT Administration Guide*, SA22-7889 for peer domains.
4. For more information about GSAPI, see *Group Services Programming Guide and Reference*, SA22-7355.

The LoadLeveler job cycle

The following description and sequence of diagrams illustrate the flow of job information through the LoadLeveler cluster.

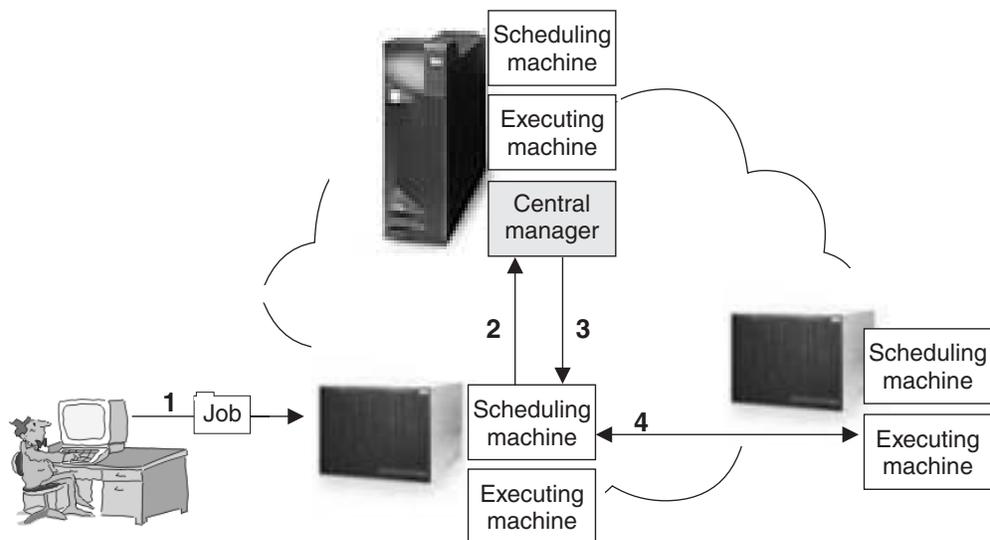


Figure 4. High-level job flow

The managing machine in a LoadLeveler cluster is known as the **central manager**. There are also machines that act as schedulers, and machines that serve as the executing machines. The arrows in Figure 4 illustrate the following:

- Arrow 1 indicates that a job has been submitted to LoadLeveler.
- Arrow 2 indicates that the scheduling machine contacts the central manager to inform it that a job has been submitted, and to find out if a machine exists that matches the job requirements.

- Arrow 3 indicates that the central manager checks to determine if a machine exists that is capable of running the job. Once a machine is found, the central manager informs the scheduling machine which machine is available.
- Arrow 4 indicates that the scheduling machine contacts the executing machine and provides it with information regarding the job. In this case, the scheduling and executing machines are different machines in the cluster, but they do not have to be different; the scheduling and executing machines may be the same physical machine.

Figure 4 on page 14 is broken down into the following more detailed diagrams illustrating how LoadLeveler processes a job. The diagrams indicate specific job states for this example, but do not list all of the possible states for LoadLeveler jobs. A complete list of job states appears in “LoadLeveler job states” on page 18.

1. Submit a LoadLeveler job:

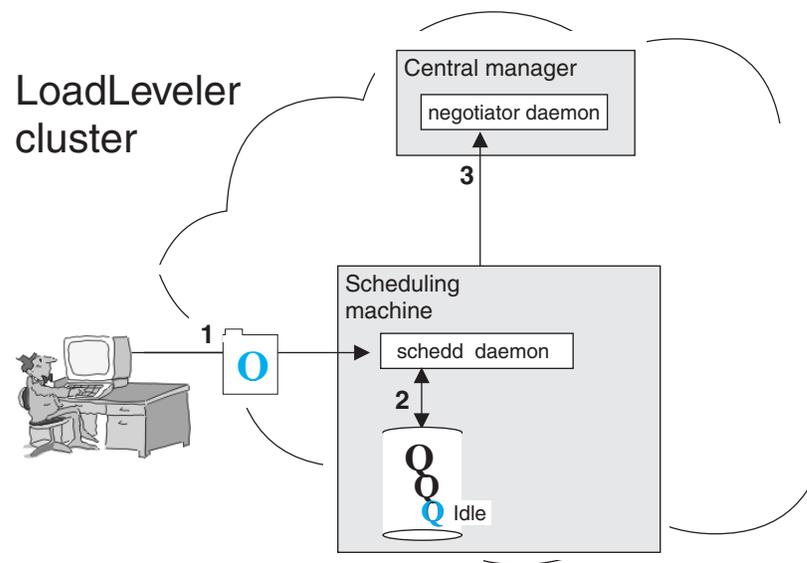


Figure 5. Job is submitted to LoadLeveler

Figure 5 illustrates that the Schedd daemon runs on the scheduling machine. This machine can also have the startd daemon running on it. The negotiator daemon resides on the central manager machine. The arrows in Figure 5 illustrate the following:

- Arrow 1 indicates that a job has been submitted to the scheduling machine.
 - Arrow 2 indicates that the Schedd daemon, on the scheduling machine, stores all of the relevant job information on local disk.
 - Arrow 3 indicates that the Schedd daemon sends job description information to the negotiator daemon. At this point, the submitted job is in the Idle state.
2. Permit to run:

LoadLeveler job cycle

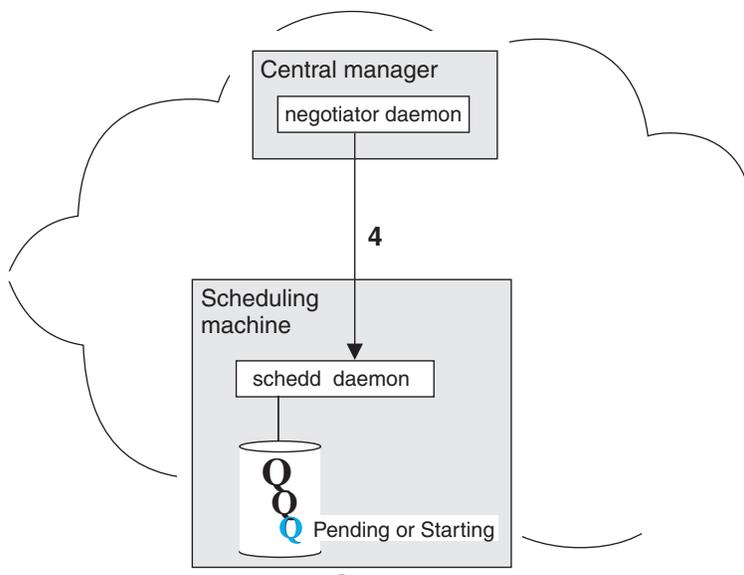


Figure 6. LoadLeveler authorizes the job

In Figure 6, arrow 4 indicates that the negotiator daemon authorizes the Schedd daemon to begin taking steps to run the job. This authorization is called a *permit to run*. Once this is done, the job is considered Pending or Starting.

3. Prepare to run:

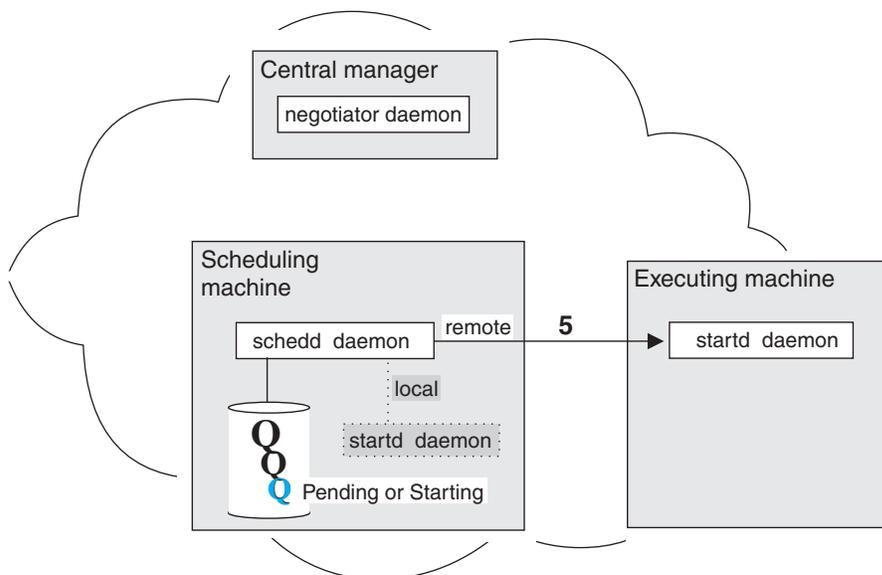


Figure 7. LoadLeveler prepares to run the job

In Figure 7, arrow 5 illustrates that the Schedd daemon contacts the startd daemon on the executing machine and requests that it start the job. The executing machine can either be a local machine (the machine to which the job was submitted) or another machine in the cluster. In this example, the local machine is **not** the executing machine.

4. Initiate job:

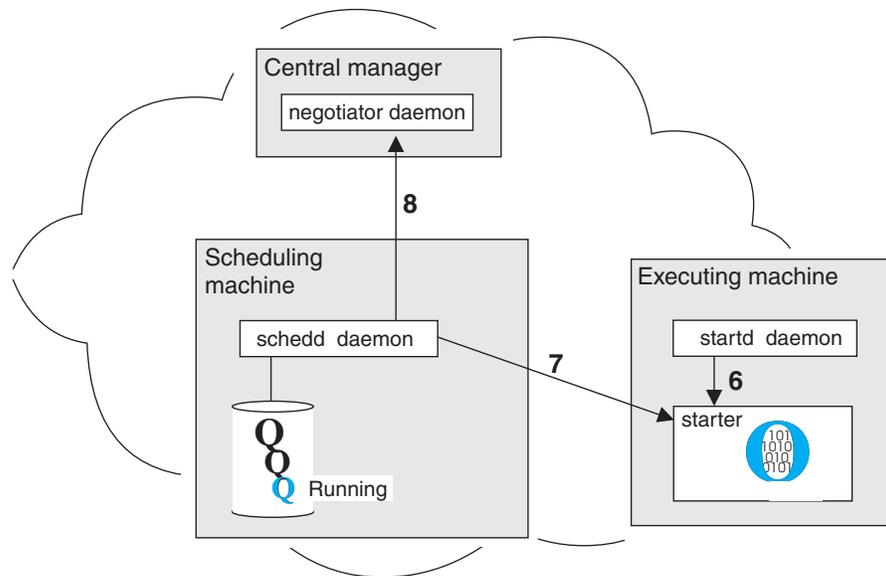


Figure 8. LoadLeveler starts the job

The arrows in Figure 8 illustrate the following:

- Arrow 6 indicates that the **startd** daemon on the executing machine spawns a **starter** process for the job.
- Arrow 7 indicates that the Schedd daemon sends the starter process the job information and the executable.
- Arrow 8 indicates that the Schedd daemon notifies the negotiator daemon that the job has been started and the negotiator daemon marks the job as Running.

The starter forks and executes the user's job, and the starter parent waits for the child to complete.

5. Complete job:

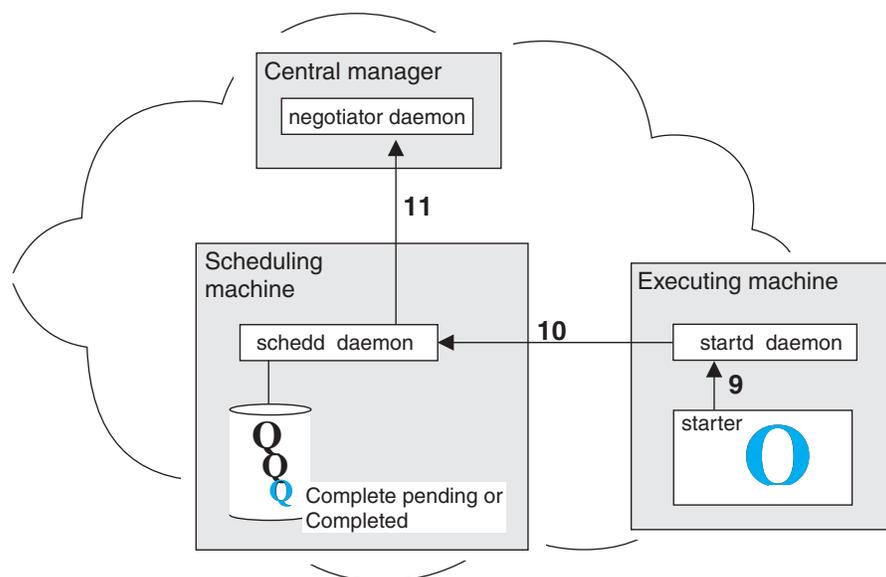


Figure 9. LoadLeveler completes the job

LoadLeveler job cycle

The arrows in Figure 9 on page 17 illustrate the following:

- Arrow 9 indicates that when the job completes, the starter process notifies the startd daemon.
- Arrow 10 indicates that the startd daemon notifies the Schedd daemon.
- Arrow 11 indicates that the Schedd daemon examines the information it has received, and forwards it to the negotiator daemon. At this point, the job is in Completed or Complete Pending state.

LoadLeveler job states

As LoadLeveler processes a job, the job moves through various states, which are listed in Table 7. Job states that include “Pending,” such as Complete Pending and Vacate Pending, are intermediate, temporary states.

Some options on LoadLeveler interfaces are valid only for jobs in certain states. For example, the **llmodify** command has options that apply only to jobs that are in the Idle state, or in states that are similar to it. To determine which job states are similar to the Idle state, use the “Similar to...” column in Table 7, which indicates whether a particular job state is similar to the Idle, Running, or Terminating state. A dash (—) indicates that the state is not similar to an Idle, Running, or Terminating state.

Table 7. Job state descriptions and abbreviations

Job state	Similar to Idle or Running state?	Abbreviation in displays / output	Description
Canceled	Terminating	CA	The job was canceled either by a user or by an administrator.
Checkpointing	Running	CK	Indicates that a checkpoint has been initiated.
Completed	Terminating	C	The job has completed.
Complete Pending	Terminating	CP	The job is in the process of being completed.
Deferred	Idle	D	The job will not be assigned to a machine until a specified date. This date may have been specified by the user in the job command file, or may have been generated by the negotiator because a parallel job did not accumulate enough machines to run the job. Only the negotiator places a job in the Deferred state.
Idle	Idle	I	The job is being considered to run on a machine, though no machine has been selected.

Table 7. Job state descriptions and abbreviations (continued)

Job state	Similar to Idle or Running state?	Abbreviation in displays / output	Description
Not Queued	Idle	NQ	The job is not being considered to run on a machine. A job can enter this state because the associated Schedd is down, the user or group associated with the job is at its maximum maxqueued or maxidle value, or because the job has a dependency which cannot be determined. For more information on these keywords, see “Controlling the mix of idle and running jobs” on page 668. (Only the negotiator places a job in the NotQueued state.)
Not Run	—	NR	The job will never be run because a dependency associated with the job was found to be false.
Pending	Running	P	The job is in the process of starting on one or more machines. (The negotiator indicates this state until the Schedd acknowledges that it has received the request to start the job. Then the negotiator changes the state of the job to Starting. The Schedd indicates the Pending state until all startd machines have acknowledged receipt of the start request. The Schedd then changes the state of the job to Starting.)
Preempted	Running	E	The job is preempted. This state applies only when LoadLeveler uses the suspend method to preempt the job.
Preempt Pending	Running	EP	The job is in the process of being preempted. This state applies only when LoadLeveler uses the suspend method to preempt the job.
Rejected	Idle	X	The job is rejected.
Reject Pending	Idle	XP	The job did not start. Possible reasons why a job is rejected are: job requirements were not met on the target machine, or the user ID of the person running the job is not valid on the target machine. After a job leaves the Reject Pending state, it is moved into one of the following states: Idle, User Hold, or Removed.
Removed	Terminating	RM	The job was stopped by LoadLeveler.
Remove Pending	Terminating	RP	The job is in the process of being removed, but not all associated machines have acknowledged the removal of the job.
Resume Pending	Running	MP	The job is in the process of being resumed.

Job states

Table 7. Job state descriptions and abbreviations (continued)

Job state	Similar to Idle or Running state?	Abbreviation in displays / output	Description
Running	Running	R	The job is running: the job was dispatched and has started on the designated machine.
Starting	Running	ST	The job is starting: the job was dispatched, was received by the target machine, and LoadLeveler is setting up the environment in which to run the job. For a parallel job, LoadLeveler sets up the environment on all required nodes. See the description of the "Pending" state for more information on when the negotiator or the Schedd daemon moves a job into the Starting state.
System Hold	Idle	S	The job has been put in system hold.
Terminated	Terminating	TX	If the negotiator and Schedd daemons experience communication problems, they may be temporarily unable to exchange information concerning the status of jobs in the system. During this period of time, some of the jobs may actually complete and therefore be removed from the Schedd's list of active jobs. When communication resumes between the two daemons, the negotiator will move such jobs to the Terminated state, where they will remain for a set period of time (specified by the <code>NEGOTIATOR_REMOVE_COMPLETED</code> keyword in the configuration file). When this time has passed, the negotiator will remove the jobs from its active list.
User & System Hold	Idle	HS	The job has been put in both system hold and user hold.
User Hold	Idle	H	The job has been put in user hold.
Vacated	Idle	V	The job started but did not complete. The negotiator will reschedule the job (provided the job is allowed to be rescheduled). Possible reasons why a job moves to the Vacated state are: the machine where the job was running was flushed, the <code>VACATE</code> expression in the configuration file evaluated to True, or LoadLeveler detected a condition indicating the job needed to be vacated. For more information on the <code>VACATE</code> expression, see "Managing job status through control expressions" on page 63.
Vacate Pending	Idle	VP	The job is in the process of being vacated.

Consumable resources

Consumable resources are assets available on machines in your LoadLeveler cluster. They are called "resources" because they model the commodities or services available on machines (including CPUs, real memory, virtual memory, software licenses, disk space). They are considered "consumable" because job steps use specified amounts of these commodities when the step is running. Once the step finishes, the resource becomes available for another job step.

Consumable resources which model the characteristics of a specific machine (such as the number of CPUs or the number of specific software licenses available only on that machine) are called machine resources. Consumable resources which model resources that are available across the LoadLeveler cluster (such as floating software licenses) are called floating resources. For example, consider a configuration with 10 licenses for a given program (which can be used on any machine in the cluster). If these licenses are defined as floating resources, all 10 can be used on one machine, or they can be spread across as many as 10 different machines.

The LoadLeveler administrator can specify:

- Consumable resources to be considered by LoadLeveler's scheduling algorithms
- Quantity of resources available on specific machines
- Quantity of floating resources available on machines in the cluster
- Consumable resources to be considered in determining the priority of executing machines
- Default amount of resources consumed by a job step of a specified job class
- Whether CPU and real memory resources should be enforced using AIX workload manager (WLM)
- Whether all jobs submitted need to specify resources

Users submitting jobs can specify the resources consumed by each task of a job step.

LoadLeveler for AIX supports memory and I/O adapter affinity options for improving IBM System p5™ and System p4™ machine performance. Memory and adapter affinity are where performance gains occur when jobs are limited to a specific multiple chip module (MCM). LoadLeveler makes use of the Resource Set (RSet) APIs available in AIX 5L™ for this purpose, attaching a task's processes to run only on CPUs of a single MCM. On an affinity-enabled system, the administrator can configure LoadLeveler to use RSets for tasks, based on consumable CPUs or based on the RSets specified in the job command file. For more information on scheduling affinity, see "LoadLeveler scheduling affinity support" on page 136.

Notes:

1. When software licenses are used as a consumable resource, LoadLeveler does not attempt to obtain software licenses or to verify that software licenses have been obtained. However, by providing a user exit that can be invoked as a submit filter, the LoadLeveler administrator can provide code to first obtain the required license and then allow the job step to run. For more information on filtering job scripts, see "Filtering a job script" on page 70.
2. LoadLeveler scheduling algorithms use the availability of requested consumable resources to determine the machine or machines on which a job will run. Consumable resources (except for CPU and real memory) are used only for scheduling purposes and are not enforced. Instead, LoadLeveler's negotiator daemon keeps track of the consumable resources available by

Consumable resources

reducing them by the amount requested when a job step is scheduled, and increasing them when a consuming job step completes.

3. If a job is preempted, the job continues to use all consumable resources except for `ConsumableCpus` and `ConsumableMemory` (real memory) which are made available to other jobs.
4. When the network adapters on a machine support RDMA, the machine is automatically given a consumable resource called RDMA with an available quantity defined by the limit on the number of concurrent jobs that use RDMA. For machines with "Switch Network Interface for HPS" network adapters, this limit is 4.
5. When steps require RDMA, either because they request `bulkxfer` or because they request `rcxtblocks` on at least one network statement, the job is automatically given a resource requirement for 1 RDMA.

Consumable resources and AIX Workload Manager

If the administrator has indicated that resources should be enforced, LoadLeveler uses AIX Workload Manager (WLM) to give greater control over CPU and real memory resource allocation. WLM monitors system resources and regulates their allocation to processes running on AIX. These actions prevent jobs from interfering with each other when they have conflicting resource requirements. WLM achieves this control by creating different classes of service and allowing attributes to be specified for those classes.

LoadLeveler dynamically generates WLM classes with specific resource entitlements. A single WLM class is created for each job step and the process id of that job step is assigned to that class. This is done for each node that a job step is assigned to execute on. LoadLeveler then defines resource shares or limits for that class depending on the LoadLeveler enforcement policy defined. These resource shares or limits represent the job's requested resource usage in relation to the amount of resources available on the machine.

When the enforcement policy is shares, LoadLeveler assigns a share value to the class based on the resources requested for the job step (one unit of resource equals one share). When the job step process is executing, AIX WLM dynamically calculates a desired resource entitlement based on the WLM class share value of the job step and the total number of shares requested by all active WLM classes. It is important to note that AIX WLM will only enforce these target percentages when the resource is under contention.

When the enforcement policy is limits (soft or hard), LoadLeveler assigns a percentage value to the class based on the resources requested for the job step and the total machine resources. This resource percentage is enforced regardless of any other active WLM classes. A soft limit indicates the maximum amount of the resource that can be made available when there is contention for the resources. This maximum can be exceeded if no one else requires the resource. A hard limit indicates the maximum amount of the resource that can be made available even if there is no contention for the resources.

Note: A WLM class is active for the duration of a job step's execution and is deleted when the job step completes. There is a limit of 27 active WLM classes per machine. Therefore, when resources are being enforced, only 27 job steps can be executing on one machine.

For more information on integrating LoadLeveler with AIX Workload Manager, see "Steps for integrating LoadLeveler with AIX Workload Manager" on page 127.

Overview of reservations

Under the BACKFILL scheduler only, LoadLeveler allows authorized users to make reservations, which specify a time period during which specific node resources are reserved for exclusive use by particular users or groups. This capability is known in the computing industry as advance reservation. Normally, jobs wait to be dispatched until the resources they require become available. Through the use of reservations, wait time can be reduced because the jobs have exclusive use of the node resources (CPUs, memory, disk drives, communication adapters, and so on) as soon as the reservation period begins.

Note: Advance reservation supports Blue Gene resources including the Blue Gene compute nodes. For more information, see “Blue Gene advance reservation support” on page 147.

In addition to reducing wait time, reservations also are useful for:

- Running a workload that needs to start or finish at a particular time. The job steps must be associated with, or bound to, the reservation before LoadLeveler can run them during the reservation period.
- Setting aside a set of nodes for maintenance purposes. In this case, job steps are not bound to the reservation.

Only bound job steps may run on the reserved nodes, which means that a bound job step competes for reserved resources only with other job steps that are bound to the same reservation.

The following sequence of events describes, in general terms, how you can set up and use reservations in the LoadLeveler environment. It also describes how LoadLeveler manages activities related to the use of reservations.

1. Configuring LoadLeveler to support reservations

An administrator uses specific keywords in the configuration and administration files to define general reservation policies. These keywords include:

- **max_reservations**, which defines both:
 - The users or groups that will be allowed to create reservations. To be authorized to create reservations, LoadLeveler administrators also must have the **max_reservations** keyword set in their own user or group stanzas.
 - How many reservations users may own.
- **max_reservation_duration**, which defines the maximum duration for reservations.
- **reservation_permitted**, which defines the nodes that may be used for reservations.

Administrators also may configure LoadLeveler to collect accounting data about reservations when the reservations complete or are canceled.

2. Creating reservations

After LoadLeveler is configured for reservations, an administrator or authorized user may create specific reservations, defining reservation attributes that include:

- The start time and the duration of the reservation. The start and end times for a reservation are based on the time-of-day (TOD) clock on the central manager machine.

Overview of reservations

- The nodes to be reserved. Until the reservation period actually begins, the selected nodes are available to run any jobs; when the reservation starts, only jobs bound to the reservation may run on the reserved nodes.
- The users or groups that may use the reservation.

LoadLeveler assigns a unique ID to the reservation, and returns that ID to the owner.

After the reservation is successfully created:

- Reservation owners may:
 - Modify, query, and cancel their reservations.
 - Allow other LoadLeveler users or groups to submit jobs to run during a reservation period.
 - Submit jobs to run during a reservation period.
- Users or groups that are allowed to use the reservation also may query reservations, and submit jobs to run during a reservation period. To run jobs during a reservation period, users must bind job steps to the reservation. You may bind both batch and interactive POE job steps to a reservation.

3. Preparing for the start of a reservation

During the preparation time for a reservation, LoadLeveler:

- Preempts any jobs that are still running on the reserved nodes.
- Checks the condition of reserved nodes, and notifies the reservation owner and LoadLeveler administrators by e-mail of any situations that might require the reservation owner or an administrator to take corrective action. Such conditions include:
 - Reserved nodes that are down, suspended, no longer in the LoadLeveler cluster, or otherwise unavailable for use.
 - Non-preemptable job steps that cannot finish running before the reservation start time.

During this time, reservation owners may modify, cancel, and add users or groups to their reservations. Owners and users or groups that are allowed to use the reservation may query the reservation or bind job steps to it.

4. Starting the reservation

When the reservation period begins, LoadLeveler dispatches job steps that are bound to the reservation.

After the reservation period begins, reservation owners may modify, cancel, and add users or groups to their reservations. Owners and users or groups that are allowed to use the reservation may query the reservation or bind job steps to it.

During the reservation period, LoadLeveler ignores system preemption rules for bound job steps; however, LoadLeveler administrators may use the **llpreempt** command to manually preempt bound job steps.

When the reservation ends or is canceled:

- LoadLeveler unbinds all job steps from the reservation. At this point, the unbound job steps compete with all other LoadLeveler jobs for available resources.
- If accounting data is being collected for the reservation, LoadLeveler also updates the reservation history file.

For more detailed information and instructions for setting up and using reservations, see:

- “Configuring LoadLeveler to support reservations” on page 121.
- “Working with reservations” on page 197.

Fair share scheduling overview

| Fair share scheduling in LoadLeveler provides a way to divide resources in a
| LoadLeveler cluster among users or groups of users. Historic resource usage data
| that is collected at the time the job ends can be used to influence job priorities to
| achieve the resource usage proportions allocated to users or groups of users in the
| LoadLeveler configuration files. The resource usage data will decay over time so
| that the relatively recent historic resource usage will have the most influence on
| job priorities. The CPU resources in the cluster and the Blue Gene resources are
| currently supported by fair share scheduling.

For information about configuring fair share scheduling in LoadLeveler, see “Using fair share scheduling” on page 148.

Fair share scheduling

Chapter 2. Getting a quick start using the default configuration

If you are very familiar with UNIX and Linux system administration and job scheduling, follow the steps listed to get LoadLeveler up and running on your network quickly in a default configuration. This default configuration will merely enable you to submit serial jobs; for a more complex setup, see Chapter 4, “Configuring the LoadLeveler environment,” on page 39.

What you need to know before you begin

LoadLeveler sets up the following default values for the configuration information:

- **loadl** is the recommended LoadLeveler user ID and the LoadLeveler group ID. LoadLeveler daemons run under this user ID to perform file I/O, and many LoadLeveler files are owned by this user ID.
- The home directory of **loadl** is the configuration directory.
- **LoadL_config** is the name of the configuration file.

For information about configuration file keyword syntax and other details, see Chapter 12, “Configuration file reference,” on page 243.

Using the default configuration files

Perform the following steps to use the default configuration files:

Note: You can find samples of the **LoadL_admin** and **LoadL_config** files in the **release** directory (in the **samples** subdirectory).

1. Ensure that the installation procedure has completed successfully and that the configuration file, **LoadL_config**, exists in LoadLeveler’s home directory or in the directory specified by the **LoadLConfig** keyword.
2. Identify yourself as the LoadLeveler administrator in the **LoadL_config** file using the **LOADL_ADMIN** keyword. The syntax of this keyword is:

LOADL_ADMIN = list_of_user_names (required)

Where *list_of_user_names* is a blank-delimited list of those individuals who will have administrative authority.

Refer to “Defining LoadLeveler administrators” on page 41 for more information.

3. Define a machine to act as the LoadLeveler central manager by coding one machine stanza as follows in the administration file, which is called **LoadL_admin**. (Replace *machine_name* with the actual name of the machine.)

```
machine_name: type = machine
```

```
central_manager = true
```

Do not specify more than one machine as the central manager. Also, if during installation, you ran **llinit** with the **-cm** flag, the central manager is already defined in the **LoadL_admin** file because the **llinit** command takes parameters that you entered and updates the administration and configuration files. See “Defining machines” on page 78 for more information.

LoadLeveler for Linux quick start

If you would like to quickly install and configure LoadLeveler for Linux and submit a serial job on a single node, follow the steps in the following procedures.

Note: This setup is for a single node only and the node used for this example is: `c197blade1b05.ppd.pok.ibm.com`.

Quick installation

Note: This installation is for RHEL 4 System x servers, but the same method is applicable to all other systems. You must install the corresponding license RPM for the system you are installing on. This installation assumes that the LoadLeveler RPMs are located at: `/mnt/cdrom/`.

1. Log on to node `c197blade1b05.ppd.pok.ibm.com` as root, which is the node you are installing on.
2. Install the license RPM by entering the following command:

```
rpm -ivh /mnt/cdrom/LoadL-full-license-RH4-X86-3.4.0.0-0.i386.rpm
```
3. Change to the LoadLeveler installation path by entering the following the command:

```
cd /opt/ibm11/LoadL/sbin
```
4. Run the LoadLeveler installation script by entering:

```
./install_ll -y -d /mnt/cdrom
```
5. Add a UNIX group for LoadLeveler users (make sure the group ID is correct) by entering the following command:

```
groupadd -g 1000 loadl
```
6. Add a UNIX user for LoadLeveler (make sure the user ID is correct) by entering the following command:

```
useradd -c "LoadLeveler User" -d /home/loadl -s /bin/bash -u 1001 -g 1000 -m loadl
```

Quick configuration

1. Change the log in to the newly created LoadLeveler user by entering the following command:

```
su - loadl
```
2. Add the LoadLeveler **bin** directory to the search path:

```
export PATH=$PATH:/opt/ibm11/LoadL/full/bin
```
3. Run the LoadLeveler initialization script:

```
/opt/ibm11/LoadL/full/bin/llinit -local /tmp/loadl -release /opt/ibm11/LoadL/full -cm c197blade1b05.ppd.pok.ibm.com
```

Quick verification

1. Start LoadLeveler by entering the following command:

```
llctl start
```

You should receive a response similar to the following:

```
llctl: Attempting to start LoadLeveler on host c197blade1b05.ppd.pok.ibm.com
LoadL_master 3.4.0.0 rven0632a 2006/08/09 RHEL 4.0 140
CentralManager = c197blade1b05.ppd.pok.ibm.com
[loadl@c197blade1b05 bin]$
```
2. Check LoadLeveler status by entering the following command:

```
llstatus
```

You should receive a response similar to the following:

```

Name                               Schedd InQ  Act Startd Run LdAvg Idle Arch  OpSys
c197blade1b05.ppd.pok.ibm Avail    0  0 Idle    0 0.00  1 i386  Linux2
i386/Linux2                        1 machines    0 jobs    0 running task
Total Machines                      1 machines    0 jobs    0 running task

```

The Central Manager is defined on c197blade1b05.ppd.pok.ibm.com

The BACKFILL scheduler is in use

All machines on the machine_list are present.
[loadl@c197blade1b05 bin]\$

- Submit a sample job, by entering the following command:

```
llsubmit /opt/ibm11/LoadL/full/samples/job1.cmd
```

You should receive a response similar to the following:

```
llsubmit: The job "c197blade1b05.ppd.pok.ibm.com.1" with 2 job steps /
has been submitted.
[loadl@c197blade1b05 samples]$
```

- Display the LoadLeveler job queue, by entering the following command:

```
llq
```

You should receive a response similar to the following:

```

Id                               Owner      Submitted  ST PRI Class      Running On
-----
c197blade1b05.1.0                loadl      8/15 17:25 R  50 No_Class      c197blade1b05
c197blade1b05.1.1                loadl      8/15 17:25 I  50 No_Class
2 job step(s) in queue, 1 waiting, 0 pending, 1 running, 0 held, 0 preempted
[loadl@c197blade1b05 samples]$

```

- Check output files into the home directory (**/home/loadl**) by entering the following command:

```
ls -ltr job*
```

You should receive a response similar to the following:

```

-rw-rw-r-- 1 loadl loadl 1940 Aug 15 17:26 job1.c197blade1b05.1.0.out
-rw-rw-rw- 1 loadl loadl 1940 Aug 15 17:27 job1.c197blade1b05.1.1.out
[loadl@c197blade1b05 ~]$

```

Post-installation considerations

This information explains how to start (or restart) and stop LoadLeveler. It also tells you where files are located after you install LoadLeveler, and it points you to troubleshooting information.

Starting LoadLeveler

You can start LoadLeveler using any LoadLeveler administrator user ID as defined in the configuration file. To start all of the machines that are defined in machine stanzas in the administration file, enter:

```
llctl -g start
```

The central manager machine is the first started, followed by other machines in the order listed in the administration file. See “llctl - Control LoadLeveler daemons” on page 409 for more information.

By default, **llctl** uses **rsh** to start LoadLeveler on the target machine. Other mechanisms, such as **ssh** can be used by setting the **LL_RSH_COMMAND** configuration keyword in **LoadL_config**. However you choose to start LoadLeveler on remote hosts, you must have the authority to run commands remotely on that host.

You can verify that the machine has been properly configured by running the sample jobs in the appropriate **samples** directory (job1.cmd, job2.cmd, and job3.cmd). You must read the job2.cmd and job3.cmd files before submitting them because job2 must be edited and a C program must be compiled to use job3. It is a good idea to copy the sample jobs to another directory before modifying them; you must have read/write permission to the directory in which they are located. You can use the **llsubmit** command to submit the sample jobs from several different machines and verify that they complete (see “llsubmit - Submit a job” on page 494).

If you are running AFS and some jobs do not complete, you might need to use the AFS **fs** command (**fs listacl**) to ensure that you have write permission to the **spool**, **execute**, and **log** directories.

If you are running with cluster security services enabled and some jobs do not complete, ensure that you have write permission to the **spool**, **execute**, and **log** directories. Also ensure that the user ID is authorized to run jobs on the submitting machine (the identity of the user must exist in the **.rhosts** file of the user on the machine on which the job is being run).

Note: LoadLeveler for Linux does not support cluster security services.

If you are running submit-only LoadLeveler, once the LoadLeveler pool is up and running, you can use the **llsubmit**, **llq**, and **llcancel** commands from the submit-only machines. For more information about these commands, see

- “llsubmit - Submit a job” on page 494
- “llq - Query job status” on page 449
- “llcancel - Cancel a submitted job” on page 392

You can also invoke the LoadLeveler graphical user interface **xloadl_so** from the submit-only machines (see Chapter 15, “Graphical user interface (GUI) reference,” on page 377).

Location of directories following installation

After installation, the product directories shown in Table 8 reside on disk. The installation process creates only those directories required to service the LoadLeveler options specified during the install. For AIX, *release_directory* indicates **/usr/lpp/LoadL/full** and for Linux, it indicates **/opt/ibmll/LoadL/full**.

Table 8. Location and description of product directories following installation

Directory	Description
<i>release_directory</i> /bin	Part of the release directory containing daemons, commands, and other binaries
<i>release_directory</i> /lib	Part of the release directory containing product libraries and resource files
<i>release_directory</i> /man	Part of the release directory containing man pages
<i>release_directory</i> /samples	Part of the release directory containing sample administration and configuration files and sample jobs
<i>release_directory</i> /include	Part of the release directory containing header files for the application programming interfaces.

Table 8. Location and description of product directories following installation (continued)

Directory	Description
Local directory	spool , execute , and log directories for each machine in the cluster
Home directory	Administration and configuration files, and symbolic links to the release directory
/usr/lpp/LoadL/codebase	Configuration tasks for AIX

Table 9 shows the location of directories for submit-only LoadLeveler:

Table 9. Location and description of directories for submit-only LoadLeveler

Directory	Description
<i>release_directory</i> /so/bin	Part of the release directory containing commands.
<i>release_directory</i> /so/man	Part of the release directory containing man pages
<i>release_directory</i> /so/samples	Part of the release directory containing sample administration and configuration files.
<i>release_directory</i> /so/lib	Contains libraries and graphical user interface resource files
Home directory	Contains administration and configuration files

If you have a mixed LoadLeveler cluster of AIX and Linux machines, you might want to make the following symbolic links:

- On AIX, as **root**, enter:


```
mkdir -p /opt/ibmll
ln -s /usr/lpp/LoadL /opt/ibmll/LoadL
```
- On Linux, as **root**, enter:


```
mkdir -p /usr/lpp
ln -s /opt/ibmll/LoadL /usr/lpp/LoadL
```

With the addition of these symbolic links, a user application can use either **/usr/lpp/LoadL** or **/opt/ibmll/LoadL** to refer to the location of LoadLeveler files regardless of whether the application is running on AIX or Linux.

If LoadLeveler will not start following installation, see “Why won’t LoadLeveler start?” on page 651 for troubleshooting information.

Chapter 3. What operating systems are supported by LoadLeveler?

LoadLeveler supports three operating systems:

- **AIX 5L**

IBM's AIX 5L is an open UNIX operating environment that conforms to The Open Group UNIX 98 Base Brand industry standard. It provides high levels of integration, flexibility, and reliability and operates on IBM System p™, and IBM eServer™ Cluster 1600 servers and workstations.

AIX 5L supports the concurrent operation of 32- and 64-bit applications, with key internet technologies such as Java™ and XML parser for Java included as part of the base operating system.

A strong affinity between AIX and Linux permits popular applications developed on Linux to run on AIX 5L with a simple recompilation.

- **Linux**

LoadLeveler supports the following distributions of Linux:

- Red Hat® Enterprise Linux (RHEL) 3 and RHEL 4 on IA-32 servers
- RHEL 3 and RHEL 4 on AMD Opteron or Intel EM64T processors
- RHEL 4 on IBM POWER servers
- SUSE Linux Enterprise Server (SLES) 9 and SLES 10 on IA-32 servers
- SLES 9 and SLES 10 on IBM POWER servers
- SLES 9 and SLES 10 on AMD Opteron or Intel EM64T processors

Note: IBM Tivoli Workload Scheduler LoadLeveler is supported when running Linux on non-IBM Intel-based and AMD hardware servers.

Supported hardware includes:

- Servers with Intel 32-bit and Intel Extended Memory 64 Technology (EM64T)
- Servers with Advanced Micro Devices (AMD) 64-bit technology

- **IBM System Blue Gene Solution**

While no LoadLeveler processes actually run on the Blue Gene machine, LoadLeveler can interact with the Blue Gene machine and supports the scheduling of jobs to the machine.

Note: For models of the Blue Gene system such as Blue Gene/S, which can only run a single job at a time, LoadLeveler does not have to be configured to schedule resources for Blue Gene jobs. For such systems, serial jobs can be used to submit work to the front end node for the Blue Gene system.

AIX and Linux compatibility

LoadLeveler 3.4 for Linux is compatible with LoadLeveler 3.4 for AIX. Its command line interfaces, graphical user interfaces, and application programming interfaces (APIs) are the same as they have been for AIX. The formats of the job command file, configuration file, and administration file also remain the same.

System administrators can set up and maintain a LoadLeveler cluster consisting of some machines running LoadLeveler for AIX and some machines running

Supported operating systems

LoadLeveler for Linux. This is called a mixed cluster. In this mixed cluster jobs can be submitted from either AIX or Linux machines. Jobs submitted to a Linux job queue can be dispatched to an AIX machine for execution, and jobs submitted to an AIX job queue can be dispatched to a Linux machine for execution.

Although the LoadLeveler products for AIX and Linux are compatible, they do have some differences in the level of support for specific features. For further details, see the following topics:

- “Restrictions for LoadLeveler for Linux.”
- “Features not supported in Linux.”
- “Restrictions for LoadLeveler AIX and Linux mixed clusters” on page 35.

Restrictions for LoadLeveler for Linux

LoadLeveler for Linux supports a subset of the features that are available in the LoadLeveler for AIX product. The following features are available, but are subject to restrictions:

- 32-bit applications using the LoadLeveler APIs

LoadLeveler for Linux supports only the 32-bit LoadLeveler API library (libllapi.so) on the following platforms:

- RHEL 3 and RHEL 4 on IBM IA-32 xSeries® servers
- SLES 9 and SLES 10 on IBM IA-32 xSeries servers

Applications linked to the LoadLeveler APIs on these platforms must be 32-bit applications.

- 64-bit applications using the LoadLeveler APIs

LoadLeveler for Linux supports only the 64-bit LoadLeveler API library (libllapi.so) on the following platforms:

- RHEL 3 and RHEL 4 on IBM xSeries servers with AMD Opteron or Intel EM64T processors
- RHEL 4 on POWER™ servers
- SLES 9 on IBM xSeries servers with AMD Opteron or Intel EM64T processors
- SLES 9 and SLES 10 on POWER servers

Applications linked to the LoadLeveler APIs on these platforms must be 64-bit applications.

- Support for AFS file systems

LoadLeveler for Linux support for authenticated access to AFS file systems is limited to RHEL 3 and RHEL 4 on xSeries servers and IBM xSeries servers with AMD Opteron or Intel EM64T processors. It is not available on systems running SLES 9 or SLES 10.

- Support for preempting jobs under the BACKFILL scheduler

In LoadLeveler for AIX, you can use several methods for preempting jobs under the BACKFILL scheduler; LoadLeveler for Linux supports all of these methods except for one: suspend. If you request preemption through the suspend method on LoadLeveler for Linux, preemption will not occur.

Features not supported in Linux

LoadLeveler 3.4 for Linux supports a subset of the features that are available in the LoadLeveler 3.4 for AIX product. The following features are not supported:

- MCM scheduling affinity
MCM scheduling affinity is not supported by LoadLeveler for Linux.
- RDMA consumable resource
RDMA consumable resources are not supported by LoadLeveler for Linux.

- User context RDMA blocks
User context RDMA blocks are not supported by LoadLeveler for Linux.
- Checkpoint/restart
LoadLeveler for AIX uses a number of features that are specific to the AIX kernel to provide support for checkpoint/restart of user applications running under LoadLeveler. Checkpoint/restart is not available in this release of LoadLeveler for Linux.
- Process tracking
On AIX, the process tracking feature is implemented as a kernel extension. Using this feature, LoadLeveler for AIX is able to ensure that when a job managed by LoadLeveler has terminated no processes or threads associated with this job are left behind and continue to consume or hold resources. Process tracking is not supported in this release of LoadLeveler for Linux.
- AIX Workload management (WLM)
WLM can strictly control use of system resources. LoadLeveler for AIX uses WLM to enforce the use of a number of consumable resources defined by LoadLeveler (such as **ConsumableCpus** and **ConsumableMemory**). This enforcement of consumable resources usage through WLM is not available in this release of LoadLeveler for Linux.
- CtSec security
LoadLeveler for AIX can exploit CtSec (Cluster Security Services) security functions. These functions authenticate the identity of users and programs interacting with LoadLeveler. These features are not available in this release of LoadLeveler for Linux.
- **LoadL_GSmonitor** daemon
The **LoadL_GSmonitor** daemon in the LoadLeveler for AIX product uses the Group Services Application Programming Interface (GSAPI) to monitor machine availability and notify the LoadLeveler Central Manager when a machine is no longer reachable. This daemon is not available in the LoadLeveler for Linux product.
- Task guide tool
- Dynamic adapter setup
LoadLeveler for AIX can be configured to dynamically determine adapter characteristics, including those associated with the Switch Network Interface for the IBM eServer pSeries[®] High Performance Switch. This feature is not supported by LoadLeveler for Linux.
- System error log
Each LoadLeveler daemon has its own log file where information relevant to its operation is recorded. In addition to this feature which exists on all platforms, LoadLeveler for AIX also uses the errlog function to record critical LoadLeveler events into the AIX system log. Support for an equivalent Linux function is not available in this release.

Restrictions for LoadLeveler AIX and Linux mixed clusters

When operating a LoadLeveler cluster that contains AIX 5L and Linux machines, the following restrictions apply:

- Nodes running LoadLeveler for AIX 3.3 must be at 3.3.0.1 or later.
- All nodes in a mixed LoadLeveler cluster must run LoadLeveler version 3.2 or later.
- The Central Manager node must run a version of LoadLeveler equal to or higher than any LoadLeveler version being run on a node in the cluster.

Supported operating systems

- CtSec security features cannot be used.
- AIX jobs that use checkpointing must be sent to AIX nodes for execution. This can be done by either defining and specifying job checkpointing for job classes that exist only on AIX nodes or by coding appropriate requirements expressions. Checkpointing jobs that are sent to a Linux node will be rejected by the **LoadL_startd** daemon running on the Linux node.
- WLM and Process tracking are supported in a mixed cluster. However, enforcement of the use of consumable resources will occur through WLM on AIX nodes only. Similarly, the functions associated with Process Tracking are effective only on AIX nodes.
- For the BACKFILL scheduler, LoadLeveler for Linux does not support preempting jobs through the suspend method. In a mixed cluster, if you request preemption through the suspend method, the specified jobs running under AIX will be preempted, but those jobs running on Linux nodes will not be preempted.

Part 2. Configuring and managing the TWS LoadLeveler environment

After installing IBM Tivoli Workload Scheduler (TWS) LoadLeveler, you may customize it by modifying both the **configuration** file and the **administration** file. The configuration file contains many parameters that you can set or modify that will control how TWS LoadLeveler operates. The administration file optionally lists and defines the machines in the TWS LoadLeveler cluster and the characteristics of classes, users, and groups.

To easily manage TWS LoadLeveler, you should have one global configuration file and only one administration file, both centrally located on a machine in the TWS LoadLeveler cluster. Every other machine in the cluster must be able to read the configuration and administration file that are located on the central machine.

You may have multiple local configuration files that specify information specific to individual machines.

TWS LoadLeveler does not prevent you from having multiple copies of administration files, but you need to be sure to update all the copies whenever you make a change to one. Having only one administration file prevents any confusion.

Table 10 identifies where you can find more information about using configuration and administration files to modify the TWS LoadLeveler environment.

Table 10. Roadmap of tasks for TWS LoadLeveler administrators

To learn about:	Read the following:
Controlling how TWS LoadLeveler operates by customizing the global or local configuration file	Chapter 4, "Configuring the LoadLeveler environment," on page 39
Controlling TWS LoadLeveler resources by customizing an administration file	Chapter 5, "Defining LoadLeveler resources to administer," on page 77
Additional ways to modify TWS LoadLeveler that require customization of both the configuration and administration files	Chapter 6, "Performing additional administrator tasks," on page 95
Ways to control or monitor TWS LoadLeveler operations by using the TWS LoadLeveler commands, GUI, and APIs	<ul style="list-style-type: none">• Chapter 16, "Commands," on page 385• Chapter 7, "Using LoadLeveler's GUI to perform administrator tasks," on page 155• Chapter 17, "Application programming interfaces (APIs)," on page 503

Chapter 4. Configuring the LoadLeveler environment

One of your main tasks as system administrator is to configure LoadLeveler. To configure LoadLeveler, you need to know what the configuration information is and where it is located. Configuration information includes the following:

- The LoadLeveler user ID and group ID
- The configuration directory
- The global configuration file

Configuring LoadLeveler involves modifying the configuration files that specify the terms under which LoadLeveler can use machines. There are two types of configuration files:

- *Global Configuration File:* This file by default is called the **LoadL_config** file and it contains configuration information common to all nodes in the LoadLeveler cluster.
- *Local Configuration File:* This file is generally called **LoadL_config.local** (although it is possible for you to rename it). This file contains specific configuration information for an individual node. The **LoadL_config.local** file is in the same format as **LoadL_config** and the information in this file overrides any information specified in **LoadL_config**. It is an optional file that you use to modify information on a local machine. Its full pathname is specified in the **LoadL_config** file by using the **LOCAL_CONFIG** keyword. See “Specifying file and directory locations” on page 45 for more information.

You can run your installation with default values set by LoadLeveler, or you can change any or all of them. Table 11 lists topics that discuss how you may configure the LoadLeveler environment by modifying the configuration file.

Table 11. Roadmap of administrator tasks related to using or modifying the LoadLeveler configuration file

To learn about:	Read the following:
Using the default configuration files shipped with LoadLeveler	Chapter 2, “Getting a quick start using the default configuration,” on page 27
Modifying the global and local configuration files	“Modifying a configuration file” on page 40
Defining major elements of the LoadLeveler configuration	<ul style="list-style-type: none">• “Defining LoadLeveler administrators” on page 41• “Defining a LoadLeveler cluster” on page 41• “Defining LoadLeveler machine characteristics” on page 51• “Defining security mechanisms” on page 52• “Defining usage policies for consumable resources” on page 56• “Steps for configuring a LoadLeveler multicluster” on page 141

Customizing the configuration file

Table 11. Roadmap of administrator tasks related to using or modifying the LoadLeveler configuration file (continued)

To learn about:	Read the following:
Enabling optional LoadLeveler functions	<ul style="list-style-type: none">• “Enabling support for bulk data transfer and rCxt blocks” on page 57• “Gathering job accounting data” on page 57• “Managing job status through control expressions” on page 63• “Tracking job processes” on page 64• “Querying multiple LoadLeveler clusters” on page 65
Modifying LoadLeveler operations through installation exits	“Providing additional job-processing controls through installation exits” on page 66

Modifying a configuration file

By taking a look at the configuration files that come with LoadLeveler, you will find that there are many parameters that you can set. In most cases, you will only have to modify a few of these parameters. In some cases, though, depending upon the LoadLeveler nodes, network connection, and hardware availability, you may need to modify additional parameters.

All LoadLeveler commands, daemons, and processes read the administration and configuration files at start up time. If you change the administration or configuration files after LoadLeveler has already started, any LoadLeveler command or process, such as the **LoadL_starter** process, will read the newer version of the files while the running daemons will continue to use the data from the older version. To ensure that all LoadLeveler commands, daemons, and processes use the same configuration data, run the reconfiguration command on all machines in the cluster each time the administration or configuration files are changed.

To override the defaults, you must update the following keywords in the **/etc/LoadL.cfg** file:

LoadLUserid

Specifies the LoadLeveler user ID.

LoadLGroupid

Specifies the LoadLeveler group ID.

LoadLConfig

Specifies the full path name of the configuration file.

Note that if you change the LoadLeveler user ID to something other than **loadl**, you will have to make sure your configuration files are owned by this ID.

If Cluster Security (CtSec) services is enabled, make sure you update the **unix.map** file if the **LoadLUserid** is specified as something other than **loadl**. Refer to “Steps for enabling CtSec services” on page 54 for more details.

You can also override the **/etc/LoadL.cfg** file. For an example of when you might want to do this, see “Querying multiple LoadLeveler clusters” on page 65.

Before you modify a configuration file, you need to:

- Ensure that the installation procedure has completed successfully and that the configuration file, **LoadL_config**, exists in LoadLeveler's home directory or in the directory specified in **/etc/LoadL.cfg**. For additional details about installation, see *TWS LoadLeveler: Installation Guide*.
- Know how to correctly specify keywords in the configuration file. For information about configuration file keyword syntax and other details, see Chapter 12, "Configuration file reference," on page 243.
- Identify yourself as the LoadLeveler administrator using the **LOADL_ADMIN** keyword.

After you finish modifying the configuration file, notify LoadLeveler daemons by issuing the **llctl** command with either the **reconfig** or **recycle** keyword. Otherwise, LoadLeveler will not process the modifications you made to the configuration file.

Defining LoadLeveler administrators

Specify the **LOADL_ADMIN** keyword with a list of user names of those individuals who will have administrative authority. These users are able to invoke the administrator-only commands such as **llctl**, **llfavorjob**, and **llfavoruser**. These administrators can also invoke the administrator-only GUI functions. For more information, see Chapter 7, "Using LoadLeveler's GUI to perform administrator tasks," on page 155.

LoadLeveler administrators on this list also receive mail describing problems that are encountered by the master daemon. When CtSec is enabled, the **LOADL_ADMIN** list is used only as a mailing list. For more information, see "Defining security mechanisms" on page 52.

An administrator on a machine is granted administrative privileges on that machine. It does not grant him administrative privileges on other machines. To be an administrator on all machines in the LoadLeveler cluster, either specify your user ID in the global configuration file with no entries in the local configuration file, or specify your user ID in every local configuration file that exists in the LoadLeveler cluster.

For information about configuration file keyword syntax and other details, see Chapter 12, "Configuration file reference," on page 243.

Defining a LoadLeveler cluster

Table 12 lists the topics that discuss how you can define the characteristics of the LoadLeveler cluster.

Table 12. Roadmap for defining LoadLeveler cluster characteristics

To learn about:	Read the following:
Defining characteristics of specific LoadLeveler daemons	<ul style="list-style-type: none"> • "Choosing a scheduler" on page 42 • "Setting negotiator characteristics and policies" on page 43 • "Specifying alternate central managers" on page 44

Customizing the configuration file

Table 12. Roadmap for defining LoadLeveler cluster characteristics (continued)

To learn about:	Read the following:
Defining other cluster characteristics	<ul style="list-style-type: none">• “Defining network characteristics” on page 45• “Specifying file and directory locations” on page 45• “Configuring recording activity and log files” on page 46• “Setting up file system monitoring” on page 50
Correctly specifying configuration file keywords	Chapter 12, “Configuration file reference,” on page 243
Working with daemons and machines in a LoadLeveler cluster	<ul style="list-style-type: none">• “llctl - Control LoadLeveler daemons” on page 409• “llinit - Initialize machines in the LoadLeveler cluster” on page 429

Choosing a scheduler

This topic discusses the types of schedulers available, which you may specify using the configuration file keyword **SCHEDULER_TYPE**. For information about configuration file keyword syntax and other details, see Chapter 12, “Configuration file reference,” on page 243.

LL_DEFAULT This scheduler runs both serial and parallel jobs, but is primarily meant for serial jobs. It efficiently uses CPU time by scheduling jobs on what otherwise would be idle nodes (and workstations). It does not require that users set a wall clock limit. Also, this scheduler starts, suspends, and resumes jobs based on workload. The default scheduler uses a reservation method to schedule parallel jobs. A possible drawback to the reservation method occurs when LoadLeveler tries to schedule a job requiring a large number of nodes. As LoadLeveler reserves nodes for the job, the reserved nodes will be idle for a period of time. Also, if the job cannot accumulate all the nodes it needs to run, the job may not get dispatched.

See “Scheduler support for parallel jobs” on page 178 for information on which keywords associated with parallel jobs are supported by the default scheduler.

BACKFILL This scheduler runs both serial and parallel jobs, but is primarily meant for parallel jobs. The objective of BACKFILL scheduling is to maximize the use of resources to achieve the highest system efficiency, while preventing potentially excessive delays in starting jobs with large resource requirements. These large jobs can run because the BACKFILL scheduler does not allow jobs with smaller resource requirements to continuously use up resource before the larger jobs can accumulate enough resource to run.

The BACKFILL scheduler supports:

- The scheduling of multiple tasks per node.
- The scheduling of multiple user space tasks per adapter.
- The preemption of jobs.
- The use of reservations.

These functions are not supported by the default LoadLeveler scheduler.

For more information about the BACKFILL scheduler, see “Using the BACKFILL scheduler” on page 101.

API

This keyword option allows you to enable an external scheduler, such as the Extensible Argonne Scheduling sYstem (EASY). The API option is intended for installations that want to create a scheduling algorithm for parallel jobs based on site-specific requirements.

For more information about external schedulers, see “Using an external scheduler” on page 105.

Setting negotiator characteristics and policies

You may set the following negotiator characteristics and policies. For information about configuration file keyword syntax and other details, see Chapter 12, “Configuration file reference,” on page 243.

- Prioritize the queue maintained by the negotiator

Each job step submitted to LoadLeveler is assigned a system priority number, based on the evaluation of the **SYSPRIO** keyword expression in the configuration file of the central manager. The LoadLeveler system priority number is assigned when the central manager adds the new job step to the queue of job steps eligible for dispatch. Once assigned, the system priority number for a job step is not changed, except under the following circumstances:

- An administrator or user issues the **llprio** command to change the system priority of the job step.
- The value set for the **NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL** keyword is not zero.
- An administrator uses the **llmodify** command with the **-s** option to alter the system priority of a job step.
- A program with administrator credentials uses the **ll_modify** subroutine to alter the system priority of a job step.

Job steps assigned higher **SYSPRIO** numbers are considered for dispatch before job steps with lower numbers.

For related information, see the following topics:

- “Controlling the central manager scheduling cycle” on page 67.
- “Setting and changing the priority of a job” on page 212.
- “llmodify - Change attributes of a submitted job step” on page 435.
- “ll_modify subroutine” on page 628.

- Prioritize the order of executing machines maintained by the negotiator

Each executing machine is assigned a machine priority number, based on the evaluation of the **MACHPRIO** keyword expression in the configuration file of the central manager. The LoadLeveler machine priority number is updated every time the central manager updates its machine data. Machines assigned higher **MACHPRIO** numbers are considered to run jobs before machines with lower numbers. For example, a machine with a **MACHPRIO** of 10 is considered to run a job before a machine with a **MACHPRIO** of 5. Similarly, a machine with a **MACHPRIO** of -2 would be considered to run a job before a machine with a **MACHPRIO** of -3.

Note that the **MACHPRIO** keyword is valid only on the machine where the central manager is running. Using this keyword in a local configuration file has no effect.

When you use a **MACHPRIO** expression that is based on load average, the machine may be temporarily ordered later in the list immediately after a job is scheduled to that machine. This temporary drop in priority happens because the

Customizing the configuration file

negotiator adds a compensating factor to the startd machine's load average every time the negotiator assigns a job. For more information, see "the NEGOTIATOR_LOADAVG_INCREMENT keyword" on page 271.

- Specify additional negotiator policies
This topic lists keywords that were not mentioned in the previous configuration steps. Unless your installation has special requirements for any of these keywords, you can use them with their default settings.
 - NEGOTIATOR_INTERVAL
 - NEGOTIATOR_CYCLE_DELAY
 - NEGOTIATOR_CYCLE_TIME_LIMIT
 - NEGOTIATOR_LOADAVG_INCREMENT
 - NEGOTIATOR_PARALLEL_DEFER
 - NEGOTIATOR_PARALLEL_HOLD
 - NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL
 - NEGOTIATOR_REJECT_DEFER
 - NEGOTIATOR_REMOVE_COMPLETED
 - NEGOTIATOR_RESCAN_QUEUE

Specifying alternate central managers

In one of your machine stanzas specified in the administration file, you specified that the machine would serve as the central manager. It is possible for some problem to cause this central manager to become unusable such as network communication or software or hardware failures. In such cases, the other machines in the LoadLeveler cluster believe that the central manager machine is no longer operating. To remedy this situation, you can assign one or more alternate central managers in the machine stanza to take control.

The following machine stanza example defines the machine `deep_blue` as an alternate central manager:

```
#
deep_blue: type=machine
central_manager = alt
```

If the primary central manager fails, the alternate central manager then becomes the central manager. The alternate central manager is chosen based upon the order in which its respective machine stanza appears in the administration file.

When an alternate becomes the central manager, jobs will not be lost, but it may take a few minutes for all of the machines in the cluster to check in with the new central manager. As a result, job status queries may be incorrect for a short time.

When you define alternate central managers, you should set the following keywords in the configuration file:

- **CENTRAL_MANAGER_HEARTBEAT_INTERVAL**
- **CENTRAL_MANAGER_TIMEOUT**

In the following example, the alternate central manager will wait for 30 intervals, where each interval is 45 seconds:

```
# Set a 45 second interval
CENTRAL_MANAGER_HEARTBEAT_INTERVAL = 45
# Set the number of intervals to wait
CENTRAL_MANAGER_TIMEOUT = 30
```

For more information on central manager backup, refer to “What happens if the central manager isn’t operating?” on page 658. For information about configuration file keyword syntax and other details, see Chapter 12, “Configuration file reference,” on page 243.

Defining network characteristics

A **port number** is an integer that specifies the port to use to connect to the specified daemon. You can define these port numbers in the configuration file or the `/etc/services` file or you can accept the defaults. LoadLeveler first looks in the configuration file for these port numbers. If LoadLeveler does not find the value in the configuration file, it looks in the `/etc/services` file. If the value is not found in this file, the default is used.

See Appendix C, “LoadLeveler port usage,” on page 685 for more information.

Specifying file and directory locations

The configuration file provided with LoadLeveler specifies default locations for all of the files and directories. You can modify their locations using the keywords shown in Table 13. Keep in mind that the LoadLeveler installation process installs files in these directories and these files may be periodically cleaned up. Therefore, you should not keep any files that do not belong to LoadLeveler in these directories.

Managing distributed software systems is a primary concern for all system administrators. Allowing users to share file systems to obtain a single, network-wide image, is one way to make managing LoadLeveler easier.

Table 13. Default locations for all of the files and directories

To specify the location of the:	Specify this keyword:
Administration file	ADMIN_FILE
Local configuration file	LOCAL_CONFIG
Local directory	<p>The following subdirectories reside in the local directory. It is possible that the local directory and LoadLeveler’s home directory are the same.</p> <ul style="list-style-type: none"> • COMM • EXECUTE • LOG • SPOOL and HISTORY <p>Tip: To maximize performance, you should keep the log, spool, and execute directories in a local file system. Also, to measure the performance of your network, consider using one of the available products, such as Toolbox/6000.</p>
Release directory	<p>RELEASEDIR</p> <p>The following subdirectories are created during installation and they reside in the release directory. You can change their locations.</p> <ul style="list-style-type: none"> • BIN • LIB

Customizing the configuration file

Table 13. Default locations for all of the files and directories (continued)

To specify the location of the:	Specify this keyword:
Core dump directory	<p>You may specify alternate directories to hold core dumps for the daemons and starter process:</p> <ul style="list-style-type: none"> • MASTER_COREDUMP_DIR • NEGOTIATOR_COREDUMP_DIR • SCHEDD_COREDUMP_DIR • STARTD_COREDUMP_DIR • GSMONITOR_COREDUMP_DIR • KBDD_COREDUMP_DIR • STARTER_COREDUMP_DIR <p>When specifying core dump directories, be sure that the access permissions are set so the LoadLeveler daemon or process can write to the core dump directory. The permissions set for path names specified in the keywords just mentioned must allow writing by both root and the LoadLeveler ID. The permissions set for the path name specified for the STARTER_COREDUMP_DIR keyword must allow writing by root, the LoadLeveler ID, and any user who can submit LoadLeveler jobs.</p> <p>The simplest way to be sure the access permissions are set correctly is to set them the same as are set for the <code>/tmp</code> directory.</p> <p>If a problem with access permissions prevents a LoadLeveler daemon or process from writing to a core dump directory, then a message will be written to the log, and the daemon or process will continue using the default <code>/tmp</code> directory for core files.</p>

For information about configuration file keyword syntax and other details, see Chapter 12, “Configuration file reference,” on page 243.

Configuring recording activity and log files

The LoadLeveler daemons and processes keep log files according to the specifications in the configuration file. A number of keywords are used to describe where LoadLeveler maintains the logs and how much information is recorded in each log. These keywords, shown in Table 14, are repeated in similar form to specify the pathname of the log file, its maximum length, and the debug flags to be used.

“Controlling debugging output” on page 47 describes the events that can be reported through logging controls.

“Saving log files” on page 49 describes the configuration keyword to use to save logs for problem diagnosis.

For information about configuration file keyword syntax and other details, see Chapter 12, “Configuration file reference,” on page 243.

Table 14. Log control statements

Daemon/Process	Log File (required) (See note 1)	Max Length (required) (See note 2)	Debug Control (required) (See note 4)
Master	MASTER_LOG = <i>path</i>	MAX_MASTER_LOG = <i>bytes</i>	MASTER_DEBUG = <i>flags</i>

Table 14. Log control statements (continued)

Daemon/ Process	Log File (required) (See note 1)	Max Length (required) (See note 2)	Debug Control (required) (See note 4)
Schedd	SCHEDD_LOG = <i>path</i>	MAX_SCHEDD_LOG = <i>bytes</i>	SCHEDD_DEBUG = <i>flags</i>
Startd	STARTD_LOG = <i>path</i>	MAX_STARTD_LOG = <i>bytes</i>	STARTD_DEBUG = <i>flags</i>
Starter	STARTER_LOG = <i>path</i>	MAX_STARTER_LOG = <i>bytes</i>	STARTER_DEBUG = <i>flags</i>
Negotiator	NEGOTIATOR_LOG = <i>path</i>	MAX_NEGOTIATOR_LOG = <i>bytes</i>	NEGOTIATOR_DEBUG = <i>flags</i>
Kbdd	KBDD_LOG = <i>path</i>	MAX_KBDD_LOG = <i>bytes</i>	KBDD_DEBUG = <i>flags</i>
GSmonitor	GSMONITOR_LOG = <i>path</i>	MAX_GSMONITOR_LOG = <i>bytes</i>	GSMONITOR_DEBUG = <i>flags</i>

Notes:

1. When coding the *path* for the log files, it is not necessary that all LoadLeveler daemons keep their log files in the same directory, however, you will probably find it a convenient arrangement.
2. There is a maximum length, in bytes, beyond which the various log files cannot grow. Each file is allowed to grow to the specified length and is then saved to an **.old** file. The **.old** files are overwritten each time the log is saved, thus the maximum space devoted to logging for any one program will be twice the maximum length of its log file. The default length is 64KB. To obtain records over a longer period of time, that don't get overwritten, you can use the SAVELOGS keyword in the local or global configuration files. See "Saving log files" on page 49 for more information on extended capturing of LoadLeveler logs.

You can also specify that the log file be started anew with every invocation of the daemon by setting the TRUNC statement to **true** as follows:

- TRUNC_MASTER_LOG_ON_OPEN = true | false
- TRUNC_STARTD_LOG_ON_OPEN = true | false
- TRUNC_SCHEDD_LOG_ON_OPEN = true | false
- TRUNC_KBDD_LOG_ON_OPEN = true | false
- TRUNC_STARTER_LOG_ON_OPEN = true | false
- TRUNC_NEGOTIATOR_LOG_ON_OPEN = true | false
- TRUNC_GSMONITOR_LOG_ON_OPEN = true | false

3. LoadLeveler creates temporary log files used by the **starter** daemon. These files are used for synchronization purposes. When a job starts, a **StarterLog.pid** file is created. When the job ends, this file is appended to the **StarterLog** file.
4. Normally, only those who are installing or debugging LoadLeveler will need to use the debug flags, described in "Controlling debugging output" The default error logging, obtained by leaving the right side of the debug control statement null, will be sufficient for most installations.

Controlling debugging output

You can control the level of debugging output logged by LoadLeveler programs. The following flags are presented here for your information, though they are used primarily by IBM personnel for debugging purposes:

D_ACCOUNT

Logs accounting information about processes. If used, it may slow down the network.

D_ACCOUNT_DETAIL

Logs detailed accounting information about processes. If used, it may slow down the network and increase the size of log files.

D_ADAPTER

Logs messages related to adapters.

D_AFS

Logs information related to AFS credentials.

Customizing the configuration file

- D_CKPT**
Logs information related to checkpoint and restart
- D_DAEMON**
Logs information regarding basic daemon set up and operation, including information on the communication between daemons.
- D_DBX**
Bypasses certain signal settings to permit debugging of the processes as they execute in certain critical regions.
- D_EXPR**
Logs steps in parsing and evaluating control expressions.
- D_FAIRSHARE**
Displays messages related to fair share scheduling in the daemon logs. In the global configuration file, **D_FAIRSHARE** can be added to **SCHEDD_DEBUG** and **NEGOTIATOR_DEBUG**.
- D_FULLDEBUG**
Logs details about most actions performed by each daemon but doesn't log as much activity as setting all the flags.
- D_HIERARCHICAL**
Used to enable messages relating to problems related to the transmission of hierarchical messages. A hierarchical message is sent from an originating node to lower ranked receiving nodes.
- D_JOB**
Logs job requirements and preferences when making decisions regarding whether a particular job should run on a particular machine.
- D_KERNEL**
Activates diagnostics for errors involving the process tracking kernel extension.
- D_LOAD**
Displays the load average on the startd machine.
- D_LOCKING**
Logs requests to acquire and release locks.
- D_LXCPUAFNT**
Logs messages related to Linux CPU affinity. This flag is only valid for the **startd** daemon.
- D_MACHINE**
Logs machine control functions and variables when making decisions regarding starting, suspending, resuming, and aborting remote jobs.
- D_MUSTER**
Logs information related to multicluster processing.
- D_NEGOTIATE**
Displays the process of looking for a job to run in the negotiator. It only pertains to this daemon.
- D_PCRED**
Directs that extra debug should be written to a file if the **setpcred()** function call fails.
- D_PROC**
Logs information about jobs being started remotely such as the number of bytes fetched and stored for each job.
- D_QUEUE**
Logs changes to the job queue.
- D_REFCOUNT**
Logs activity associated with reference counting of internal LoadLeveler objects.

D_RESERVATION

Logs reservation information in the negotiator and Schedd daemon logs. **D_RESERVATION** can be added to **SCHEDD_DEBUG** and **NEGOTIATOR_DEBUG**.

D_RESOURCE

Logs messages about the management and consumption of resources. These messages are recorded in the negotiator log.

D_SCHEDD

Displays how the Schedd works internally.

D_SDO

Displays messages detailing LoadLeveler objects being transmitted between daemons and commands.

D_SECURITY

Logs information related to Cluster Security (CtSec) services identities.

D_STANZAS

Displays internal information about the parsing of the administration file.

D_STARTD

Displays how the startd works internally.

D_STARTER

Displays how the starter works internally.

D_STREAM

Displays messages detailing socket I/O.

D_SWITCH

Logs entries related to switch activity and LoadLeveler Switch Table Object data.

D_THREAD

Displays the ID of the thread producing the log message. The thread ID is displayed immediately following the date and time. This flag is useful for debugging threaded daemons.

D_XDR

Logs information regarding External Data Representation (XDR) communication protocols.

For example:

```
SCHEDD_DEBUG = D_CKPT D_XDR
```

Causes the scheduler to log information about checkpointing user jobs and exchange xdr messages with other LoadLeveler daemons. These flags will primarily be of interest to LoadLeveler implementers and debuggers.

The **LL_COMMAND_DEBUG** environment variable can be set to a string of debug flags the same way as the ***_DEBUG** configuration keywords are set. Normally, LoadLeveler commands and APIs do not print debug messages, but with this environment variable set, the requested classes of debugging messages will be logged to stderr. For example:

```
LL_COMMAND_DEBUG="D_ALWAYS D_STREAM" llstatus
```

will cause the **llstatus** command to print out debug messages related to I/O to stderr.

Saving log files

By default, LoadLeveler stores only the two most recent iterations of a daemon's log file (*<daemon name>Log*, and *<daemon name>Log.old*). Occasionally, for problem diagnosing, users will need to capture LoadLeveler logs over an extended period. Users can specify that all log files be saved to a particular directory by using the **SAVELOGS** keyword in a local or global configuration file. Be aware that

Customizing the configuration file

LoadLeveler does not provide any way to manage and clean out all of those log files, so users must be sure to specify a directory in a file system with enough space to accommodate them. This file system should be separate from the one used for the LoadLeveler log, spool, and execute directories.

Each log file is represented by the name of the daemon that generated it, the exact time the file was generated, and the name of the machine on which the daemon is running. When you list the contents of the **SAVELOGS** directory, the list of log file names looks like this:

```
NegotiatorLogNov02.16:10:39.123456.c163n10.ppd.pok.ibm.com
NegotiatorLogNov02.16:10:42.987654.c163n10.ppd.pok.ibm.com
NegotiatorLogNov02.16:10:46.564123.c163n10.ppd.pok.ibm.com
NegotiatorLogNov02.16:10:48.234345.c163n10.ppd.pok.ibm.com
NegotiatorLogNov02.16:10:51.123456.c163n10.ppd.pok.ibm.com
NegotiatorLogNov02.16:10:53.566987.c163n10.ppd.pok.ibm.com
StarterLogNov02.16:09:19.622387.c163n10.ppd.pok.ibm.com
StarterLogNov02.16:09:51.499823.c163n10.ppd.pok.ibm.com
StarterLogNov02.16:10:30.876546.c163n10.ppd.pok.ibm.com
SchedLogNov02.16:09:05.543677.c163n10.ppd.pok.ibm.com
SchedLogNov02.16:09:26.688901.c163n10.ppd.pok.ibm.com
SchedLogNov02.16:09:47.443556.c163n10.ppd.pok.ibm.com
SchedLogNov02.16:10:12.712680.c163n10.ppd.pok.ibm.com
SchedLogNov02.16:10:37.342156.c163n10.ppd.pok.ibm.com
StartLogNov02.16:09:05.697753.c163n10.ppd.pok.ibm.com
StartLogNov02.16:09:26.881234.c163n10.ppd.pok.ibm.com
StartLogNov02.16:09:47.231234.c163n10.ppd.pok.ibm.com
StartLogNov02.16:10:12.125556.c163n10.ppd.pok.ibm.com
StartLogNov02.16:10:37.961486.c163n10.ppd.pok.ibm.com
```

For information about configuration file keyword syntax and other details, see Chapter 12, “Configuration file reference,” on page 243.

Setting up file system monitoring

You can use the file system keywords to monitor the file system space or inodes used by LoadLeveler for:

- Logs
- Saving executables
- Spool information
- History files

You can also use the file system keywords to take preventive action and avoid problems caused by running out of file system space or inodes. This is done by setting the frequency that LoadLeveler checks the file system free space or inodes and by setting the upper and lower thresholds that initialize system responses to the free space or inodes available. By setting a realistic span between the lower and upper thresholds, you will avoid excessive system actions.

The file system monitoring keywords are:

- **FS_INTERVAL**
- **FS_NOTIFY**
- **FS_SUSPEND**
- **FS_TERMINATE**
- **INODE_NOTIFY**
- **INODE_SUSPEND**
- **INODE_TERMINATE**

For information about configuration file keyword syntax and other details, see Chapter 12, “Configuration file reference,” on page 243.

Defining LoadLeveler machine characteristics

You can use the following keywords to define the characteristics of machines in the LoadLeveler cluster. For information about configuration file keyword syntax and other details, see Chapter 12, “Configuration file reference,” on page 243.

- ARCH
- CLASS
- CUSTOM_METRIC
- CUSTOM_METRIC_COMMAND
- FEATURE
- GSMONITOR_RUNS_HERE
- MAX_STARTERS
- SCHEDD_RUNS_HERE
- SCHEDD_SUBMIT_AFFINITY
- STARTD_RUNS_HERE
- START_DAEMONS
- VM_IMAGE_ALGORITHM
- X_RUNS_HERE

Defining job classes that a LoadLeveler machine will accept

The following examples illustrate possible ways of defining job classes.

- **Example 1**

This example specifies multiple classes:

```
Class = No_Class(2)
```

or

```
Class = { "No_Class" "No_Class" }
```

The machine will only run jobs that have either defaulted to or explicitly requested class **No_Class**. A maximum of two LoadLeveler jobs are permitted to run simultaneously on the machine if the **MAX_STARTERS** keyword is not specified. See “Specifying how many jobs a machine can run” on page 52 for more information on **MAX_STARTERS**.

- **Example 2**

This example specifies multiple classes:

```
Class = No_Class(1) Small(1) Medium(1) Large(1)
```

or

```
Class = { "No_Class" "Small" "Medium" "Large" }
```

The machine will only run a maximum of four LoadLeveler jobs that have either defaulted to, or explicitly requested **No_Class**, **Small**, **Medium**, or **Large** class. A LoadLeveler job with class **IO_bound**, for example, would not be eligible to run here.

- **Example 3**

This example specifies multiple classes:

```
Class = B(2) D(1)
```

or

```
Class = { "B" "B" "D" }
```

Customizing the configuration file

The machine will run only LoadLeveler jobs that have explicitly requested class **B** or **D**. Up to three LoadLeveler jobs may run simultaneously: two of class **B** and one of class **D**. A LoadLeveler job with class **No_Class**, for example, would not be eligible to run here.

Specifying how many jobs a machine can run

To specify how many jobs a machine can run, you need to take into consideration both the **MAX_STARTERS** keyword and the **Class** statement, which described in more detail in “Defining LoadLeveler machine characteristics” on page 51.

For example, if the configuration file contains these statements:

```
Class = A(1) B(2) C(1)
MAX_STARTERS = 2
```

then the machine can run a maximum of two LoadLeveler jobs simultaneously. The possible combinations of LoadLeveler jobs are:

- A and B
- A and C
- B and B
- B and C
- Only A, or only B, or only C

If this keyword is specified together with a **Class** statement, the maximum number of jobs that can be run is equal to the lower of the two numbers. For example, if:

```
MAX_STARTERS = 2
Class = class_a(1)
```

then the maximum number of job steps that can be run is one (the **Class** statement defines one class).

If you specify **MAX_STARTERS** keyword without specifying a **Class** statement, by default one class still exists (called **No_Class**). Therefore, the maximum number of jobs that can be run when you do not specify a **Class** statement is one.

Note: If the **MAX_STARTERS** keyword is not defined in either the global configuration file or the local configuration file, the maximum number of jobs that the machine can run is equal to the number of classes in the **Class** statement.

Defining security mechanisms

LoadLeveler can be configured to control authentication and authorization of LoadLeveler functions by using Cluster Security (CtSec) services, a subcomponent of Reliable Scalable Cluster Technology (RSCT), which uses the host-based authentication (HBA) as an underlying security mechanism.

LoadLeveler permits only one security service to be configured at a time. You can skip this topic if you do not plan to use this security feature or if you plan to use the credential forwarding provided by the **llgetdce** and **llsetdce** program pair. Refer to “Using the alternative program pair: llgetdce and llsetdce” on page 68 for more information.

LoadLeveler for Linux does not support CtSec security.

LoadLeveler can be enabled to interact with OpenSSL for secure multicluster communications

Table 15 lists the topics that explain LoadLeveler daemons and how you may define their characteristics and modify their behavior.

Table 15. Roadmap of configuration tasks for securing LoadLeveler operations

To learn about:	Read the following:
Securing LoadLeveler operations using cluster security services	<ul style="list-style-type: none"> • “Configuring LoadLeveler to use cluster security services” • “Steps for enabling CtSec services” on page 54 • “Limiting which security mechanisms LoadLeveler can use” on page 56
Enabling LoadLeveler to secure multicluster communication with OpenSSL	“Steps for securing communications within a LoadLeveler multicluster” on page 143
Correctly specifying configuration file keywords	Chapter 12, “Configuration file reference,” on page 243

Configuring LoadLeveler to use cluster security services

Cluster security (CtSec) services allows a software component to authenticate and authorize the identity of one of its peers or clients.

When configured to use CtSec services, LoadLeveler will:

- Authenticate the identity of users and programs interacting with LoadLeveler.
- Authorize users and programs to use LoadLeveler services. It prevents unauthorized users and programs from misusing resources or disrupting services.

To use CtSec services, all nodes running LoadLeveler must first be configured as part of a cluster running Reliable Scalable Cluster Technology (RSCT). For details on CtSec services administration, see *IBM Reliable Scalable Cluster Technology for AIX 5L and Linux Administration Guide*, SA22-7889.

CtSec services are designed to use multiple security mechanisms and each security mechanism must be configured for LoadLeveler. At the present time, directions are provided only for configuring the host-based authentication (HBA) security mechanism for LoadLeveler’s use. If CtSec is configured to use additional security mechanisms that are not configured for LoadLeveler’s use, then the LoadLeveler configuration file keyword **SEC_IMPOSED_MECHS** must be specified. This keyword is used to limit the security mechanisms that will be used by CtSec services to only those that are configured for use by LoadLeveler.

Authorization is based on user identity. When CtSec services are enabled for LoadLeveler, user identity will differ depending on the authentication mechanism in use. A user’s identity in UNIX host-based authentication is the user’s network identity which is comprised of the user name and host name, such as user_name@host.

LoadLeveler uses CtSec services to authorize owners of jobs, administrators and LoadLeveler daemons to perform certain actions. CtSec services uses its own identity mapping file to map the clients’ network identity to a local identity when performing authorizations. A typical local identity is the user name without a

Customizing the configuration file

hostname. The local identities of the LoadLeveler administrators must be added as members of the group specified by the keyword **SEC_ADMIN_GROUP**. The local identity of the LoadLeveler user name must be added as the sole member of the group specified by the keyword **SEC_SERVICES_GROUP**. The LoadLeveler Services and Administrative groups, those identified by the keywords **SEC_SERVICES_GROUP** and **SEC_ADMIN_GROUP**, must be the same across all nodes in the LoadLeveler cluster. To ensure consistency in performing tasks which require owner, administrative or daemon privileges across all nodes in the LoadLeveler cluster, user network identities must be mapped identically across all nodes in the LoadLeveler cluster. If this is not the case, LoadLeveler authorizations may fail.

Steps for enabling CtSec services

To enable LoadLeveler to use CtSec services, perform the following steps:

1. Include, in the Trusted Host List, the host names of all hosts with which communications may take place. If LoadLeveler tries to communicate with a host not on the Trusted Host List the message: The host identified in the credentials is not a trusted host on this system will occur. Additionally, the system administrator should ensure that public keys are manually exchanged between all hosts in the LoadLeveler cluster. Refer to *IBM Reliable Scalable Cluster Technology for AIX 5L and Linux Administration Guide, SA22-7889* for information on setting up Trusted Host Lists and manually transferring public keys.
2. Create user IDs. Each LoadLeveler administrator and the LoadLeveler user ID need to be created, if they don't already exist. You can do this through SMIT or the **mkuser** command.
3. The **unix.map** file must contain the correct value for the service name **ctloadl** which specifies the LoadLeveler user name. If you have configured LoadLeveler to use **loadl** as the LoadLeveler user name, either by default or by specifying **loadl** in the **LoadLUserid** keyword of the **/etc/LoadL.cfg** file, nothing needs to be done. The default map file will contain the **ctloadl** service name already assigned to **loadl**. If you have configured a different user name in the **LoadLUserid** keyword of the **/etc/LoadL.cfg** file, you will need to make sure that the **/var/ct/cfg/unix.map** file exists and that it assigns the same user name to the **ctloadl** service name. If the **/var/ct/cfg/unix.map** file does not exist, create one by copying the default map file **/usr/sbin/rsct/cfg/unix.map**. Do not modify the default map file.

If the value of the **LoadLUserid** and the value associated with **ctloadl** are not the same a security services error which indicates a UNIX identity mismatch will occur.

4. To map network identities to local identities, add entries to the global mapping file of each machine in the LoadLeveler cluster. This file is located at: **/var/ct/cfg/ctsec_map.global**. If this file doesn't yet exist, you should copy the default global mapping file to this location—don't modify the default mapping file. The default global mapping file, which is shared among all CtSec services exploiters, is located at **/usr/sbin/rsct/cfg/ctsec_map.global**. See *IBM Reliable Scalable Cluster Technology for AIX 5L: Technical Reference, SA22-78900* for more information on the mapping file.

When adding names to the global mapping file, enter more specific entries ahead of the other, less specific entries. Remember that you must update the global mapping file on each machine in the LoadLeveler cluster, and each mapping file has to be updated with the security services identity of each member of the **administrator** group, the **services** group, and the users.

Therefore, you would have entries like this:

```

unix:brad@mach1.pok.ibm.com=bradleyf
unix:brad@mach2.pok.ibm.com=bradleyf
unix:brad@mach3.pok.ibm.com=bradleyf
unix:marsha@mach2.pok.ibm.com=marshab
unix:marsha@mach3.pok.ibm.com=marshab
unix:load1@mach1.pok.ibm.com=load1
unix:load1@mach2.pok.ibm.com=load1
unix:load1@mach3.pok.ibm.com=load1

```

However, if you're sure your LoadLeveler cluster is secure, you could specify mapping for all machines this way:

```

unix:brad@*=bradleyf
unix:marsha@*=marshab
unix:load1@*=load1

```

This indicates that the UNIX network identity of the users **brad**, **marsha** and **load1** will map to their respective security services identities on every machine in the cluster. Refer to *IBM Reliable Scalable Cluster Technology for AIX 5L: RSCT Technical Reference, SA22-7800* for a description of the syntax used in the **ctsec_map.global** file.

5. Create UNIX groups. The LoadLeveler **administrator** group and **services** group need to be created for every machine in the cluster and should contain the local identities of members. This can be done either by using SMIT or the **mkgroup** command.

For example, to create the group **lladmin** which lists the LoadLeveler administrators:

```
mkgroup "users=sam,betty,load1" lladmin
```

These groups must be created on each machine in the LoadLeveler cluster and must contain the same entries.

To create the group **llsvcs** which lists the identity under which LoadLeveler daemons run using the default id of **load1**:

```
mkgroup users=load1 llsvcs
```

These groups must be created on each machine in the LoadLeveler cluster and must contain the same entries.

6. Add or update these keywords in the LoadLeveler configuration file:

```

SEC_ENABLEMENT=CTSEC
SEC_ADMIN_GROUP=name of lladmin group
SEC_SERVICES_GROUP=group name that contains identities of LoadLeveler daemons

```

The **SEC_ENABLEMENT=CTSEC** keyword indicates that CtSec services mechanism should be used. **SEC_ADMIN_GROUP** points to the name of the UNIX group which contains the local identities of the LoadLeveler administrators. The **SEC_SERVICES_GROUP** keyword points to the name of the UNIX group which contains the local identity of the LoadLeveler daemons. All LoadLeveler daemons run as the LoadLeveler user ID. Refer to step 5 for discussion of the **administrators** and **services** groups.

7. Update the **.rhosts** file in each user's home directory. This file is used to identify which UNIX identities can run LoadLeveler jobs on the local host machine. If the file does not exist in a user's home directory, you must create it. The **.rhosts** file must contain entries which specify all host and user combinations allowed to submit jobs which will run as the local user, either explicitly or through the use of wildcards.

Entries in the **.rhosts** file are specified this way:

```
HostNameField [UserNameField]
```

Customizing the configuration file

Refer to *IBM AIX Files Reference*, SC23-4168 for further details about the `.rhosts` file format.

Tips for configuring LoadLeveler to use CtSec services: When using CtSec services for LoadLeveler, each machine in the LoadLeveler cluster must be set up properly. CtSec authenticates network identities based on trust established between individual machines in a cluster, based on local host configurations. Because of this it is possible for most of the cluster to run correctly but to have transactions from certain machines experience authentication or authorization problems.

If unexpected authentication or authorization problems occur in a LoadLeveler cluster with CtSec enabled, check that the steps in “Steps for enabling CtSec services” on page 54 were correctly followed for each machine in the LoadLeveler cluster.

If any machine in a LoadLeveler cluster is improperly configured to run CtSec you may see that:

- Users cannot perform user tasks (such as cancel) for jobs they submitted.
Either the machine the job was submitted from or the machine the user operation was submitted from (or both) do not contain mapping files for the user that specify the same security services identity. The user should attempt the operation from the same machine the job was submitted from and record the results. If the user still cannot perform a user task on a job they submitted, then they should contact the LoadLeveler administrator who should review the steps in “Steps for enabling CtSec services” on page 54.
- LoadLeveler daemons fail to communicate.
When LoadLeveler daemons communicate they must first authenticate each other. If the daemons cannot authenticate a message will be put in the daemon log indicating an authentication failure. Ensure the Trusted Hosts List on all LoadLeveler nodes contains the correct entries for all of the nodes in the LoadLeveler cluster. Also, make sure that the LoadLeveler Services group on all nodes of the LoadLeveler cluster contains the local identity for the LoadLeveler user name. The `ctsec_map.global` must contain mapping rules to map the LoadLeveler user name from every machine in the LoadLeveler cluster to the local identity for the LoadLeveler user name. An example of what may happen when daemons fail to communicate is that an Alternate Central Manager may take over while the Primary Central Manager is still active. This can occur when the Alternate Central Manager does not trust the Primary Central Manager.

Limiting which security mechanisms LoadLeveler can use

As more security mechanisms become available, they must be configured for LoadLeveler’s use. If there are security mechanisms configured for CtSec that are not configured for LoadLeveler’s use, then the LoadLeveler configuration file keyword `SEC_IMPOSED_MECHS` must specify the mechanisms configured for LoadLeveler.

Defining usage policies for consumable resources

The LoadLeveler scheduler can schedule jobs based on the availability of consumable resources. You can use the following keywords to configure consumable resources:

- `ENFORCE_RESOURCE_MEMORY`
- `ENFORCE_RESOURCE_POLICY`
- `ENFORCE_RESOURCE_SUBMISSION`
- `ENFORCE_RESOURCE_USAGE`

- FLOATING_RESOURCES
- RESOURCES
- SCHEDULE_BY_RESOURCES

For information about configuration file keyword syntax and other details, see Chapter 12, “Configuration file reference,” on page 243.

Enabling support for bulk data transfer and rCxt blocks

On AIX systems with device drivers and network adapters that support remote direct-memory access (RDMA), LoadLeveler supports bulk data transfer for jobs that use either the Internet or User Space communication protocol mode. For jobs using the Internet protocol (IP jobs), LoadLeveler does not monitor or control the use of bulk transfer. For User Space jobs that request bulk transfer, however, LoadLeveler creates a consumable RDMA resource, and limits RDMA resources to only four for a single machine.

You do not need to perform specific configuration or job-definition tasks to enable bulk transfer for LoadLeveler jobs that use the IP network protocol. LoadLeveler cannot affect whether IP communication uses bulk transfer; the implementation of IP where the job runs determines whether bulk transfer is supported.

To enable User Space jobs to use bulk data transfer, however, you must update the LoadLeveler configuration file to include the value RDMA in the SCHEDULE_BY_RESOURCES list.

Example:

```
SCHEDULE_BY_RESOURCES = RDMA others
```

For additional information about using bulk data transfer and job-definition requirements, see “Using bulk data transfer” on page 173.

Gathering job accounting data

Your organization may have a policy of charging users or groups of users for the amount of resources that their jobs consume. You can do this using LoadLeveler’s accounting feature. Using this feature, you can produce accounting reports that contain job resource information for completed serial and parallel jobs. You can also view job resource information on jobs that are continuing to run.

The following keywords allow you to control accounting functions:

- ACCT
- ACCT_VALIDATION
- GLOBAL_HISTORY
- HISTORY_PERMISSION
- JOB_ACCT_Q_POLICY
- JOB_LIMIT_POLICY

For example, the following section of the configuration file specifies that the accounting function is turned on. It also identifies the default module used to perform account validation and the directory containing the global history files:

```
ACCT                = A_ON A_VALIDATE
ACCT_VALIDATION    = $(BIN)/llacctval
GLOBAL_HISTORY     = $(SPOOL)
```

Customizing the configuration file

Table 16 lists the topics related to configuring, gathering and using job accounting data.

Table 16. Roadmap of tasks for gathering job accounting data

To learn about:	Read the following:
Configuring LoadLeveler to gather job accounting data	<ul style="list-style-type: none">• “Collecting job resource data on serial and parallel jobs”• “Collecting job resource data based on machines”• “Collecting job resource data based on events” on page 59• “Collecting job resource information based on user accounts” on page 59• “Collecting accounting data for reservations” on page 126• “Collecting the accounting information and storing it into files” on page 60• “64-bit support for accounting functions” on page 61• “Example: Setting up job accounting files” on page 61
Managing accounting data	<ul style="list-style-type: none">• “Producing accounting reports” on page 60• “Correlating AIX and LoadLeveler accounting records” on page 61• “llacctmrg - Collect machine history files” on page 387• “llsummary - Return job resource information for accounting” on page 496
Correctly specifying configuration file keywords	Chapter 12, “Configuration file reference,” on page 243

Collecting job resource data on serial and parallel jobs

Information on completed serial and parallel jobs is gathered using the UNIX *wait3* system call. Information on non-completed serial and parallel jobs is gathered in a platform-dependent manner by examining data from the UNIX process.

Accounting information on a completed serial job is determined by accumulating resources consumed by that job on the machines that ran the job. Similarly, accounting information on completed parallel jobs is gathered by accumulating resources used on all of the nodes that ran the job.

You can also view resource consumption information on serial and parallel jobs that are still running by specifying the *-x* option of the *llq* command. To enable *llq -x*, specify the following keywords in the configuration file:

- **ACCT = A_ON A_DETAIL**
- **JOB_ACCT_Q_POLICY = number**

Collecting job resource data based on machines

LoadLeveler can collect job resource usage information for every machine on which a job may run. A job may run on more than one machine because it is a parallel job or because the job is vacated from one machine and rescheduled to another machine.

To enable recording of resources by machine, you need to specify **ACCT = A_ON A_DETAIL** in the configuration file.

The machine’s speed is part of the data collected. With this information, an installation can develop a charge back program which can charge more or less for

resources consumed by a job on different machines. For more information on a machine's speed, refer to the machine stanza information. See "Defining machines" on page 78.

Collecting job resource data based on events

In addition to collecting job resource information based upon machines used, you can gather this information based upon an event or time that you specify. For example, you may want to collect accounting information at the end of every work shift or at the end of every week or month. To collect accounting information on all machines in this manner, use the `llctl` command with the `capture` parameter:

```
llctl -g capture eventname
```

eventname is any string of continuous characters (no white space is allowed) that defines the event about which you are collecting accounting data. For example, if you were collecting accounting data on the *graveyard* work shift, your command could be:

```
llctl -g capture graveyard
```

This command allows you to obtain a snapshot of the resources consumed by active jobs up to and including the moment when you issued the command. If you want to capture this type of information on a regular basis, you can set up a crontab entry to invoke this command regularly. For example:

```
# sample crontab for accounting
# shift crontab 94/8/5
#
# Set up three shifts, first, second, and graveyard shift.
# Crontab entries indicate the end of shift.
#
#M H d m day command
#
00 08 * * * /u/load1/bin/llctl -g capture graveyard
00 16 * * * /u/load1/bin/llctl -g capture first
00 00 * * * /u/load1/bin/llctl -g capture second
```

For more information on the `llctl` command, refer to "llctl - Control LoadLeveler daemons" on page 409. For more information on the collection of accounting records, see "llq - Query job status" on page 449.

Collecting job resource information based on user accounts

If your installation is interested in keeping track of resources used on an account basis, you can require all users to specify an account number in their job command files. They can specify this account number with the `account_no` keyword which is explained in detail in "Job command file keyword descriptions" on page 336. Interactive POE jobs can specify an account number using the `LOADL_ACCOUNT_NO` environment variable.

LoadLeveler validates this account number by comparing it against a list of account numbers specified for the user in the user stanza in the administration file.

Account validation is under the control of the `ACCT` keyword in the configuration file. The routine that performs the validation is called `llacctval`. You can supply your own validation routine by specifying the `ACCT_VALIDATION` keyword in the configuration file. The following are passed as character string arguments to the validation routine:

- User name
- User's login group name

Customizing the configuration file

- Account number specified on the Job
- Blank separated list of account numbers obtained from the user's stanza in the administration file.

The account validation routine must exit with a return code of zero if the validation succeeds. If it fails, the return code is a nonzero number.

Collecting the accounting information and storing it into files

LoadLeveler stores the accounting information that it collects in a file called *history* in the spool directory of the machine that initially scheduled this job, the Schedd machine. Data on parallel jobs are also stored in the *history* files.

Resource information collected on the LoadLeveler job is constrained by the capabilities of the wait3 system call. Information for processes which fork child processes will include data for those child processes as long as the parent process waits for the child process to terminate. Complete data may not be collected for jobs which are not composed of simple parent/child processes. For example, if you have a LoadLeveler job which invokes an rsh command to execute a function on another machine, the resources consumed on the other machine will not be collected as part of the LoadLeveler accounting data.

LoadLeveler accounting uses the following types of files:

- The local history file which is local to each Schedd machine is where job resource information is first recorded. These files are usually named *history* and are located in the spool directory of each Schedd machine, but you may specify an alternate name with the **HISTORY** keyword in either the global or local configuration file.
- The global history file is a combination of the history files from some or all of the machines in the LoadLeveler cluster merged together. The command **llacctmrg** is used to collect files together into a global file. As the files are collected from each machine, the local history file for that machine is reset to contain no data. The file is named *globalhist.YYYYMMDDHHmm*. You may specify the directory in which to place the file when you invoke the **llacctmrg** command or you can specify the directory with the **GLOBAL_HISTORY** keyword in the configuration file. The default value set up in the sample configuration file is the local spool directory.

Producing accounting reports

You can produce three types of reports using either the local or global history file. These reports are called the *short*, *long*, and *extended* versions. As their names imply, the short version of the report is a brief listing of the resources used by LoadLeveler jobs. The long version provides more comprehensive detail with summarized resource usage, and the extended version of the report provides the comprehensive detail with detailed resource usage.

If you do not specify a report type, you will receive the default short version. The short report displays the number of jobs along with the total CPU usage according to user, class, group, and account number. The extended version of the report displays all of the data collected for every job.

- For examples of the short and extended versions of the report, see "llsummary - Return job resource information for accounting" on page 496.
- For information on the accounting APIs, refer to Chapter 17, "Application programming interfaces (APIs)," on page 503.

Correlating AIX and LoadLeveler accounting records

For jobs running on AIX systems, you can use a job accounting key to correlate AIX accounting records with LoadLeveler accounting records. The job accounting key uniquely identifies each job step. LoadLeveler derives this key from the job key and the date and time at which the job entered the queue (see the QDate variable description). The key is associated with the starter process for the job step and any of its child processes.

For checkpointed jobs, LoadLeveler does not change the job accounting key, regardless of how it restarts the job step. Jobs restarted from a checkpoint file or through a new job step retain the job accounting key for the original job step.

To access the job accounting key for a job step, you can use the following interfaces:

- The **llsummary** command, requesting the long version of the report. For details about using this command, see “llsummary - Return job resource information for accounting” on page 496.
- The **GetHistory** subroutine. For details about using this subroutine, see “GetHistory subroutine” on page 507.
- The **ll_get_data** subroutine, through the **LL_StepAcctKey** specification. For details about using this subroutine, see “ll_get_data subroutine” on page 532.

For information about AIX accounting records, see the system accounting topic in *AIX 5L System Management Guide: Operating System and Devices*.

64-bit support for accounting functions

LoadLeveler 64-bit support for accounting functions includes:

- Statistics of jobs such as usage, limits, consumable resources, and other 64-bit integer data are preserved in the history file as `rusage64`, `rlimit64` structures and as data items of type `int64_t`.
- The `LL_job_step` structure defined in `llapi.h` allows access to the 64-bit data items either as data of type `int64_t` or as data of type `int32_t`. In the latter case, the returned values may be truncated.
- The **llsummary** command displays 64-bit information where appropriate.
- The data access API supports both 64-bit and 32-bit access to accounting and usage information in a history file. See “Examples of using the data access API” on page 584 for an example of how to use the **ll_get_data** subroutine to access information stored in a LoadLeveler history file.

Example: Setting up job accounting files

The sample procedure shown in Table 17 walks you through the process of collecting account data. You can perform all of the steps or just the ones that apply to your situation.

1. Edit the configuration file according to the following table:

Table 17. Collecting account data - modifying the configuration file

Edit this keyword:	To:
ACCT	Turn accounting and account validation on and off and specify detailed accounting.
ACCT_VALIDATION	Specify the account validation routine.
GLOBAL_HISTORY	Specify a directory in which to place the global history files.

Customizing the configuration file

2. Specify account numbers and set up account validation by performing the following steps:
 - a. Specify a list of account numbers a user may use when submitting jobs, by using the **account** keyword in the user stanza in the administration file.
 - b. Instruct users to associate an account number with their job, by using the **account_no** keyword in the job command file.

Alternative: You may use the LoadLeveler GUI to associate account numbers with jobs:

Select **File** → **Build a Job** from the main window.

 - ▲ The Build a Job window appears.

Type The account number in the **account_no** field on the Build a Job window.

Press **OK**

 - ▲ The window closes and you return to the main window.
 - c. Specify the **ACCT_VALIDATION** keyword in the configuration file that identifies the module that will be called to perform account validation. The default module is called **llacctval**. You can replace this module with your installation's own accounting routine by specifying a new module with this keyword.
3. Specify machines and their weights by using the **speed** keyword in a machine's machine stanza in the administration file.

Also, if you have in your cluster machines of differing speeds and you want LoadLeveler accounting information to be normalized for these differences, specify **cpu_speed_scale=true** in each machine's respective machine stanza.

For example, suppose you have a cluster of two machines, called A and B, where Machine B is three times as fast as Machine A. Machine A has **speed=1.0**, and Machine B has **speed=3.0**. Suppose a job runs for 12 CPU seconds on Machine A. The same job runs for 4 CPU seconds on Machine B. When you specify **cpu_speed_scale=true**, the accounting information collected on Machine B for that job shows the normalized value of 12 CPU seconds rather than the actual 4 CPU seconds.
4. Merge multiple files collected from each machine into one file, using the **llacmrg** command.

Alternative: You may use the LoadLeveler GUI to merge the files:

Select A machine from the Machines window

Select **Admin** → **Collect Account Data...** from the Machines window.

 - ▲ A window appears prompting you to enter a directory name where the file will be placed. If no directory is specified, the directory specified with the **GLOBAL_HISTORY** keyword in the global configuration file is the default directory.

Press **OK**

 - ▲ The window closes and you return to the main window.
5. Report job information on all the jobs in the history file, using the **llsummary** command.

Alternative: You may use the LoadLeveler GUI to report job information:

Select **Admin** → **Create Account Report...** from the Machines window.

Note: If you want to receive an extended accounting report, select the **extended** cascading button.

▲ A window appears prompting you to enter the following information:

- A short, long, or extended version of the output. The short version is the default version.
- Start and end date ranges for the report. If no date is specified, the default is to report all of the data in the report.
- The name of the input data file.
- The name of the output data file.

Press OK

▲ The window closes and you return to the main window. The report appears in the Messages window if no output data file was specified.

Managing job status through control expressions

You can control running jobs by using five control functions as Boolean expressions in the configuration file. These functions are useful primarily for serial jobs. You define the expressions, using normal C conventions, with the following functions:

- START
- SUSPEND
- CONTINUE
- VACATE
- KILL

The expressions are evaluated for each job running on a machine using both the job and machine attributes. Some jobs running on a machine may be suspended while others are allowed to continue.

The START expression is evaluated twice; once to see if the machine can accept jobs to run and second to see if the specific job can be run on the machine. The other expressions are evaluated after the jobs have been dispatched and in some cases, already running.

When evaluating the START expression to determine if the machine can accept jobs, **Class != "Z"** evaluates to true only if Z is not in the class definition. This means that if two different classes are defined on a machine, **Class != "Z"** (where Z is one of the defined classes) always evaluates to false when specified in the START expression and, therefore, the machine will not be considered to start jobs.

Typically, machine load average, keyboard activity, time intervals, and job class are used within these various expressions to dynamically control job execution.

For additional information about:

- Time-related variables that you may use for this keyword, see “Variables to use for setting times” on page 299.
- Coding these control expressions in the configuration file, see Chapter 12, “Configuration file reference,” on page 243.

How control expressions affect jobs

After LoadLeveler selects a job for execution, the job can be in any of several states. Figure 10 on page 64 shows how the control expressions can affect the state a job is in. The rectangles represent job or daemon states (Idle, Completed,

Customizing the configuration file

Running, Suspended, and Vacating) and the diamonds represent the control expressions (Start, Suspend, Continue, Vacate, and Kill).

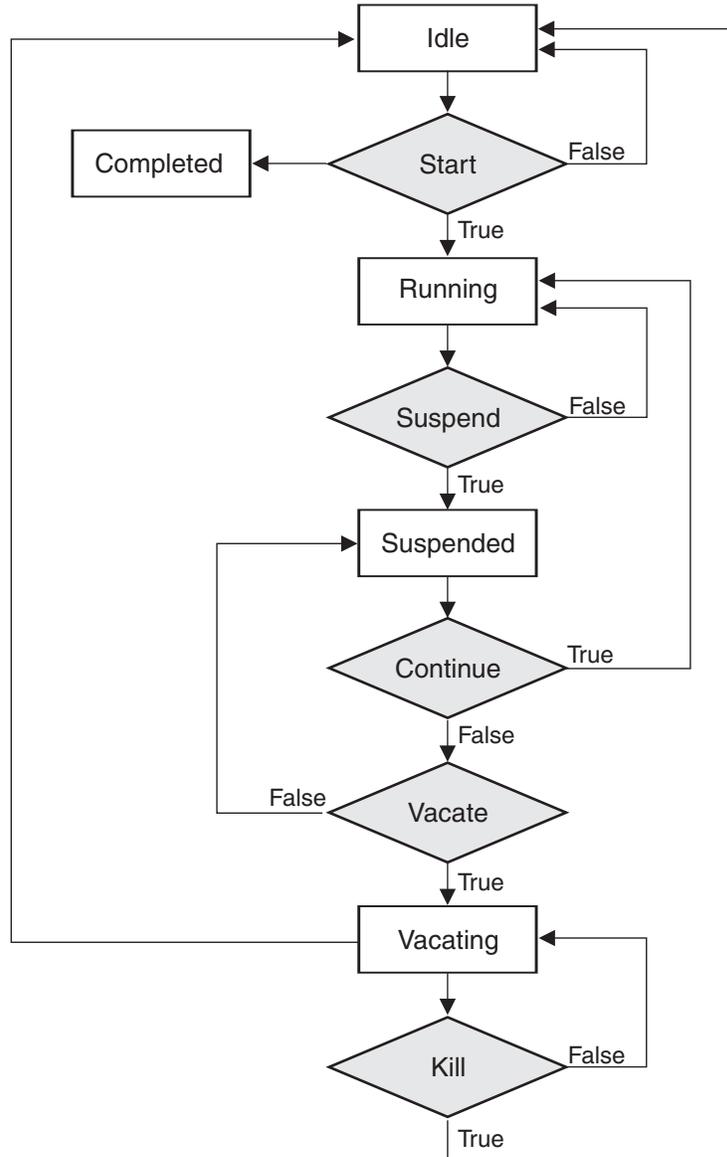


Figure 10. How control expressions affect jobs

Criteria used to determine when a LoadLeveler job will enter Start, Suspend, Continue, Vacate, and Kill states are defined in the LoadLeveler configuration files and they can be different for each machine in the cluster. They can be modified to meet local requirements.

Tracking job processes

When a job terminates, its orphaned processes may continue to consume or hold resources, thereby degrading system performance, or causing jobs to hang or fail. Process tracking allows LoadLeveler to cancel any processes (throughout the entire cluster), left behind when a job terminates. Process tracking is required to do preemption by the suspend method when running either the BACKFILL or API schedulers. Process tracking is optional in all other cases.

When process tracking is enabled, all child processes are terminated when the main process terminates. This will include any background or orphaned processes started in the prolog, epilog, user prolog, and user epilog.

LoadLeveler for Linux does not support process tracking.

There are two keywords used in specifying process tracking:

PROCESS_TRACKING

To activate process tracking, set **PROCESS_TRACKING=TRUE** in the LoadLeveler global configuration file. By default, **PROCESS_TRACKING** is set to **FALSE**.

PROCESS_TRACKING_EXTENSION

This keyword specifies the path to the kernel extension binary **LoadL_pt_ke** in the local or global configuration file. If the **PROCESS_TRACKING_EXTENSION** keyword is not supplied, then LoadLeveler will search the default directory **\$HOME/bin**.

The process tracking kernel extension is not unloaded when the **startd** daemon terminates. Therefore if a mismatch in the version of the loaded kernel extension and the installed kernel extension is found when the **startd** starts up the daemon will exit. In this case a reboot of the node is needed to unload the currently loaded kernel extension. If you install a new version of LoadLeveler which contains a new version of the kernel extension you may need to reboot the node.

For information about configuration file keyword syntax and other details, see Chapter 12, "Configuration file reference," on page 243.

Querying multiple LoadLeveler clusters

This topic applies only to those installations having more than one LoadLeveler cluster, where the separate clusters have not been organized into a multicluster environment. To organize separate LoadLeveler clusters into a multicluster environment, see "LoadLeveler multicluster support" on page 139.

You can query, submit, or cancel jobs in multiple LoadLeveler clusters by setting up a master configuration file for each cluster and using the **LOADL_CONFIG** environment variable to specify the name of the master configuration file that the LoadLeveler commands must use. The master configuration file must be located in the **/etc** directory and the file name must have a format of *base_name.cfg* where *base_name* is a user defined identifier for the cluster.

The default name for the master configuration file is **/etc/LoadL.cfg**. The format for the **LOADL_CONFIG** environment variable is **LOADL_CONFIG=/etc/*base_name.cfg*** or **LOADL_CONFIG=*base_name***. When you use the form **LOADL_CONFIG=*base_name***, the prefix **/etc** and suffix **.cfg** are appended to the *base_name*.

The following example explains how you can set up a machine to query multiple clusters:

You can configure **/etc/LoadL.cfg** to point to the configuration files for the "default" cluster, and you can configure **/etc/othercluster.cfg** to point to the configuration files of another cluster which the user can select.

For example, you can enter the following query command:

Customizing the configuration file

```
$ llq
```

The **llq** command uses the configuration from `/etc/LoadL.cfg` and queries job information from the "default" cluster.

To send a query to the cluster defined in the configuration file of `/etc/othercluster.cfg`, enter:

```
$ env LOADL_CONFIG=othercluster llq
```

Note that the machine from which you issue the **llq** command is considered as a submit-only machine by the other cluster.

Handling switch-table errors

You may use the following configuration file keywords to control how LoadLeveler responds to switch-table errors:

- ACTION_ON_SWITCH_TABLE_ERROR
- DRAIN_ON_SWITCH_TABLE_ERROR
- RESUME_ON_SWITCH_TABLE_ERROR_CLEAR

For information about configuration file keyword syntax and other details, see Chapter 12, "Configuration file reference," on page 243.

Providing additional job-processing controls through installation exits

LoadLeveler allows administrators to further configure the environment through installation exits. Table 18 lists these additional job-processing controls.

Table 18. Roadmap of administrator tasks accomplished through installation exits

To learn about:	Read the following:
Writing a program to control when jobs are scheduled to run	"Controlling the central manager scheduling cycle" on page 67
Writing a pair of programs to override the default LoadLeveler DCE authentication method	"Handling DCE security credentials" on page 68
Writing a program to refresh an AFS token when a job starts	"Handling an AFS token" on page 69
Writing a program to check or modify job requests when they are submitted	"Filtering a job script" on page 70
Writing programs to run before and after job requests	"Writing prolog and epilog programs" on page 70
Overriding the LoadLeveler default mail notification method	"Using your own mail program" on page 75
Defining a cluster metric to determine where a remote job is distributed	See the CLUSTER_METRIC configuration keyword description in Chapter 12, "Configuration file reference," on page 243.
Defining cluster user mapper for multicluster environment	See the CLUSTER_USER_MAPPER configuration keyword description in Chapter 12, "Configuration file reference," on page 243.
Correctly specifying configuration file keywords	Chapter 12, "Configuration file reference," on page 243

Controlling the central manager scheduling cycle

To determine when to run the LoadLeveler scheduling algorithm, the central manager uses the values set in the configuration file for the `NEGOTIATOR_INTERVAL` and the `NEGOTIATOR_CYCLE_DELAY` keywords. The central manager will run the scheduling algorithm every `NEGOTIATOR_INTERVAL` seconds, unless some event takes place such as the completion of a job or the addition of a machine to the cluster. In such cases, the scheduling algorithm is run immediately. When `NEGOTIATOR_CYCLE_DELAY` is set, a minimum of `NEGOTIATOR_CYCLE_DELAY` seconds will pass between the central manager's scheduling attempts, regardless of what other events might take place. When the `NEGOTIATOR_INTERVAL` is set to zero, the central manager will not run the scheduling algorithm until instructed to do so by an authorized process. This setting enables your program to control the central manager's scheduling activity through one of the following:

- The `llrunscheduler` command.
- The `ll_run_scheduler` subroutine.

Both the command and the subroutine instruct the central manager to run the scheduling algorithm.

You might choose to use this setting if, for example, you want to write a program that directly controls the assignment of the system priority for all LoadLeveler jobs. In this particular case, you would complete the following steps to control system priority assignment and the scheduling cycle:

1. Decide the following:
 - Which system priority value to assign to jobs from specific sources or with specific resource requirements.
 - How often the central manager should run the scheduling algorithm. Your program has to be designed to issue the `ll_run_scheduler` subroutine at regular intervals; otherwise, LoadLeveler will not attempt to schedule any job steps.

You also need to understand how changing the system priority affects the job queue. After you successfully use the `ll_modify` subroutine or the `llmodify` command to change system priority values, LoadLeveler will not readjust the values for those job steps when the negotiator recalculates priorities at regular intervals set through the

`NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL` keyword. Also, you can change the system priority for jobs only when those jobs are in the Idle state or a state similar to it. To determine which job states are similar to the Idle state or to the Running state, see the table in "LoadLeveler job states" on page 18.

2. Code a program to use LoadLeveler APIs to perform the following functions:
 - a. Use the Data Access APIs to obtain data about all jobs.
 - b. Determine whether jobs have been added or removed.
 - c. Use the `ll_modify` subroutine to set the system priority for the LoadLeveler jobs. The values you set through this subroutine will not be readjusted when the negotiator recalculates job step priorities.
 - d. Use the `ll_run_scheduler` subroutine to instruct the central manager to run the scheduling algorithm.
 - e. Set a timer for the scheduling interval, to repeat the scheduling instruction at regular intervals. This step is required to replace the effect of setting the configuration keyword `NEGOTIATOR_CYCLE_DELAY`, which LoadLeveler ignores when `NEGOTIATOR_INTERVAL` is set to zero.
3. In the configuration file, set values for the following keywords:

Customizing the configuration file

- Set the **NEGOTIATOR_INTERVAL** keyword to zero to stop the central manager from automatically recalculating system priorities for jobs.
 - (Optional) Set the **SYSPRIO_THRESHOLD_TO_IGNORE_STEP** keyword to specify a threshold value. If the system priority assigned to a job step is less than this threshold value, the job will remain idle.
4. Issue the **llctl** command with either the **reconfig** or **recycle** keyword. Otherwise, LoadLeveler will not process the modifications you made to the configuration file.
 5. (Optional) To make sure that the central manager's automatic scheduling activity has been disabled (by setting the **NEGOTIATOR_INTERVAL** keyword to zero), use the **llstatus** command.
 6. Run your program under a user ID with administrator authority.

Once this procedure is complete, you might want to use one or more of the following commands to make sure that jobs are scheduled according to the correct system priority. The value of *q_sysprio* in command output indicates the system priority for the job step.

- Use the command **llq -s** to detect whether a job step is idle because its system priority is below the value set for the **SYSPRIO_THRESHOLD_TO_IGNORE_STEP** keyword.
- Use the command **llq -l** to display the previous system priority for a job step.
- When unusual circumstances require you to change system priorities manually:
 1. Use the command **llmodify -s** to set the system priority for LoadLeveler jobs. The values you set through this command will not be readjusted when the negotiator recalculates job step priorities.
 2. Use the **llrnscheduler** command to instruct the central manager to run the scheduling algorithm.

Handling DCE security credentials

You can write a pair of programs to override the default LoadLeveler DCE authentication method. To enable the programs, use the **DCE_AUTHENTICATION_PAIR** keyword in your configuration file:

- As an alternative, you can also specify the program pair:

```
DCE_AUTHENTICATION_PAIR = $(BIN)/llgetdce, $(BIN)/llsetdce
```

Specifying the **DCE_AUTHENTICATION_PAIR** keyword enables LoadLeveler support for forwarding DCE credentials to LoadLeveler jobs. You may override the default function provided by LoadLeveler to establish DCE credentials by substituting your own programs.

Using the alternative program pair: llgetdce and llsetdce

The program pair, **llgetdce** and **llsetdce**, forwards DCE credentials by copying credential cache files from the submitting machine to the executing machines. While this technique may require less overhead, it has been known to produce credentials on the executing machines which are not fully capable of being forwarded by rsh commands. This is the only pair of programs offered in earlier releases of LoadLeveler.

Forwarding DCE credentials

An example of a credentials object is a character string containing the DCE principle name and a password. *program1* writes the following to standard output:

- The length of the handle to follow
- The handle

If *program1* encounters errors, it writes error messages to standard error.

program2 receives the following as standard input:

- The length of the handle to follow
- The same handle written by *program1*

program2 writes the following to standard output:

- The length of the login context to follow
- An exportable DCE login context, which is the `idl_byte` array produced from the `sec_login_export_context` DCE API call. For more information, see the DCE Security Services API chapter in the Distributed Computing Environment for AIX Application Development Reference.
- A character string suitable for assigning to the `KRB5CCNAME` environment variable. This string represents the location of the credentials cache established in order for *program2* to export the DCE login context.

If *program2* encounters errors, it writes error messages to standard error. The parent process, the LoadLeveler starter process, writes those messages to the starter log.

For examples of programs that enable DCE security credentials, see the `samples/lldce` subdirectory in the release directory.

Handling an AFS token

You can write a program, run by the scheduler, to refresh an AFS token when a job is started. To invoke the program, use the `AFS_GETNEWTOKEN` keyword in your configuration file.

Before running the program, LoadLeveler sets up standard input and standard output as pipes between the program and LoadLeveler. LoadLeveler also sets up the following environment variables:

LOADL_STEP_OWNER

The owner (UNIX user name) of the job

LOADL_STEP_COMMAND

The name of the command the user's job step invokes.

LOADL_STEP_CLASS

The class this job step will run.

LOADL_STEP_ID

The step identifier, generated by LoadLeveler.

LOADL_JOB_CPU_LIMIT

The number of CPU seconds the job is limited to.

LOADL_WALL_LIMIT

The number of wall clock seconds the job is limited to.

LoadLeveler writes the following current AFS credentials, in order, over the standard input pipe:

- The `ktc_principal` structure indicating the service.
- The `ktc_principal` structure indicating the client.
- The `ktc_token` structure containing the credentials.

The `ktc_principal` structure is defined in the AFS header file `afs_rxkad.h`. The `ktc_token` structure is defined in the AFS header file `afs_auth.h`.

Customizing the configuration file

LoadLeveler expects to read these same structures in the same order from the standard output pipe, except these should be refreshed credentials produced by the installation exit.

The installation exit can modify the passed credentials (to extend their lifetime) and pass them back, or it can obtain new credentials. LoadLeveler takes whatever is returned and uses it to authenticate the user prior to starting the user's job.

Filtering a job script

You can write a program to filter a job script when the job is submitted to the local cluster and when the job is submitted from a remote cluster. This program can, for example, modify defaults or perform site specific verification of parameters. To invoke the local job filter, specify the **SUBMIT_FILTER** keyword in your configuration file. To invoke the remote job filter, specify the **CLUSTER_REMOTE_JOB_FILTER** keyword in your configuration file. For more information on these keywords, see the **SUBMIT_FILTER** or **CLUSTER_REMOTE_JOB_FILTER** keyword in Chapter 12, "Configuration file reference," on page 243.

LoadLeveler sets the following environment variables when the program is invoked:

LOADL_ACTIVE

LoadLeveler version

LOADL_STEP_COMMAND

Job command file name

LOADL_STEP_ID

The job identifier, generated by LoadLeveler

LOADL_STEP_OWNER

The owner (UNIX user name) of the job

For details about specific keyword syntax and use in the configuration file, see Chapter 12, "Configuration file reference," on page 243.

Writing prolog and epilog programs

An administrator can write *prolog* and *epilog* installation exits that can run before and after a LoadLeveler job runs, respectively.

Prolog and epilog programs fall into two types:

- Those that run as the LoadLeveler user ID.
- Those that run in a user's environment.

Depending on the type of processing you want to perform before or after a job runs, specify one or more of the following configuration file keywords, in any combination:

- To run a prolog or epilog program under the LoadLeveler user ID, specify **JOB_PROLOG** or **JOB_EPILOG**, respectively.
- To run a prolog or epilog program under the user's environment, specify **JOB_USER_PROLOG** or **JOB_USER_EPILOG**, respectively.

You do not have to provide a prolog/epilog pair of programs. You may, for example, use only a prolog program that runs under the LoadLeveler user ID.

For details about specific keyword syntax and use in the configuration file, see Chapter 12, "Configuration file reference," on page 243.

Note: If process tracking is enabled and your prolog or epilog program invokes the **mailx** command, set the **sendwait** variable to prevent the background mail process from being killed by process tracking.

A user environment prolog or epilog runs with AFS authentication if installed and enabled. For security reasons, you must code these programs on the machines where the job runs *and* on the machine that schedules the job. If you do not define a value for these keywords, the user environment prolog and epilog settings on the executing machine are ignored.

The user environment prolog and epilog can set environment variables for the job by sending information to standard output in the following format:

```
env id = value
```

Where:

id Is the name of the environment variable

value Is the value (setting) of the environment variable

For example, the user environment prolog sets the environment variable **STAGE_HOST** for the job:

```
#!/bin/sh
echo env STAGE_HOST=shd22
```

Coding conventions for prolog programs

The prolog program is invoked by the starter process. Once the starter process invokes the prolog program, the program obtains information about the job from environment variables.

Syntax:

```
prolog_program
```

Where *prolog_program* is the name of the prolog program as defined in the **JOB_PROLOG** keyword.

No arguments are passed to the program, but several environment variables are set. For more information on these environment variables, see “Run-time environment variables” on page 374.

The real and effective user ID of the prolog process is the LoadLeveler user ID. If the prolog program requires root authority, the administrator must write a secure C or Perl program to perform the desired actions. You should *not* use shell scripts with set uid permissions, since these scripts may make your system susceptible to security problems.

Return code values:

0 The job will begin.

If the prolog program is ended with a signal, the job does not begin and a message is written to the starter log.

Sample prolog programs:

- **Sample of a prolog program for korn shell**

```
#!/bin/ksh
#
# Set up environment
set -a
```

Customizing the configuration file

```
. /etc/environment
. /.profile
export PATH="$PATH:/loctools/lladmin/bin"
export LOG="/tmp/$LOADL_STEP_OWNER.$LOADL_STEP_ID.prolog"
#
# Do set up based upon job step class
#
case $LOADL_STEP_CLASS in
  # A OSL job is about to run, make sure the osl filesystem is
  # mounted. If status is negative then filesystem cannot be
  # mounted and the job step should not run.
  "OSL")
    mount_osl_files >> $LOG
    if [ status = 0 ]
      then EXIT_CODE=1
    else
      EXIT_CODE=0
    fi
    ;;
  # A simulation job is about to run, simulation data has to
  # be made available to the job. The status from copy script must
  # be zero or job step cannot run.
  "sim")
    copy_sim_data >> $LOG
    if [ status = 0 ]
      then EXIT_CODE=0
    else
      EXIT_CODE=1
    fi
    ;;
  # All other job will require free space in /tmp, make sure
  # enough space is available.
  *)
    check_tmp >> $LOG
    EXIT_CODE=$?
    ;;
esac
# The job step will run only if EXIT_CODE == 0
exit $EXIT_CODE
```

- **Sample of a prolog program for C shell**

```
#!/bin/csh
#
# Set up environment
source /u/load1/.login
#
setenv PATH "${PATH}:/loctools/lladmin/bin"
setenv LOG "/tmp/${LOADL_STEP_OWNER}.${LOADL_STEP_ID}.prolog"
#
# Do set up based upon job step class
#
switch ($LOADL_STEP_CLASS)
  # A OSL job is about to run, make sure the osl filesystem is
  # mounted. If status is negative then filesystem cannot be
  # mounted and the job step should not run.
  case "OSL":
    mount_osl_files >> $LOG
    if ( $status < 0 ) then
      set EXIT_CODE = 1
    else
      set EXIT_CODE = 0
    endif
    breaksw
  # A simulation job is about to run, simulation data has to
  # be made available to the job. The status from copy script must
  # be zero or job step cannot run.
```

```

case "sim":
  copy_sim_data >> $LOG
  if ($status == 0 ) then
    set EXIT_CODE = 0
  else
    set EXIT_CODE = 1
  endif
  breaksw
# All other job will require free space in /tmp, make sure
# enough space is available.
default:
  check_tmp >> $LOG
  set EXIT_CODE = $status
  breaksw
endsw

# The job step will run only if EXIT_CODE == 0
exit $EXIT_CODE

```

Coding conventions for epilog programs

The installation defined epilog program is invoked after a job step has completed. The purpose of the epilog program is to perform any required clean up such as unmounting file systems, removing files, and copying results. The exit status of both the prolog program and the job step is set in environment variables.

Syntax:

epilog_program

Where *epilog_program* is the name of the epilog program as defined in the JOB_EPILOG keyword.

No arguments are passed to the program but several environment variables are set. These environment variables are described in “Run-time environment variables” on page 374. In addition, the following environment variables are set for the epilog programs:

LOADL_PROLOG_EXIT_CODE

The exit code from the prolog program. This environment variable is set only if a prolog program is configured to run.

LOADL_USER_PROLOG_EXIT_CODE

The exit code from the user prolog program. This environment variable is set only if a user prolog program is configured to run.

LOADL_JOB_STEP_EXIT_CODE

The exit code from the job step.

Note: To interpret the exit status of the prolog program and the job step, convert the string to an integer and use the macros found in the `sys/wait.h` file.

These macros include:

- WEXITSTATUS: gives you the exit code
- WTERMSIG: gives you the signal that terminated the program
- WIFEXITED: tells you if the program exited
- WIFSIGNALED: tells you if the program was terminated by a signal

The exit codes returned by the WEXITSTATUS macro are the valid codes. However, if you look at the raw numbers in `sys/wait.h`, the exit code may appear to be 256 times the expected return code. The numbers in `sys/wait.h` are the wait3 system calls.

Customizing the configuration file

Sample epilog programs:

- **Sample of an epilog program for korn shell**

```
#!/bin/ksh
#
# Set up environment
set -a
. /etc/environment
. /.profile
export PATH="$PATH:/loctools/lladmin/bin"
export LOG="/tmp/$LOADL_STEP_OWNER.$LOADL_STEP_ID.epilog"
#
if [ [ -z $LOADL_PROLOG_EXIT_CODE ] ]
then
echo "Prolog did not run" >> $LOG
else
echo "Prolog exit code = $LOADL_PROLOG_EXIT_CODE" >> $LOG
fi
#
if [ [ -z $LOADL_USER_PROLOG_EXIT_CODE ] ]
then
echo "User environment prolog did not run" >> $LOG
else
echo "User environment exit code = $LOADL_USER_PROLOG_EXIT_CODE" >> $LOG
fi
#
if [ [ -z $LOADL_JOB_STEP_EXIT_CODE ] ]
then
echo "Job step did not run" >> $LOG
else
echo "Job step exit code = $LOADL_JOB_STEP_EXIT_CODE" >> $LOG
fi
#
#
# Do clean up based upon job step class
#
case $LOADL_STEP_CLASS in
# A OSL job just ran, unmount the filesystem.
"OSL")
umount_osl_files >> $LOG
;;
# A simulation job just ran, remove input files.
# Copy results if simulation was successful (second argument
# contains exit status from job step).
"sim")
rm_sim_data >> $LOG
if [ $2 = 0 ]
then copy_sim_results >> $LOG
fi
;;
# Clean up /tmp
*)
clean_tmp >> $LOG
;;
esac
```

- **Sample of an epilog program for C shell**

```
#!/bin/csh
#
# Set up environment
source /u/loadl/.login
#
setenv PATH "${PATH}:/loctools/lladmin/bin"
setenv LOG "/tmp/${LOADL_STEP_OWNER}.${LOADL_STEP_ID}.prolog"
#
if ( ${?LOADL_PROLOG_EXIT_CODE} ) then
echo "Prolog exit code = $LOADL_PROLOG_EXIT_CODE" >> $LOG
```

```

else
echo "Prolog did not run" >> $LOG
endif
#
if ( ${?LOADL_USER_PROLOG_EXIT_CODE} ) then
echo "User environment exit code = $LOADL_USER_PROLOG_EXIT_CODE" >> $LOG
else
echo "User environment prolog did not run" >> $LOG
endif
#
if ( ${?LOADL_JOB_STEP_EXIT_CODE} ) then
echo "Job step exit code = $LOADL_JOB_STEP_EXIT_CODE" >> $LOG
else
echo "Job step did not run" >> $LOG
endif
#
# Do clean up based upon job step class
#
switch ($LOADL_STEP_CLASS)
# A OSL job just ran, unmount the filesystem.
case "OSL":
umount_osl_files >> $LOG
breaksw
# A simulation job just ran, remove input files.
# Copy results if simulation was successful (second argument
# contains exit status from job step).
case "sim":
rm_sim_data >> $LOG
if ($argv{2} == 0 ) then
copy_sim_results >> $LOG
endif
breaksw
# Clean up /tmp
default:
clean_tmp >> $LOG
breaksw
endsw

```

Using your own mail program

You can write a program to override the LoadLeveler default mail notification method. You can use this program, for example, to display your own messages to users when a job completes, or to automate tasks such as sending error messages to a network manager.

The syntax for the program is the same as it is for standard UNIX mail programs; the command is called with the following arguments:

- **-s** to indicate a subject.
- A pointer to a string containing the subject.
- A pointer to a string containing a list of mail recipients.

The mail message is taken from standard input.

To enable this program to replace the default mail notification method, use the **MAIL** keyword in the configuration file. For details about specific keyword syntax and use in the configuration file, see Chapter 12, "Configuration file reference," on page 243.

Chapter 5. Defining LoadLeveler resources to administer

After installing LoadLeveler, you may customize it by modifying the **administration** file. The administration file optionally lists and defines the machines in the LoadLeveler cluster and the characteristics of classes, users, and groups.

LoadLeveler does not prevent you from having multiple copies of administration files, but you need to be sure to update all the copies whenever you make a change to one. Having only one administration file prevents any confusion.

Table 19 lists the LoadLeveler resources you may define by modifying the administration file.

Table 19. Roadmap of tasks for modifying the LoadLeveler administration file

To learn about:	Read the following:
Modifying the administration file	“Steps for modifying an administration file”
Defining LoadLeveler resources to administer	<ul style="list-style-type: none">• “Defining machines” on page 78• “Defining adapters” on page 80• “Defining classes” on page 83• “Defining users” on page 91• “Defining groups” on page 92• “Defining clusters” on page 93
Correctly specifying administration file keywords	Chapter 13, “Administration file reference,” on page 301

Steps for modifying an administration file

All LoadLeveler commands, daemons, and processes read the administration and configuration files at start up time. If you change the administration or configuration files after LoadLeveler has already started, any LoadLeveler command or process, such as the **LoadL_starter** process, will read the newer version of the files while the running daemons will continue to use the data from the older version. To ensure that all LoadLeveler commands, daemons, and processes use the same configuration data, run the reconfiguration command on all machines in the cluster each time the administration or configuration files are changed.

Before you begin: You need to:

- Ensure that the installation procedure has completed successfully and that the administration file, **LoadL_admin**, exists in LoadLeveler’s home directory. For additional details about installation, see *TWS LoadLeveler: Installation Guide*.
- Know how to correctly specify keywords in the administration file. For information about administration file keyword syntax and other details, see Chapter 13, “Administration file reference,” on page 301.
- (Optional) Know how to correctly issue the **llextrRPD** command, if you choose to use it (see “llextrRPD - Extract data from an RSCT peer domain” on page 415).

Customizing the administration file

Perform the following steps to modify the administration file, **LoadL_admin**:

1. Identify yourself as a LoadLeveler administrator using the **LOADL_ADMIN** keyword.
2. In the administration file, provide the following stanza types:
 - One machine stanza to define the central manager for the LoadLeveler cluster. You also may create machine stanzas for other machines in the LoadLeveler cluster.
You can use the **llextrPD** command to automatically create machine stanzas.
 - (Optional) An adapter stanza for each type of network adapter that you want LoadLeveler jobs to be able to request.
You can use the **llextrPD** command to automatically create adapter stanzas.
3. (Optional) Specify one or more of the following stanza types:
 - A class stanza for each set of LoadLeveler jobs that have similar characteristics or resource requirements.
 - A user stanza for specific users, if their requirements do not match those characteristics defined in the default user stanza.
 - A group stanza for each set of LoadLeveler users that have similar characteristics or resource requirements.
4. (Optional) You may specify values for additional administration file keywords, which are listed and described in “Administration file keyword descriptions” on page 306.
5. Notify LoadLeveler daemons by issuing the **llctl** command with either the **reconfig** or **recycle** keyword. Otherwise, LoadLeveler will not process the modifications you made to the administration file.

Defining machines

The information in a machine stanza defines the characteristics of that machine. You do not have to specify a machine stanza for every machine in the LoadLeveler cluster, but you must have one machine stanza for the machine that will serve as the central manager.

If you do not specify a machine stanza for a machine in the cluster, the machine and the central manager still communicate and jobs are scheduled on the machine but the machine is assigned the default values specified in the default machine stanza. If there is no default stanza, the machine is assigned default values set by LoadLeveler.

Any machine name used in the stanza must be a name which can be resolved to an IP address. This name is referred to as an interface name because the name can be used for a program to interface with the machine. Generally, interface names match the machine name, but they do not have to.

By default, LoadLeveler will append the DNS domain name to the end of any machine name without a domain name appended before resolving its address. If you specify a machine name without a domain name appended to it and you do not want LoadLeveler to append the DNS domain name to it, specify the name using a trailing period. You may have a need to specify machine names in this way if you are running a cluster with more than one nameserving technique. For example, if you are using a DNS nameserver and running NIS, you may have some machine names which are resolved by NIS which you do not want LoadLeveler to append DNS names to. In situations such as this, you also want to specify **name_server** keyword in your machine stanzas.

Under the following conditions, you must have a machine stanza for the machine in question:

- If you set the **MACHINE_AUTHENTICATE** keyword to **true** in the configuration file, then you must create a machine stanza for each node that LoadLeveler includes in the cluster.
- If the machine's hostname (the name of the machine returned by the UNIX hostname command) does not match an interface name. In this case, you must specify the interface name as the machine stanza name and specify the machine's hostname using the **alias** keyword.
- If the machine's hostname does match an interface name but not the correct interface name.

For information about automatically creating machine stanzas, see “llextrPD - Extract data from an RSCT peer domain” on page 415.

Planning considerations for defining machines

Before customizing the administration file, consider the following:

- **Node availability**

Some workstation owners might agree to accept LoadLeveler jobs only when they are not using the workstation themselves. Using LoadLeveler keywords, these workstations can be configured to be available at designated times only.

- **Common name space**

To run jobs on any machine in the LoadLeveler cluster, a user needs the same uid (the user ID number for a user) and gid (the group ID number for a group) on every machine in the cluster.

For example, if there are two machines in your LoadLeveler cluster, *machine_1* and *machine_2*, user john must have the same user ID and login group ID in the */etc/passwd* file on both machines. If user john has user ID 1234 and login group ID 100 on *machine_1*, then user john must have the same user ID and login group ID in */etc/passwd* on *machine_2*. (LoadLeveler requires a job to run with the same group ID and user ID of the person who submitted the job.)

If you do not have a user ID on one machine, your jobs will not run on that machine. Also, many commands, such as **llq**, will not work correctly if a user does not have a user ID on the central manager machine.

However, there are cases where you may choose to not give a user a login ID on a particular machine. For example, a user does not need an ID on every submit-only machine; the user only needs to be able to submit jobs from at least one such machine. Also, you may choose to restrict a user's access to a Schedd machine that is not a public scheduler; again, the user only needs access to at least one Schedd machine.

- **Resource handling**

Some nodes in the LoadLeveler cluster might have special software installed that users might need to run their jobs successfully. You should configure LoadLeveler to distinguish those nodes from other nodes using, for example, machine features.

Machine stanza format and keyword summary

Machine stanzas take the following format. Default values for keywords appear in bold:

Customizing the administration file

```
label: type = machine
adapter_stanzas = stanza_list
alias = machine_name
central_manager = true | false | alt
cpu_speed_scale = true | false
machine_mode = batch | interactive | general
master_node_exclusive = true | false
max_jobs_scheduled = number
name_server = list
pool_list = pool_numbers
reservation_permitted = true | false
resources = name(count) name(count) ... name(count)
schedd_fenced = true | false
schedd_host = true | false
spacct_exclude_enable = true | false
speed = number
submit_only = true | false
```

Figure 11. Format of a machine stanza

Examples: Machine stanzas

- **Example 1**

In this example, the machine is being defined as the central manager.

```
#
machine_a: type = machine
central_manager = true # central manager runs here
```

- **Example 2**

This example sets up a submit-only node. Note that the **submit-only** keyword in the example is set to **true**, while the **schedd_host** keyword is set to **false**. You must also ensure that you set the **schedd_host** to **true** on at least one other node in the cluster.

```
#
machine_b: type = machine
central_manager = false # not the central manager
schedd_host = false # not a scheduling machine
submit_only = true # submit only machine
alias = machineb # interface name
```

- **Example 3**

In the following example, machine_c is the central manager and has an alias associated with it:

```
#
machine_c: type = machine
central_manager = true # central manager runs here
schedd_host = true # defines a public scheduler
alias = brianne
```

Defining adapters

An adapter stanza identifies network adapters that are available on the machines in the LoadLeveler cluster. If you want LoadLeveler jobs to be able to request specific adapters, you must either specify adapter stanzas or configure dynamic adapters in the administration file.

Note the following when using an adapter stanza:

- An adapter stanza is required for each adapter stanza name you specify on the **adapter_stanzas** keyword of the machine stanza.

- The `adapter_name`, `interface_address` and `interface_name` keywords are required.

For information about creating adapter stanzas, see “llextrRPD - Extract data from an RSCT peer domain” on page 415 for peer domains.

Configuring dynamic adapters

LoadLeveler can dynamically determine the adapters in any operating system instance (OSI) that has RSCT installed. LoadLeveler must be told on an OSI basis if it is to handle dynamic adapter configuration changes for that OSI. The specification of whether to use dynamic or static adapter configuration for an OSI is done through the presence or absence of the machine stanza’s `adapter_stanzas` keyword.

If a machine stanza in the administration file contains an `adapter_stanzas` statement then this is taken as a directive by the LoadLeveler administrator to use only those specified adapters. For this OSI, LoadLeveler will not perform any dynamic adapter configuration or processing. If an adapter change occurs in this OSI then the administrator will have to make the corresponding change in the administration file and then stop and restart or reconfigure the LoadLeveler startd daemon to pick up the adapter changes. If an OSI (machine stanza) in the administration file does not contain the `adapter_stanzas` keyword then this is taken as a directive by the LoadLeveler administrator for LoadLeveler to dynamically configure the adapters for that OSI. For that OSI, LoadLeveler will determine what adapters are present at startup via calls to the RMCAPL. If an adapter change occurs during execution in the OSI then LoadLeveler will automatically detect and handle the change without requiring a restart or reconfiguration.

Configuring InfiniBand adapters

InfiniBand adapters, known as host channel adapters (HCAs) can be multiported. Tasks can use ports of an HCA independently, which allows them to be allocated by the scheduling algorithm independently.

Note: InfiniBand adapters are supported in SUSE Linux Enterprise Server (SLES) 9 on POWER TWS LoadLeveler clusters.

An InfiniBand adapter can have multiple adapter ports. Each port on the InfiniBand adapter can be connected to one network and will be managed by TWS LoadLeveler as a switch adapter. InfiniBand adapter ports derive their resources and usage state from the InfiniBand adapter with which they are associated, but are allocated to jobs separately.

If you want LoadLeveler jobs to be able to request InfiniBand adapters, you must either specify adapter stanzas or configure dynamic adapters in the administration file. The InfiniBand ports are identified to TWS LoadLeveler in the same way other adapters are. Stanzas are specified in the administration file if static adapters are used and the ports are discovered by RSCT if dynamic adapters are used.

The `port_number` administration keyword has been added to support an InfiniBand port. The `port_number` keyword specifies the port number of the InfiniBand adapter port. Only InfiniBand ports are managed and displayed by TWS LoadLeveler; the InfiniBand adapter itself is not. The adapter stanza for InfiniBand support only contains the adapter port information. There is no InfiniBand adapter information in the adapter stanza (see example 2 in “Examples: Adapter stanzas” on page 82).

Customizing the administration file

Notes:

1. TWS LoadLeveler distributes the switch adapter windows of the InfiniBand adapter equally among its ports and the allocation is not adjusted should all of the resources on one port be consumed.
2. The InfiniBand ports determine their usage state and availability from their InfiniBand adapter. If one port is in use exclusively, no other ports on the adapter can be used for any other job.
3. If you have a mixed cluster where some nodes use the InfiniBand adapter and some nodes use the HPS adapter, you have to organize the nodes into pools so that the job is only dispatched to nodes with the same kind of switch adapter.
4. There is no change to the way the InfiniBand adapters are requested on the job command file network statement; that is, InfiniBand adapters are requested the same way as any other adapter would be.
5. Because InfiniBand adapters do not support rCxt blocks, jobs that would otherwise use InfiniBand adapters, but which also request rCxt blocks with the `rcxtblks` keyword on the network statement will remain in the idle state. This behavior is consistent with how other adapters (for example, the HPS) behave in the same situation. You can use the `llstatus -a` command to see rCxt blocks on adapters (see “`llstatus - Query machine status`” on page 477 for more information).

Adapter stanza format and keyword summary

An adapter stanza has the following format:

```
label: type = adapter
adapter_name = name
adapter_type = type
device_driver_name = name
interface_address = IP_address
interface_name = name
logical_id = id
multilink_address = ip_address
multilink_list = adapter_name <, adapter_name>*
network_id = id
network_type = type
port_number = number
switch_node_number = integer
```

Figure 12. Format of an adapter stanza

Examples: Adapter stanzas

• Example 1: Specifying an HPS adapter

In the following example, the adapter stanza called “c121s0n10.ppd.pok.ibm.com” specifies an HPS adapter. Note that c121s0n10.ppd.pok.ibm.com is also specified on the `adapter_stanzas` keyword of the machine stanza for the “yugo” machine.

```
yugo: type=machine
      adapter_stanzas = c121s0n10.ppd.pok.ibm.com
      ...

c121s0n10.ppd.pok.ibm.com: type = adapter
                           adapter_name = sn0
                           network_type = switch
                           interface_address = 192.168.0.10
                           interface_name = c121s0n10.ppd.pok.ibm.com
                           multilink_address = 10.10.10.10
                           logical_id = 2
```

```

adapter_type = Switch_Network_Interface_For_HPS
device_driver_name = sni0
network_id = 1

c121f2rp02.ppd.ibm.com: type = adapter
adapter_name = en0
network_type = ethernet
interface_address = 9.114.66.74
interface_name = c121f2rp02.ppd.ibm.com
device_driver_name = ent0

```

• **Example 2: Specifying an InfiniBand adapter**

In the following example, the **port_number** specifies the port number of the InfiniBand adapter port:

```

192.168.9.58: type = adapter
adapter_name = ib1
network_type = InfiniBand
interface_address = 192.168.9.58
interface_name = 192.168.9.58
logical_id = 23
adapter_type = InfiniBand
device_driver_name = ehca0
network_id = 18338657682652659714
port_number = 2

```

Defining classes

The information in a class stanza defines characteristics for that class. These characteristics can include the quantities of consumable resources that may be used by a class per machine or cluster.

Within a class stanza, you can have optional user substanzas that define policies that apply to a user’s job steps that need to use this class. For more information about user substanzas, see “Defining user substanzas in class stanzas” on page 88. For information about user stanzas, see “Defining users” on page 91.

Using limit keywords

A limit is the amount of a resource that a job step or a process is allowed to use. (A process is a dispatchable unit of work.) A job step may be made up of several processes.

Limits include both a **hard limit** and a **soft limit**. When a hard limit is exceeded, the job is usually terminated. When a soft limit is exceeded, the job is usually given a chance to perform some recovery actions. Limits are enforced either per process or per job step, depending on the type of limit. For parallel jobs steps, which consist of multiple tasks running on multiple machines, limits are enforced on a per task basis.

The class stanza includes the **limit** keywords shown in Table 20, which allow you to control the amount of resources used by a job step or a job process.

Table 20. Types of limit keywords

Limit	How the limit is enforced
ckpt_time_limit	Per job step
core_limit	Per process
cpu_limit	Per process

Customizing the administration file

Table 20. Types of limit keywords (continued)

Limit	How the limit is enforced
<code>data_limit</code>	Per process
<code>default_wall_clock_limit</code>	Per job step
<code>file_limit</code>	Per process
<code>job_cpu_limit</code>	Per job step
<code>rss_limit</code>	Per process
<code>stack_limit</code>	Per process
<code>wall_clock_limit</code>	Per job step

For example, a common limit is the `cpu_limit`, which limits the amount of CPU time a single process can use. If you set `cpu_limit` to five hours and you have a job step that forks five processes, each process can use up to five hours of CPU time, for a total of 25 CPU hours. Another limit that controls the amount of CPU used is `job_cpu_limit`. For a serial job step, if you impose a `job_cpu_limit` of five hours, the entire job step (made up of all five processes) cannot consume more than five CPU hours. For information on using this keyword with parallel jobs, see “`job_cpu_limit` keyword” on page 353.

You can specify limits in either the class stanza of the administration file or in the job command file. The lower of these two limits will be used to run the job even if the system limit for the user is lower. For more information, see:

- “Enforcing limits”
- “Administration file keyword descriptions” on page 306 or “Job command file keyword descriptions” on page 336

Enforcing limits

LoadLeveler depends on the underlying operating system to enforce process limits. Users should verify that a process limit such as `rss_limit` is enforced by the operating system, otherwise setting it in LoadLeveler will have no effect.

Exceeding job step limits: When a hard limit is exceeded LoadLeveler sends a *non-trappable* signal (except in the case of a parallel job) to the process group that LoadLeveler created for the job step. When a soft limit is exceeded, LoadLeveler sends a *trappable* signal to the process group. Any job application that intends to trap a signal sent by LoadLeveler must ensure that all processes in the process group set up the appropriate signal handler.

All processes in the job step normally receive the signal. The exception to this rule is when a child process creates its own process group. That action isolates the child’s process, and its children, from any signals that LoadLeveler sends. Any child process creating its own process group is still known to process tracking. So, if process tracking is enabled, all the child processes are terminated when the main process terminates.

Table 21 summarizes the actions that the `LoadL_starter` daemon takes when a job step limit is exceeded.

Table 21. Enforcing job step limits

Type of Job	When a Soft Limit is Exceeded	When a Hard Limit is Exceeded
Serial	SIGXCPU or SIGKILL issued	SIGKILL issued

Table 21. Enforcing job step limits (continued)

Type of Job	When a Soft Limit is Exceeded	When a Hard Limit is Exceeded
Parallel	SIGXCPU issued to both the user program and to the parallel daemon	SIGTERM issued

On systems that do not support SIGXCPU, LoadLeveler does not distinguish between hard and soft limits. When a soft limit is reached on these platforms, LoadLeveler issues a SIGKILL.

Enforcing per process limits: For per process limits, what happens when your job reaches and exceeds either the soft limit or the hard limit depends on the operating system you are using.

When a job forks a process that exceeds a per process limit, such as the CPU limit, the operating system (not LoadLeveler) terminates the process by issuing a SIGXCPU. As a result, you will not see an entry in the LoadLeveler logs indicating that the process exceeded the limit. The job will complete with a 0 return code. LoadLeveler can only report the status of any processes it has started.

If you need more specific information, refer to your operating system documentation.

How LoadLeveler uses hard limits: See Table 22 for more information on specifying limits.

Table 22. Setting limits

If the hard limit is:	Then LoadLeveler does the following:
Set in both the class stanza and the job command file	Smaller of the two limits is taken into consideration. If the smaller limit is the job limit, the job limit is then compared with the user limit set on the machine that runs the job. The smaller of these two values is used. If the limit used is the class limit, the class limit is used without being compared to the machine limit.
Not set in either the class stanza or the job command file	User per process limit set on the machine that runs the job is used.
Set in the job command file and is less than its respective job soft limit	The job is not submitted.
Set in the class stanza and is less than its respective class stanza soft limit	Soft limit is adjusted downward to equal the hard limit.
Specified in the job command file	Hard limit must be greater than or equal to the specified soft limit and less than or equal to the limit set by the administrator in the class stanza of the administration file. Note: If the per process limit is not defined in the administration file and the hard limit defined by the user in the job command file is greater than the limit on the executing machine, then the hard limit is set to the machine limit.

Allowing users to use a class

In a class stanza, you may define a list of users or a list of groups to identify those who may use the class. To do so, use the **include_users** or **include_groups** keyword, respectively, or you may use both keywords. If you specify both keywords, a particular user must satisfy both the **include_users** and the **include_groups** restrictions for the class. This requirement means that a particular user must be defined not only in a User stanza in the administration file, but also in one of the following ways:

- The user's name must appear in the **include_users** keyword in a Group stanza whose name corresponds to a name in the **include_groups** keyword of the Class stanza.
- The user's name must appear in the **include_groups** keyword of the Class stanza. For information about specifying a user name in a group list, see the **include_groups** keyword description in "Administration file keyword descriptions" on page 306.

Class stanza format and keyword summary

Class stanzas are optional. Class stanzas take the following format. Default values for keywords appear in bold.

```
label: type = class
admin= list
ckpt_dir = directory
ckpt_time_limit = hardlimit,softlimit
class_comment = "string"
core_limit = hardlimit,softlimit
cpu_limit = hardlimit,softlimit
data_limit = hardlimit,softlimit
default_resources = name(count) name(count)...name(count)
env_copy = all | master
exclude_groups = list
exclude_users = list
file_limit = hardlimit,softlimit
include_groups = list
include_users = list
job_cpu_limit = hardlimit,softlimit
master_node_requirement = true | false
max_node = number
max_processors = number
max_protocol_instances = number
max_top_dogs = number
max_total_tasks = number
maxjobs = number
nice = value
priority = number
rss_limit = hardlimit,softlimit
smt = yes | no
stack_limit = hardlimit,softlimit
total_tasks = number
wall_clock_limit = hardlimit,softlimit
default_wall_clock_limit = hardlimit,softlimit
```

Figure 13. Format of a class stanza

Examples: class stanzas

- **Example 1: Creating a class that excludes certain users**

```
class_a: type=class           # class that excludes users
priority=10                  # ClassSysprio
exclude_users=green judy    # Excluded users
```

- **Example 2: Creating a class for small-size jobs**

```

small: type=class                # class for small jobs
priority=80                      # ClassSysprio (max=100)
cpu_limit=00:02:00              # 2 minute limit
data_limit=30mb                 # max 30 MB data segment
default_resources=ConsumableVirtualMemory(10mb) # resources consumed by each
ConsumableCpus(1) resA(3) floatinglicenseX(1) # task of a small job step if
                                                # resources are not explicitly
                                                # specified in the job command file

ckpt_time_limit=3:00,2:00      # 3 minute hardlimit,
                                # 2 minute softlimit
                                # max 10 MB core file
core_limit=10mb                # max file size 50 MB
file_limit=50mb                # max stack size 10 MB
stack_limit=10mb               # max resident set size 35 MB
rss_limit=35mb                 # authorized users
include_users = bob sally

```

- **Example 3: Creating a class for medium-size jobs**

```

medium: type=class              # class for medium jobs
priority=70                     # ClassSysprio
cpu_limit=00:10:00             # 10 minute run time limit
data_limit=80mb,60mb          # max 80 MB data segment
                                # soft limit 60 MB data segment

ckpt_time_limit=5:00,4:30      # 5 minute hardlimit,
                                # 4 minute 30 second softlimit to checkpoint

core_limit=30mb                # max 30 MB core file
file_limit=80mb                # max file size 80 MB
stack_limit=30mb               # max stack size 30 MB
rss_limit=100mb                # max resident set size 100 MB
job_cpu_limit=1800,1200        # hard limit is 30 minutes,
                                # soft limit is 20 minutes

```

- **Example 4: Creating a class for large-size jobs**

```

large: type=class               # class for large jobs
priority=60                     # ClassSysprio
cpu_limit=00:10:00             # 10 minute run time limit
data_limit=120mb               # max 120 MB data segment
default_resources=ConsumableVirtualMemory(40mb) # resources consumed
ConsumableCpus(2) resA(8) floatinglicenseX(1) resB(1) # by each task of
                                                # a large job step if resources are not
                                                # explicitly specified in the job command file

ckpt_time_limit=7:00,5:00      # 7 minute hardlimit,
                                # 5 minute softlimit to checkpoint

core_limit=30mb                # max 30 MB core file
file_limit=120mb               # max file size 120 MB
stack_limit=unlimited            # unlimited stack size
rss_limit=150mb                # max resident set size 150 MB
job_cpu_limit = 3600,2700        # hard limit 60 minutes
                                # soft limit 45 minutes

wall_clock_limit=12:00:00,11:59:55 # hard limit is 12 hours

```

- **Example 5: Creating a class for master node machines**

```

sp-6hr-sp: type=class           # class for master node machines
priority=50                     # ClassSysprio (max=100)
ckpt_time_limit=25:00,20:00     # 25 minute hardlimit,
                                # 20 minute softlimit to checkpoint

cpu_limit = 06:00:00            # 6 hour limit
job_cpu_limit = 06:00:00        # hard limit is 6 hours
core_limit = 1mb                # max 1MB core file
master_node_requirement = true # master node definition

```

- **Example 6: Creating a class for MPICH-GM jobs**

```

MPICHGM: type=class             # class for MPICH-GM jobs
default_resources = gmports(1) # one gmports resource is consumed by each
                                # task, if resources are not explicitly
                                # specified in the job command file

```

Defining user substanzas in class stanzas

In a class stanza, you might define user substanzas using the same syntax as you would for any stanza in the LoadLeveler administration file. A user substanza within a class stanza defines policies that apply to job steps submitted by that user and belonging to that class. User substanzas are optional and are independent of user stanzas (for information about user stanzas, see “Defining users” on page 91).

Class stanzas that contain user substanzas have the following format:

```
label: {  
    type = class  
    label: {  
        type = user  
        maxidle = number  
        maxjobs = number  
        maxqueued = number  
        max_total_tasks = number  
    }  
}
```

Figure 14. Format of a user substanza

When defining substanzas within other stanzas, you must use opening and closing braces ({ and }) to mark the beginning and the end of the stanza and substanza. The only keywords that are supported in a user substanza are **type** (required), **maxidle**, **maxjobs**, **maxqueued**, and **max_total_tasks**. For detailed descriptions of these keywords, see “Administration file keyword descriptions” on page 306.

Examples: substanzas

In the following example, the default machine and class stanzas do not require braces, but the parallel class stanza does require them. Without braces to open and close the parallel stanza, it would not be clear that the default user and **dept_head** user stanza belong to the parallel class:

```
default:  
    type = machine  
    central_manager = false  
    schedd_host = true  
  
default:  
    type = class  
    wall_clock_limit = 60:00,30:00  
  
parallel: {  
    type = class  
  
    # Allow at most 50 running jobs for class parallel  
    maxjobs = 50  
  
    # Allow at most 10 running jobs for any single  
    # user of class parallel  
    default: {  
        type = user  
        maxjobs = 10  
    }  
  
    # Allow user dept_head to run as many as 20 jobs  
    # of class parallel  
    dept_head: {type = user  
        maxjobs = 20
```

```

    }
}

dept_head: type = user
    maxjobs = 30

```

When user substanzas are used in class stanzas, a default user substanza can be defined. Each class stanza can have its own default user substanza, and even the default class stanza can have a default user substanza. In this example, the default user substanza in the default class indicates that for any combination of class and user, the limits **maxidle=20** and **maxqueued=30** apply, and that **maxjobs** and **max_total_tasks** are unlimited. Some of these values are overridden in the physics class stanza. Here is an example of how class stanzas can be configured:

```

default: {
    type = class
    default: {
        type = user
        maxidle = 20
        maxqueued = 30
        maxjobs = -1
        max_total_tasks = -1
    }
}

physics: {
    type = class
    default: {
        type = user
        maxjobs = 10
        max_total_tasks = 128
    }
    john: {
        type = user
        maxidle = 10
        maxjobs = 14
    }
    jane: {
        type = user
        max_total_tasks = 192
    }
}

```

In the following example, the physics stanza shows which values are inherited from which stanzas:

```

physics: {
    type = class
    default: {
        type = user
        # inherited from default class, default user
        # maxidle = 20

        # inherited from default class, default user
        # maxqueued = 30

        # overrides value of -1 in default class, default user
        maxjobs = 10

        # overrides value of -1 in default class, default user
        max_total_tasks = 128
    }
    john: {
        type = user
        # overrides value of 10 in default user
        maxidle = 10
    }
}

```

Customizing the administration file

```
        # inherited from default user, which was inherited
        # from default class, default user
        # maxqueued = 30

        # overrides value of 10 in default user
        maxjobs = 14

        # inherited from default user
        # max_total_tasks = 128
    }

jane: {
    type = user
    # inherited from default user, which was inherited
    # from default class, default user
    # maxidle = 20

    # inherited from default user, which was inherited
    # from default class, default user
    # maxqueued = 30

    # inherited from default user
    # maxjobs = 10

    # overrides value of 128 in default user
    max_total_tasks = 192
}
}
```

Any user other than john and jane who submits jobs of class physics is subject to the constraints in the default user stanza in the physics class stanza. Should john or jane submit jobs of any class other than physics, they are subject to the constraints in the default user stanza in the default class stanza.

In addition to specifying a default user stanza within the default class stanza, an administrator can specify other user stanzas in the default class stanza. It is important to note that all class stanzas will inherit all user stanzas from the default class stanza.

Note: An important rule to understand is that a user stanza within a class stanza will inherit its values from the user stanza in the default class stanza first, if a stanza for that user is present. The next location a user stanza inherits values from is the default user stanza within the same class stanza.

When no default stanzas or stanzas are provided, the LoadLeveler default for all four keywords is -1 or unlimited.

If a user stanza is provided for a user on the class **exclude_users** list, **exclude_users** takes precedence and the user stanza will be effectively ignored because that user cannot use the class at all. On the other hand, when **include_users** is used in a class, the presence of a user stanza implies that the user is permitted to use the class (it is as if the user were present on the **include_users** list).

Defining users

The information specified in a user stanza defines the characteristics of that user. You can have one user stanza for each user but this is not necessary. If an individual user does not have their own user stanza, that user uses the defaults defined in the default user stanza.

User stanza format and keyword summary

User stanzas take the following format:

```
label: type = user
account = list
default_class = list
default_group = group name
default_interactive_class = class name
env_copy = all | master
fair_shares = number
max_node = number
max_processors = number
max_reservation_duration = number
max_reservations = number
max_total_tasks = number
maxidle = number
maxjobs = number
maxqueued = number
priority = number
total_tasks = number
```

Figure 15. Format of a user stanza

For more information about the keywords listed in the user stanza format, see Chapter 13, “Administration file reference,” on page 301.

Examples: User stanzas

- Example 1

In this example, user fred is being provided with a user stanza. User fred's jobs will have a user priority of 100. If user fred does not specify a job class in the job command file, the default job class `class_a` will be used. In addition, he can have a maximum of 15 jobs running at the same time.

```
# Define user stanzas
fred: type = user
priority = 100
default_class = class_a
maxjobs = 15
```

- Example 2

This example explains how a default interactive class for a parallel job is set by presenting a series of user stanzas and class stanzas. This example assumes that users do not specify the `LOADL_INTERACTIVE_CLASS` environment variable.

```
default: type = user
        default_interactive_class = red
        default_class = blue

carol:  type = user
        default_class = single double
        default_interactive_class = ijobs

steve:  type = user
        default_class = single double

ijobs:  type = class
```

Customizing the administration file

```
wall_clock_limit = 08:00:00

red:    type = class
        wall_clock_limit = 30:00
```

If the user Carol submits an interactive job, the job is assigned to the default interactive class called **ijobs**. The job is assigned a wall clock limit of 8 hours. If the user Steve submits an interactive job, the job is assigned to the **red** class from the default user stanza. The job is assigned a wall clock limit of 30 minutes.

- **Example 3**

In this example, Jane's jobs have a user priority of 50, and if she does not specify a job class in her job command file the default job class **small_jobs** is used. This user stanza does not specify the maximum number of jobs that Jane can run at the same time so this value defaults to the value defined in the default stanza. Also, suppose Jane is a member of the primary UNIX group "staff." Jobs submitted by Jane will use the default LoadLeveler group "staff." Lastly, Jane can use three different account numbers.

```
# Define user stanzas
jane: type = user
      priority = 50
      default_class = small_jobs
      default_group = Unix_Group
      account = dept10 user3 user4
```

Defining groups

LoadLeveler groups are another way of granting control to the system administrator. Although a LoadLeveler group is independent from a UNIX group, you can configure a LoadLeveler group to have the same users as a UNIX group by using the **include_users** keyword.

Group stanza format and keyword summary

The information specified in a group stanza defines the characteristics of that group. Group stanzas are optional and take the following format:

```
label: type = group
admin = list
env_copy = all | master
fair_shares = number
exclude_users = list
include_users = list
max_node = number
max_processors = number
max_reservation_duration = number
max_reservations = number
max_total_tasks = number
maxidle = number
maxjobs = number
maxqueued = number
priority = number
total_tasks = number
```

Figure 16. Format of a group stanza

For more information about the keywords listed in the group stanza format, see Chapter 13, "Administration file reference," on page 301.

Examples: Group stanzas

- **Example 1**

In this example, the group name is **department_a**. The jobs issued by users belonging to this group will have a priority of 80. There are three members in this group.

```
# Define group stanzas
department_a: type = group
priority = 80
include_users = susann holly fran
```

- **Example 2**

In this example, the group called **great_lakes** has five members and these user's jobs have a priority of 100:

```
# Define group stanzas
great_lakes: type = group
priority = 100
include_users = huron ontario michigan erie superior
```

Defining clusters

The cluster stanza defines the LoadLeveler multicluster environment. Any cluster that wants to participate in the multicluster must have cluster stanzas defined for all clusters with which the local cluster interacts. If you have a cluster stanza defined, LoadLeveler is configured to be in the multicluster environment.

Cluster stanza format and keyword summary

Cluster stanzas are optional. Cluster stanzas take the following format. Default values for keywords appear in bold>.

The cluster stanza label must define a unique cluster name within the multicluster environment.

```
label: type = cluster
exclude_classes = class_name[(cluster_name)] ...
exclude_groups = group_name[(cluster_name)] ...
exclude_users = user_name[(cluster_name)] ...
inbound_hosts = hostname[(cluster_name)] ...
inbound_schedd_port = port_number
include_classes = class_name[(cluster_name)] ...
include_groups = group_name[(cluster_name)] ...
include_users = user_name[(clustername)] ...
local = true | false
multicluster_security = SSL
outbound_hosts = hostname[(cluster_name)] ...
secure_schedd_port = port_number
ssl_cipher_list = cipher_list
```

Figure 17. Format of a cluster stanza

Examples: Cluster stanzas

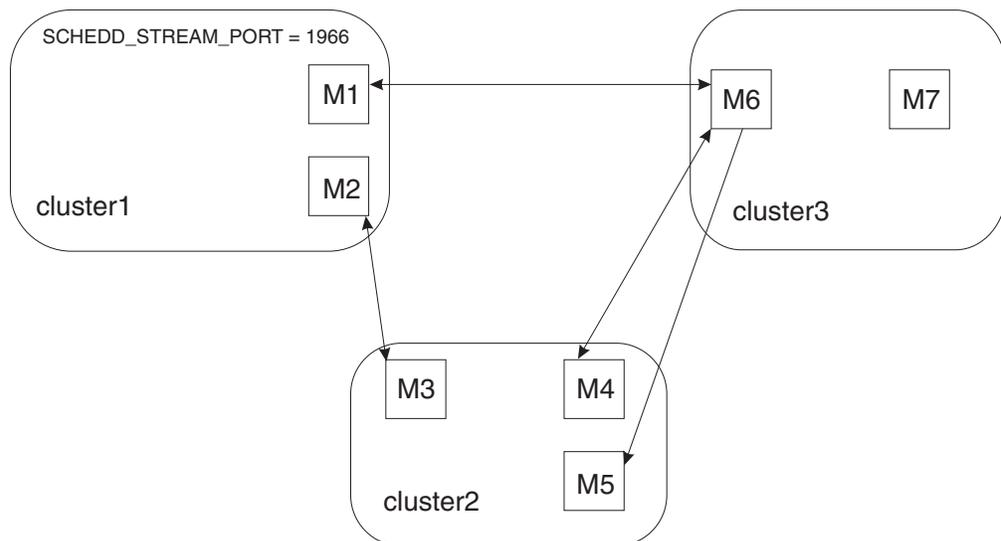


Figure 18. Multicluster Example

Figure 18 shows a simple multicluster with three clusters defined as members. Cluster1 has defined an alternate port number for the Schedds running in its cluster by setting the **SCHEDD_STREAM_PORT = 1966**. All of the other clusters need to define what port to use when connecting to the inbound Schedds of cluster1 by specifying the **inbound_schedd_port = 1966** keyword in the cluster1 stanza. Cluster2 has a single machine connected to cluster1 and 2 machines connected to cluster3. Cluster3 has a single machine connected to both cluster2 and cluster1. Each cluster would set the **local** keyword to **true** for their cluster stanza in the cluster's administration file.

Multicluster with 3 clusters defined as members

```
cluster1: type=cluster
  outbound_hosts = M2(cluster2) M1(cluster3)
  inbound_hosts = M2(cluster2) M1(cluster3)
  inbound_schedd_port = 1966

cluster2: type=cluster
  outbound_hosts = M3(cluster1) M4(cluster3)
  inbound_hosts = M3(cluster1) M4(cluster3) M5(cluster3)

cluster3: type=cluster
  outbound_hosts = M6
  inbound_hosts = M6
```

Chapter 6. Performing additional administrator tasks

Table 23 lists additional ways to modify the LoadLeveler environment that either require an administrator to customize both the configuration and administration files, or require the use of the LoadLeveler commands or APIs.

Table 23. Roadmap of additional administrator tasks

To learn about:	Read the following:
Setting up the environment for parallel jobs	“Setting up the environment for parallel jobs” on page 96
Configuring and using alternative schedulers	<ul style="list-style-type: none"> • “Using the BACKFILL scheduler” on page 101 • “Using an external scheduler” on page 105 • “Example: Changing scheduler types” on page 115
Using additional features available with an alternative scheduler	<ul style="list-style-type: none"> • “Preempting and resuming jobs” on page 116 • “Configuring LoadLeveler to support reservations” on page 121
Working with AIX’s workload balancing component	“Steps for integrating LoadLeveler with AIX Workload Manager” on page 127
Enabling LoadLeveler’s checkpoint/restart function	“LoadLeveler support for checkpointing jobs” on page 129
Enabling LoadLeveler’s affinity support	<ul style="list-style-type: none"> • LoadLeveler scheduling affinity for AIX (see “LoadLeveler scheduling affinity support” on page 136) • LoadLeveler CPU affinity for Linux (see “Linux CPU affinity support” on page 137)
Enabling LoadLeveler’s multicluster support	<ul style="list-style-type: none"> • “LoadLeveler multicluster support” on page 139 • “Configuring a LoadLeveler multicluster” on page 140
Enabling LoadLeveler’s Blue Gene support	<ul style="list-style-type: none"> • “LoadLeveler Blue Gene support” on page 143 • “Configuring LoadLeveler Blue Gene support” on page 145
Enabling LoadLeveler’s fair share scheduling support	<ul style="list-style-type: none"> • “Fair share scheduling overview” on page 25 • “Using fair share scheduling” on page 148
Moving job records from a down Schedd to another Schedd within the local cluster	<ul style="list-style-type: none"> • “Procedure for recovering a job spool” on page 154 • “llmovespool - Move job records” on page 442
Correctly specifying configuration and administration file keywords	<ul style="list-style-type: none"> • Chapter 12, “Configuration file reference,” on page 243 • Chapter 13, “Administration file reference,” on page 301
Managing LoadLeveler operations	<ul style="list-style-type: none"> • Querying status <ul style="list-style-type: none"> • “llclass - Query class information” on page 403 • “llq - Query job status” on page 449 • “llqres - Query a reservation” on page 468 • “llstatus - Query machine status” on page 477

Setting up the environment for parallel jobs

Table 23. Roadmap of additional administrator tasks (continued)

To learn about:	Read the following:
<ul style="list-style-type: none">• Changing attributes of submitted jobs	<ul style="list-style-type: none">• “llfavorjob - Reorder system queue by job” on page 419• “llfavoruser - Reorder system queue by user” on page 421• “llmodify - Change attributes of a submitted job step” on page 435• “llprio - Change the user priority of submitted job steps” on page 447
<ul style="list-style-type: none">• Changing the state of submitted jobs	<ul style="list-style-type: none">• “llcancel - Cancel a submitted job” on page 392• “llhold - Hold or release a submitted job” on page 426

Setting up the environment for parallel jobs

This topic describes the following administration tasks that apply to parallel jobs:

- Scheduling support
- Reducing job launch overhead
- Submitting interactive POE jobs
- Setting up a class
- Setting up a parallel master node
- Configuring MPICH jobs
- Configuring MVAPICH jobs
- Configuring MPICH-GM jobs

For information on submitting parallel jobs, see “Working with parallel jobs” on page 178.

Scheduling considerations for parallel jobs

For parallel jobs, LoadLeveler supports BACKFILL scheduling for efficient use of system resources. These schedulers run both serial and parallel jobs, but they are meant primarily for installations running parallel jobs.

BACKFILL scheduling also supports:

- Multiple tasks per node
- Multiple user space tasks per adapter
- Preemption

Specify the LoadLeveler scheduler using the `SCHEDULER_TYPE` keyword. For more information on this keyword and supported scheduler types, see “Choosing a scheduler” on page 42.

Steps for reducing job launch overhead for parallel jobs

Administrators may define a number of LoadLeveler starter processes to be ready and waiting to handle job requests. Having this pool of ready processes reduces the amount of time LoadLeveler needs to prepare jobs to run. You also may control how environment variables are copied for a job. Reducing the number of environment variables that LoadLeveler has to copy reduces the amount of time LoadLeveler needs to prepare jobs to run.

Before you begin: You need to know:

- How many jobs might be starting at the same time. This estimate determines how many starter processes to have LoadLeveler start in advance, to be ready and waiting for job requests.
- The type of parallel jobs that typically are used. If IBM Parallel Environment (PE) is used for parallel jobs, PE copies the user's environment to all executing nodes. In this case, you may configure LoadLeveler to avoid redundantly copying the same environment variables.
- How to correctly specify configuration keywords. For details about specific keyword syntax and use:
 - In the administration file, see Chapter 13, "Administration file reference," on page 301.
 - In the configuration file, see Chapter 12, "Configuration file reference," on page 243.

Perform the following steps to configure LoadLeveler to reduce job launch overhead for parallel jobs.

1. In the local or global configuration file, specify the number of starter processes for LoadLeveler to automatically start before job requests are submitted. Use the `PRESTARTED_STARTERS` keyword to set this value.
Tip: The default value of 1 should be sufficient for most installations.
2. If typical parallel jobs use a facility such as Parallel Environment, which copies user environment variables to all executing nodes, set the `env_copy` keyword in the class, user, or group stanzas to specify that LoadLeveler only copy user environment variables to the master node by default.

Rules:

- Users also may set this keyword in the job command file. If the `env_copy` keyword is set in the job command file, that setting overrides any setting in the administration file. For more information, see "Step for controlling whether LoadLeveler copies environment variables to all executing nodes" on page 179.
 - If the `env_copy` keyword is set in more than one stanza in the administration file, LoadLeveler determines the setting to use by examining all values set in the applicable stanzas. See the table in the `env_copy` administration file keyword to determine what value LoadLeveler will use.
3. Notify LoadLeveler daemons by issuing the `llctl` command with either the `reconfig` or `recycle` keyword. Otherwise, LoadLeveler will not process the modifications you made to the configuration and administration files.

When you are done with this procedure, you can use the POE stderr and LoadLeveler logs to trace actions during job launch.

Steps for allowing users to submit interactive POE jobs

Perform the following steps to set up your system so that users can submit interactive POE jobs to LoadLeveler.

1. Make sure that you have installed LoadLeveler and defined LoadLeveler administrators. See "Defining LoadLeveler administrators" on page 41 for information on defining LoadLeveler administrators.
2. If running user space jobs, LoadLeveler must be configured to use switch adapters. A way to do this is to run the `llextrRPD` command to extract node and adapter information from the RSCT peer domain. See "llextrRPD - Extract data from an RSCT peer domain" on page 415 for additional information.

Setting up the environment for parallel jobs

3. In the configuration file, define your scheduler to be the LoadLeveler BACKFILL scheduler by specifying **SCHEDULER_TYPE = BACKFILL**. See “Choosing a scheduler” on page 42 for more information.
4. In the administration file, specify batch, interactive, or general use for nodes. You can use the **machine_mode** keyword in the machine stanza to specify the type of jobs that can run on a node; you must specify either **interactive** or **general** if you are going to run interactive jobs.
5. In the administration file, configure optional functions, including:
 - Setting up pools: you can organize nodes into pools by using the **pool_list** keyword in the machine stanza. See “Defining machines” on page 78 for more information.
 - Enabling SP™ exclusive use accounting: you can specify that the accounting function on an SP system be informed that a job step has exclusive use of a machine by specifying **spacct_exclusive_enable = true** in the machine stanza (as shown in the previous example).
See “Defining machines” on page 78 for more information on these keywords.
6. Consider setting up a class stanza for your interactive POE jobs. See “Setting up a class for parallel jobs” for more information. Define this class to be your default class for interactive jobs by specifying this class name on the **default_interactive_class** keyword. See “Defining users” on page 91 for more information.

Setting up a class for parallel jobs

To define the characteristics of parallel jobs run by your installation you should set up a class stanza in the administration file and define a class (in the **Class** statement in the configuration file) for each task you want to run on a node.

Suppose your installation plans to submit long-running parallel jobs, and you want to define the following characteristics:

- Only certain users can submit these jobs
- Jobs have a 30 hour run time limit
- A job can request a maximum of 60 nodes and 120 total tasks
- Jobs will have a relatively low run priority

The following is a sample class stanza for long-running parallel jobs which takes into account these characteristics:

```
long_parallel: type=class
wall_clock_limit = 1800
include_users = jack queen king ace
priority = 50
total_tasks = 120
max_node = 60
maxjobs = 2
```

Note the following about this class stanza:

- The **wall_clock_limit** keyword sets a wall clock limit of 1800 seconds (30 hours) for jobs in this class
- The **include_users** keyword allows four users to submit jobs in this class
- The **priority** keyword sets a relative priority of 50 for jobs in this class
- The **total_tasks** keyword specifies that a user can request up to 120 total tasks for a job in this class

Setting up the environment for parallel jobs

- The **max_node** keyword specifies that a user can request up to 60 nodes for a job in this class
- The **maxjobs** keyword specifies that a maximum of two jobs in this class can run simultaneously

Suppose users need to submit job command files containing the following statements:

```
node = 30
tasks_per_node = 4
```

In your **LoadL_config** file, you must code the **Class** statement such that at least 30 nodes have four or more long_parallel classes defined. That is, the configuration file for each of these nodes must include the following statement:

```
Class = { "long_parallel" "long_parallel" "long_parallel" "long_parallel" }
```

or

```
Class = long_parallel(4)
```

For more information, see “Defining LoadLeveler machine characteristics” on page 51.

Setting up a parallel master node

LoadLeveler allows you to define a parallel master node that LoadLeveler will use as the first node for a job submitted to a particular class. To set up a parallel master node, code the following keywords in the node’s class and machine stanzas in the administration file:

```
# MACHINE STANZA: (optional)
mach1:    type = machine
master_node_exclusive = true
```

```
# CLASS STANZA: (optional)
pmv3:    type = class
master_node_requirement = true
```

Specifying **master_node_requirement = true** forces all parallel jobs in this class to use—as their first node—a machine with the **master_node_exclusive = true** setting. For more information on these keywords, see “Defining machines” on page 78 and “Defining classes” on page 83.

Configuring LoadLeveler to support MPICH jobs

The MPICH package can be configured so that LoadLeveler will be used to spawn all tasks in a MPICH application. Using LoadLeveler to spawn MPICH tasks allows LoadLeveler to accumulate accounting data for the tasks and also allows LoadLeveler to ensure that all tasks are terminated when the job completes.

For LoadLeveler to spawn the tasks of a MPICH job, the MPICH package must be configured to use the LoadLeveler **llspawn.stdio** command when starting tasks. To configure MPICH to use **llspawn.stdio**, set the environment variable **RSHCOMMAND** to the location of the **llspawn.stdio** command and run the configure command for the MPICH package.

On Linux systems, enter the following:

```
# export RSHCOMMAND=/opt/ibm11/LoadL/full/bin/llspawn.stdio
# ./configure
```

Setting up the environment for parallel jobs

Note: This configuration works on MPICH-1.2.7. Additional documentation for MPICH is available from the Argonne National Laboratory web site at <http://www-unix.mcs.anl.gov/mpi/mpich1/>.

Configuring LoadLeveler to support MVAPICH jobs

To run MVAPICH jobs under LoadLeveler control, you must specify the **llspawn** command to replace the default **RSHCOMMAND** value during software configuration.

The compiled MVAPICH implementation code uses the **llspawn** command to start tasks under LoadLeveler control. This allows LoadLeveler to have total control over the remote tasks for accounting and cleanup.

To configure the MVAPICH code to use the **llspawn** command as **RSHCOMMAND**, change the **mpirun_rsh.c** program source code by following these steps before compiling MVAPICH:

1. Replace:

```
Void child_handler(int);
```

with:

```
Void child_handler(int);  
Void term_handler(int);
```

2. For Linux, replace:

```
#define RSH_CMD "/usr/bin/rsh"  
#define RSH_CMD "/usr/bin/ssh"
```

with:

```
#define RSH_CMD "/opt/ibm11/LoadL/full/bin/llspawn"  
#define SSH_CMD "/opt/ibm11/LoadL/full/bin/llpsawn"
```

3. Replace:

```
signal(SIGCHLD, child_handler);
```

with:

```
signal(SIGCHLD, SIG_IGN);  
signal(SIGTERM, term_handler);
```

4. Add the definition for **term_handler** function at the end:

```
Void term_handler(int signal)  
{  
    exit(0);  
}
```

Configuring LoadLeveler to support MPICH-GM jobs

To run MPICH-GM jobs under LoadLeveler control, you need to configure the MPICH-GM implementation you are using by specifying the **llspawn** command as **RSHCOMMAND**. The compiled MPICH-GM implementation code uses the **llspawn** command to start tasks under LoadLeveler control. This allows LoadLeveler to have total control over the remote tasks for accounting and cleanup.

To configure the MPICH-GM code to use the **llspawn** command as **RSHCOMMAND**, change the **mpich.make.gcc** script before compiling the MPICH-GM:

Replace:

```
Setenv RSHCOMMAND /usr/bin/rsh
```

with:

```
Setenv RSHCOMMAND /opt/ibm11/LoadL/full/bin/llspawn
```

LoadLeveler does not manage the GM ports on the Myrinet switch. For LoadLeveler to keep track of the GM ports they must be identified as LoadLeveler consumable resources.

Perform the following steps to use consumable resources to manage GM ports:

1. Pick a name for the GM port resource.

Example: As an example, this procedure assumes the name is **gmports**, but you may use another name.

Tip: Users who submit MPICH-GM jobs need to know the name that you define for the GM port resource.

2. In the LoadLeveler configuration file, specify the GM port resource name on the **SCHEDULE_BY_RESOURCES** keyword.

Example:

```
SCHEDULE_BY_RESOURCES = gmports
```

Tip: If the **SCHEDULE_BY_RESOURCES** keyword already is specified in the configuration file, you can just add the GM port resource name to other values already listed.

3. In the administration file, specify how many GM ports are available on each machine. Use the **resources** keyword to specify the GM port resource name and the number of GM ports.

Example:

```
resources=gmports(n)
```

Tips:

- The **resources** keyword also must appear in the job command file for an MPICH-GM job.

Example:

```
resources=gmports(1)
```

- To determine the value of *n* use either the number specified in the GM documentation or the number of GM ports you have successfully used. Certain system configurations may not support all available GM ports, so you might need to specify a lower value for the **gmports** resource than what is actually available.

4. Issue the **llctl** command with either the **reconfig** or **recycle** keyword. Otherwise, LoadLeveler will not process the modifications you made to the configuration and administration files.

For information about submitting MPICH-GM jobs, see “Running MPICH, MVA PICH, and MPICH-GM jobs” on page 188.

Using the BACKFILL scheduler

The BACKFILL scheduling algorithm in LoadLeveler is designed to maximize the use of resources to achieve the highest system efficiency, while preventing potentially excessive delays in starting jobs with large resource requirements. These large jobs can run because the BACKFILL scheduler does not allow jobs with smaller resource requirements to continuously use up resources before the larger jobs can accumulate enough resources to run. While BACKFILL can be used for both serial and parallel jobs, the potential advantage is greater with parallel jobs.

Using the BACKFILL scheduler

Job steps are arranged in a queue based on their **SYSPRIO** order as they arrive from the Schedd nodes in the cluster. The queue can be periodically reordered depending on the value of the **RECALCULATE_SYSPRIO_INTERVAL** keyword. In each dispatching cycle, as determined by the **NEGOTIATOR_INTERVAL** and **NEGOTIATOR_CYCLE_DELAY** configuration keywords, the BACKFILL algorithm examines these job steps sequentially in an attempt to find available resources to run each job step, then dispatches those steps to run. Once the BACKFILL algorithm encounters a job step for which it cannot immediately find enough resources, that job step becomes known as a "top dog". The BACKFILL algorithm can allocate multiple top dogs in the same dispatch cycle. By using the **MAX_TOP_DOGS** configuration keyword (for more information, see Chapter 12, "Configuration file reference," on page 243), you can define the maximum number of top dogs that the central manager will allocate. For each top dog, the BACKFILL algorithm will attempt to calculate the earliest time at which enough resources will become free to run the corresponding top dog. This is based on the assumption that each currently running job step will run until its hard wall clock limit is reached and that when a job step terminates, the resources which that step has been using will become available. The time at which enough currently running job steps will have terminated, meaning enough resources have become available to run a top dog, is called top dog's future start time. The future start time of each top dog is effectively guaranteed for the remainder of the execution of the BACKFILL algorithm. The resources that each top dog will use at its corresponding start time and for its duration, as specified by its hard wall clock limit, are reserved (not to be confused with the reservation feature available in LoadLeveler).

Note: A job that is bound to a reservation is not considered for top-dog scheduling, so there is no top-dog scheduling performed inside reservations.

In some cases, it may not be possible to calculate the future start time of a job step. Consider, for example, a case where there are 20 nodes in the cluster and a job step requires 24 nodes to run. Even when all nodes in the cluster are idle, it will not be possible for this job step to run. Only the addition of nodes to the cluster would allow the job step to run, and there is no way the BACKFILL algorithm can make any assumptions about when that could take place. In situations like this, the job step is not considered a "top dog", no resources are "reserved", and the BACKFILL algorithm goes on to the next job step in the queue.

The BACKFILL scheduling algorithm classifies job steps into three distinct types: REGULAR, TOP DOG, and BACKFILL:

- The REGULAR job step is a job step for which enough resources are currently available and no top dogs have yet been allocated.
- The TOP DOG job step is a job step for which not enough resources are currently available, but enough resources are available at a future time and one of the following conditions is met:
 - The TOP DOG job step is not expected to run at a time when any other top dog is expected to run.
 - If the TOP DOG is expected to run at a time when some other top dogs are expected to run, then it cannot be using resources reserved by such top dogs.
- The BACKFILL job step is a job step for which enough resources are currently available and one of the following conditions is met:
 - The BACKFILL job step is expected to complete before the future start times of all top dogs, based on the hard wall clock limit of the BACKFILL job step.

- If the BACKFILL job step is not expected to complete before the future start time of at least one top dog, then it cannot be using resources reserved by the top dogs that are expected to start before BACKFILL job step is expected to complete.

The following example shows how the BACKFILL scheduling algorithm might classify the job steps:

```

job1: REGULAR - can start now
job2: REGULAR - can start now
job3: TOP DOG - can't start now, can start at 13:00
job4: BACKFILL - can start now, completes by 12:30
job5: BACKFILL - can start now, completes by 12:00
job6: BACKFILL - can start now, completes by 14:00, /
      doesn't need resources reserved for job3
job7: TOP DOG - can't start now, can start at 12:30 and complete by 13:30, /
      doesn't need resources reserved for job3
job8: BACKFILL - can start now, completes by 12:00
job9: BACKFILL - can start now, completes by 15:00, /
      doesn't need resources reserved for job3 and job7
  
```

Because of the conditions imposed on dispatching BACKFILL job steps, it is possible for lower priority job steps to be scheduled ahead of and delay the dispatching of higher priority job steps. However, the highest priority job steps in the queue, or the "top dogs", are guaranteed not to be delayed any longer once they become the "top dogs".

For coscheduled jobs, remember the following:

- If a coscheduled step becomes a top dog, then resource is reserved for all job steps in the set of coscheduled steps.
- A coscheduled step will be backfilled only when all job steps in the set of coscheduled steps can be backfilled.

Table 24 provides a roadmap of BACKFILL scheduler tasks.

Table 24. Roadmap of BACKFILL scheduler tasks

Subtask	Associated instructions (see . . .)
Configuring the BACKFILL scheduler	<ul style="list-style-type: none"> • "Choosing a scheduler" on page 42 • "Tips for using the BACKFILL scheduler" on page 104 • "Example: BACKFILL scheduling" on page 104
Using additional LoadLeveler features available under the BACKFILL scheduler	<ul style="list-style-type: none"> • "Preempting and resuming jobs" on page 116 • "Configuring LoadLeveler to support reservations" on page 121
Use the BACKFILL scheduler to dispatch and manage jobs	<ul style="list-style-type: none"> • "Scheduler support for parallel jobs" on page 178 • "llclass - Query class information" on page 403 • "llmodify - Change attributes of a submitted job step" on page 435 • "llpreempt - Preempt a submitted job step" on page 444 • "llq - Query job status" on page 449 • "llsubmit - Submit a job" on page 494 • "Data access API" on page 522 • "Error handling API" on page 590 • "ll_modify subroutine" on page 628 • "ll_preempt subroutine" on page 637

Tips for using the BACKFILL scheduler

Note the following when using the BACKFILL scheduler:

- To use this scheduler, either users must set a wall-clock limit in their job command file or the administrator must define a wall-clock limit value for the class to which a job is assigned. Jobs with the **wall_clock_limit** of **unlimited** cannot be used to backfill because they may not finish in time.
- Using wall clock limits that accurately reflect the actual running time of the job steps will result in a more efficient utilization of resources. When a job step's wall clock limit is substantially longer than the amount of time the job step actually needs, it results in two inefficiencies in the BACKFILL algorithm:
 - The future start time of a "top dog" will be calculated to be much later due to the long wall clock limits of the running job steps, leaving a larger window for BACKFILL job steps to run. This causes the "top dog" to start later than it would have if more accurate wall clock limits had been given.
 - A job step is less likely to be backfilled if its wall clock limit is longer because it is more likely to run past the future start time of a "top dog".
- You should use only the default settings for the **START** expression and the other job control functions described in "Managing job status through control expressions" on page 63. If you do not use these default settings, jobs will still run but the scheduler will not be as efficient. For example, the scheduler will not be able to guarantee a time at which the highest priority job will run.
- You should configure any multiprocessor (SMP) nodes such that the number of jobs that can run on a node (determined by the **MAX_STARTERS** keyword) is always less than or equal to the number of processors on the node.
- Due to the characteristics of the BACKFILL algorithm, in some cases this scheduler may not honor the **MACHPRIO** statement. For more information on **MACHPRIO**, see "Setting negotiator characteristics and policies" on page 43.
- When using **PREEMPT_CLASS** rules it is helpful to create a **SYSPRIO** expression which is consistent with the preemption rules. This can be done by using the **ClassSysprio** built-in variable with a multiplier, such as **SYSPRIO: (ClassSysprio * 10000) - QDate**. If classes which appear on the left-hand side of **PREEMPT_CLASS** rules are given a higher priority than those which appear on the right, preemption won't be required as often because the job steps which can preempt will be higher in the queue than the job steps which can be preempted.
- Entering **llq -s** against a top-dog step will display that this step is a top-dog.

Example: BACKFILL scheduling

On a rack with 10 nodes, 8 of the nodes are being used by Job A. Job B has the highest priority in the queue, and requires 10 nodes. Job C has the next highest priority in the queue, and requires only two nodes. Job B has to wait for Job A to finish so that it can use the freed nodes. Because Job A is only using 8 of the 10 nodes, the BACKFILL scheduler can schedule Job C (which only needs the two available nodes) to run as long as it finishes before Job A finishes (and Job B starts). To determine whether or not Job C has time to run, the BACKFILL scheduler uses Job C's **wall_clock_limit** value to determine whether or not it will finish before Job A ends. If Job C has a **wall_clock_limit** of **unlimited**, it may not finish before Job B's start time, and it won't be dispatched.

Using an external scheduler

The LoadLeveler API provides interfaces that allow an external scheduler to manage the assignment of resources to jobs and dispatching those jobs. The primary interfaces for the tasks of an external scheduler are:

- **ll_query** to obtain information about the LoadLeveler cluster, the machines of the cluster, jobs and AIX Workload Manager
- **ll_get_data** to obtain information about specific objects such as jobs, machines and adapters

There are two interfaces for starting a LoadLeveler job: **ll_start_job** and **ll_start_job_ext**. Both support starting serial jobs. **ll_start_job_ext** should be used to start parallel jobs because it allows control over which adapters are used by the communication protocols of each task. Without this control, it is impossible to assure that each task uses the same network for communication over a given protocol.

The steps for dispatching jobs with an external scheduler are:

1. Gather information about the LoadLeveler cluster (**ll_query(CLUSTER)**).
2. Gather information about the machines in the LoadLeveler cluster (**ll_query(MACHINES)**).
3. Gather information about the jobs in the cluster (**ll_query(JOBS)**).
4. Determine the resources that are currently free. (See the note that follows.)
5. Determine which jobs to start. Assign resources to jobs to be started and dispatch (**ll_start_job_ext(LL_start_job_info_ext*)**).
6. Repeat steps 1 through 5.

When an external scheduler is used, the LoadLeveler Negotiator does not keep track of the resources used by jobs started by the external scheduler. There are two ways that an external scheduler can keep track of the free resources available for starting new jobs. The method that should be used depends on whether the external scheduler runs continuously while all scheduling is occurring or is executed to start a finite number of jobs and then terminates:

- If the external scheduler runs continuously, it should query the total resources available in the LoadLeveler system with **ll_query** and **ll_get_data**. Then it can keep track of the resource assigned to jobs it starts while they are running and return the resources to the available pool when the jobs complete.
- If the external scheduler is executed to start a finite number of jobs and then terminates, it must determine the pool of available resources when it first starts. It can do this by first querying the total resources in the LoadLeveler system using **ll_query** and **ll_get_data**. Then it would query the jobs in the system (again using **ll_query**), looking for jobs that are running. For each running job, it would remove the resources used by the job from the available pool. After all the running jobs are processed, the available pool would indicate the amount of free resource for starting new jobs.

To find out more about dispatching jobs with an external scheduler, use the information in Table 25 on page 106.

Using an external scheduler

Table 25. Roadmap of tasks for using an external scheduler

Subtask	Associated instructions (see . . .)
Learn about the LoadLeveler functions that are limited or not available when you use an external scheduler	“Replacing the default LoadLeveler scheduling algorithm with an external scheduler”
Prepare the LoadLeveler environment for using an external scheduler	“Customizing the configuration file to define an external scheduler” on page 107
Use an external scheduler to dispatch jobs	<ul style="list-style-type: none">• “Steps for getting information about the LoadLeveler cluster, its machines, and jobs” on page 108• “Assigning resources and dispatching jobs” on page 112

Replacing the default LoadLeveler scheduling algorithm with an external scheduler

It is important to know how LoadLeveler keywords and commands behave when you replace the default LoadLeveler scheduling algorithm with an external scheduler. LoadLeveler scheduling keywords and commands fall into the following categories:

- Keywords not involved in scheduling decisions are unchanged.
- Keywords kept in the job object or in the machine which are used by the LoadLeveler default scheduler have their values maintained as before and passed to the query API.
- Keywords used only by the LoadLeveler default scheduler have no effect.

Table 26 discusses specific keywords and commands and how they behave when you disable the default LoadLeveler scheduling algorithm.

Table 26. Effect of LoadLeveler keywords under an external scheduler

Keyword type / name	Notes
Job command file keywords	
class	This value is provided by the query APIs. Machines chosen by ll_start_job <i>must</i> have the class of the job available or the request will be rejected.
dependency	Supported as before. Job objects for which dependency cannot be evaluated (because a previous step has not run) are maintained in the NotQueued state, and attempts to start them via ll_start_job will result in an error. If the dependency is met, ll_start_job can start the proc.
hold	ll_start_job cannot start a job that is in Hold status.
min_processors	ll_start_job must specify at least this number of processors.
max_processors	ll_start_job must specify no more than this number of processors.
preferences	Passed to the query API.
requirements	ll_start_job returns an error if the machine(s) specified do not match the requirements of the job. This includes Disk and Virtual Memory requirements.

Table 26. Effect of LoadLeveler keywords under an external scheduler (continued)

Keyword type / name	Notes
startdate	The job remains in the Deferred state until the startdate specified in the job is reached. ll_start_job cannot start a job in the Deferred state.
user_priority	Used in calculating the system priority (as described in “Setting and changing the priority of a job” on page 212). The system priority assigned to the job is available through the query API. No other control of the order in which jobs are run is enforced.
Administration file keywords	
master_node_exclusive	Ignored
master_node_requirement	Ignored
max_jobs_scheduled	Ignored
max_reservations	Ignored
max_reservation_duration	Ignored
max_total_tasks	Ignored
maxidle	Supported
maxjobs	Ignored
maxqueued	Supported
priority	Used to calculate the system priority (where appropriate).
speed	Available through the query API.
Configuration file keywords	
MACHPRIO	Calculated but is not used.
MAX_STARTERS	Calculated, and if starting the job causes this value to be exceeded, ll_start_job returns an error.
SYSPRIO	Calculated and available to the query API.
NEGOTIATOR_PARALLEL_DEFER	Ignored
NEGOTIATOR_PARALLEL_HOLD	Ignored
NEGOTIATOR_RESCAN_QUEUE	Ignored
NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL	Works as before. Set this value to 0 if you do not want the system priorities of job objects recalculated.

Customizing the configuration file to define an external scheduler

To use an external scheduler, one of the tasks you must perform is setting the configuration file keyword **SCHEDULER_TYPE** to the value **API**. This keyword option provides a time-based (rather than an event-based) interface. That is, your application must use the Query API to poll LoadLeveler at specific times for machine and job information.

When you enable a scheduler type of **API**, you must specify **AGGREGATE_ADAPTERS=NO** to make the individual switch adapters available to the external scheduler. This means the external scheduler receives each individual adapter connected to the network, instead of collectively grouping them together. You'll see each adapter listed individually in the **llstatus -l** command

Using an external scheduler

output. When this keyword is set to **YES**, the `llstatus -l` command will show an aggregate adapter which contains information on all switch adapters on the same network. For detailed information about individual switch adapters, issue the `llstatus -a` command.

You also may use the **PREEMPTION_SUPPORT** keyword, which specifies the level of preemption support for a cluster. Preemption allows for a running job step to be suspended so that another job step can run.

Steps for getting information about the LoadLeveler cluster, its machines, and jobs

Perform the following steps to retrieve and use information about the LoadLeveler cluster, machines, jobs and AIX Workload Manager:

1. Create a query object for the kind of information you want.

Example: To query machine information, code the following instruction:

```
LL_element * query_element = ll_query(MACHINES);
```

2. Customize the query to filter the specific information you want. You can filter the list of objects for which you want information. For some queries, you can also filter how much information you want.

Example: The following lines customize the query for just hosts `node01.ibm.com` and `node02.ibm.com` and to return the information contained in the `llstatus -f` command:

```
char * hostlist[] = { "node01.ibm.com", "node02.ibm.com", NULL };  
ll_set_request(query_element, QUERY_HOST, hostlist, STATUS_LINE);
```

3. Once the query has been customized:
 - a. Submit it using `ll_get_objs`, which returns the first object that matches the query.
 - b. Interrogate the returned object using the `ll_get_data` command to retrieve specific attributes. Depending on the information being queried for, the query may be directed to a specific node and a specific daemon on that node.

Example: A **JOBS** query for all data may be directed to the negotiator, Schedd or the history file. If it is directed to the Schedd, you must specify the host of the Schedd you are interested in. The following demonstrates retrieving the name of the first machine returned by the query constructed previously:

```
int machine_count;  
int rc;  
LL_element * element = ll_get_objs(query_element, LL_CM, NULL, &machine_count, &rc)  
char * mname;  
ll_get_data(element, LL_MachineName, &mname);
```

Because there is only one negotiator in a LoadLeveler cluster, the host does not have to be specified. The third parameter is the address of an integer that will receive the count of objects returned and the fourth parameter is the address of an integer that will receive the completion code of the call. If the call fails, `NULL` is returned and the location pointed to by the fourth parameter is set to a reason code. If the call succeeds, the value returned is used as the first parameter to a call to `ll_get_data`. The second parameter to `ll_get_data` is a specification that indicates what attribute of the object is being interrogated. The third parameter to `ll_get_data` is the address of the location into which to store the result. `ll_get_data` returns zero if it is successful and nonzero if an error occurs. It is important that the specification (the second parameter to `ll_get_data`) be valid for the object passed in (the first parameter) and that the

address passed in as the third parameter point to the correct type for the specification. Undefined, potentially dangerous behavior will occur if either of these conditions is not met.

Example: Retrieving specific information about machines

The following example demonstrates printing out the name and adapter list of all machines in the LoadLeveler cluster. The example could be extended to retrieve all of the information available about the machines in the cluster such as memory, disk space, pool list, features, supported classes, and architecture, among other things. A similar process would be used to retrieve information about the cluster overall.

```

int i, w, rc;
int machine_count;
LL_element * query_elem;
LL_element * machine;
LL_element * adapter;
char * machine_name;
char * adapter_name;
int * window_list;
int window_count;

/* First we need to obtain a query element which is used to pass
/* parameters in to the machine query
if ((query_elem = ll_query(MACHINES)) == NULL)
{
    fprintf(stderr,"Unable to obtain query element\n");
    /* without the query object we will not be able to do anything */
    exit(-1);
}

/* Get information relating to machines in the LoadLeveler cluster. */

/* QUERY_ALL: we are querying all machines
/* NULL: since we are querying all machines we do not need to
/* specify a filter to indicate which machines
/* ALL_DATA: we want all the information available about the machine */
rc=ll_set_request(query_elem,QUERY_ALL,NULL,ALL_DATA);
if(rc<0)
{
    /* A real application would map the return code to a message */
    printf("%d returned from ll_set_request\n"rc);
    /* Without customizing the query we cannot proceed */
    exit(rc);
}

/* If successful, ll_get_objs() returns the first object that
/* satisfies the criteria that are set in the query element and
/* the parameters. In this case those criteris are:
/* A machine (from the type of query object)
/* LL_CM: that the negotiator knows about
/* NULL: since there is only one negotiator we don't have to
/* specify which host it is on
/* The number of machines is returned in machine_count and the
/* return code is returned in rc
machine = ll_get_objs(query_elem,LL_CM,NULL,&machine_count,&rc);
if(rc<0)
{
    /* A real application would map the return code to a message */
    printf("%d returned from ll_get_objs\n"rc);

    /* query was not successful -- we cannot proceed but we need to
    /* release the query element
    if(ll_deallocate(query_elem) == -1)
    {
        fprintf(stderr,"Attempt to deallocate invalid query element\n");

```

Using an external scheduler

```
    }
    exit(rc);
}

printf("Number of Machines = %d\n", machine_count);
i = 0;
while(machine!=NULL)
{
    printf("-----\n");
    printf("Machine %d:\n", i);

    int rc = ll_get_data(machine,LL_MachineName,&machine_name);
    if(0==rc)
    {
        printf("Machine name = %s\n",machine_name);
    }
    else
    {
        printf("Error %d returned attempting to get machine name\n",rc);
    }

    printf("Adapters\n");
    ll_get_data(machine,LL_MachineGetFirstAdapter,&adapter);
    while(adapter != NULL)
    {
        rc = ll_get_data(adapter,LL_AdapterName,&adapter_name);
        if(0!=rc)
        {
            printf("Error %d returned attempting to get adapter name\n",rc);
        }
    }
    else
    {
        /* Because the list of windows on an adapter is returned */
        /* as an array of integers, we also need to know how big */
        /* the list is. First we query the window count, */
        /* storing the result in an integer, then we query for */
        /* the list itself, storing the result in a pointer to */
        /* an integer. The window list is allocated for us so */
        /* we need to free it when we are done */

        printf("%s : ",adapter_name);
        ll_get_data(adapter,LL_AdapterTotalWindowCount,&window_count);
        ll_get_data(adapter,LL_AdapterWindowList,&window_list);
        for (w = 0;w<iBuffer;w++)
        {
            printf("%d ",window_list[w]);
        }

        printf("\n");
        free(window_list);
        /* After the first object has been gotten, GetNext returns */
        /* the next until the list is exhausted */
        ll_get_data(machine,LL_MachineGetNextAdapter,&adapter);
    }

    printf("\n");
    i++;
    machine = ll_next_obj(query_elem);
}

/* First we need to release the individual objects that were */
/* obtained by the query */
if(ll_free_objs(query_elem) == -1)
{
    fprintf(stderr,"Attempt to free invalid query element\n");
}
}
```

```

/* Then we need to release the query itself */
if(ll_deallocate(query_elem) == -1)
{
    fprintf(stderr,"Attempt to deallocate invalid query element\n");
}

```

Example: Retrieving information about jobs

The following example demonstrates retrieving information about jobs up to the point of starting a job:

```

int i, rc;
int job_count;
LL_element * query_elem;
LL_element * job;
LL_element * step;
int step_state;

/* First we need to obtain a query element which is used to pass */
/* parameters in to the jobs query */
if ((query_elem = ll_query(JOBS)) == NULL)
{
    fprintf(stderr,"Unable to obtain query element\n");
    /* without the query object we will not be able to do anything */
    exit(-1);
}

/* Get information relating to Jobs in the LoadLeveler cluster. */
printf("Jobs Information =====\n\n");
/* QUERY_ALL: we are querying all jobs */
/* NULL: since we are querying all jobs we do not need to */
/* specify a filter to indicate which jobs */
/* ALL_DATA: we want all the information available about the job */
rc=ll_set_request(query_elem,QUERY_ALL,NULL,ALL_DATA);
if(rc<0)
{
    /* A real application would map the return code to a message */
    printf("%d returned from ll_set_request\n"rc);
    /* Without customizing the query we cannot proceed */
    exit(rc);
}

/* If successful, ll_get_objs() returns the first object that */
/* satisfies the criteria that are set in the query element and */
/* the parameters. In this case those criteris are: */
/* A job (from the type of query object) */
/* LL_CM: that the negotiator knows about */
/* NULL: since there is only one negotiator we don't have to */
/* specify which host it is on */
/* The number of jobs is returned in job_count and the */
/* return code is returned in rc */
job = ll_get_objs(query_elem,LL_CM,NULL,&job_count,&rc);
if(rc<0)
{
    /* A real application would map the return code to a message */
    printf("%d returned from ll_get_objs\n"rc);

    /* query was not successful -- we cannot proceed but we need to */
    /* release the query element */
    if(ll_deallocate(query_elem) == -1)
    {
        fprintf(stderr,"Attempt to deallocate invalid query element\n");
    }
    exit(rc);
}

printf("Number of Jobs = %d\n", job_count);
step = NULL;

```

Using an external scheduler

```
while(job!=NULL)
{
    /* Each job is composed of one or more steps which are started */
    /* individually. We need to check the state of the job's steps */
    ll_get_data(job,LL_JobGetFirstStep,&step);
    while(step!=NULL)
    {
        ll_get_data(step,LL_StepState,&step_state);
        /* We are looking for steps that are in idle state. The */
        /* state is returned as an int so we cast it to */
        /* enum StepState as declared in llapi.h */
        if((enum StepState)step_state == STATE_IDLE)
            break;
    }
    /* If we exit the loop with a valid step, it is the one to start */
    /* otherwise we need to keep looking */
    if(step != NULL)
        break;

    ll_next_obj(query_elem);
}

if(step==NULL)
{
    printf("No step to start\n");
    exit(0);
}
```

Assigning resources and dispatching jobs

In “Example: Retrieving information about jobs” on page 111, we reached the point where a step to start was identified. In a real external scheduler, the decision would be reached after consideration of all the idle jobs and constructing a priority value based on attributes such as class and submit time, all of which are accessible through `ll_get_data`. Next, the list of available machines would be examined to determine whether a set exists with sufficient resources to run the job. This process also involves determining the size of that set of machines using attributes of the step such as number of nodes, instances of each node and tasks per node. The LoadLeveler data query API allows access to that information about each job but the interface for starting the job does not require that the machine and adapter resource match the specifications when the job was submitted. For example, a job could be submitted specifying `node=4` but could be started by an external scheduler on a single node only. Similarly, the job could specify the LAPI protocol with `network.lapi=...` but be started and told to use the MPI protocol. This is not considered an error since it is up to the scheduler to interpret (and enforce, if necessary), the specifications in the job command file.

In allocating adapter resources for a step, it is important that the order of the adapter usages be consistent with the structure of the step. In some environments a task can use multiple instances of adapter windows for a protocol. If the protocol requests striping (`sn_all`), an adapter window (or set of windows if instances are used) is allocated on each available network. If multiple protocols are used by the task (eg. MPI and LAPI), each protocol defines its own set of windows. The array of adapter usages passed in to `ll_start_job_ext` must group the windows for all of the instances on one network for the same protocol together. If the protocol requests striping, that grouping must be immediately followed by the grouping for the next network. If the task uses multiple protocols, the set of adapter usages for the first protocol must be immediately followed by the set for the next protocol. Each task will have exactly the same pattern of adapter usage entries. Corresponding entries across all the tasks represent a communication path and

must be able to communicate with each other. If the usages are for User Space communication, a network table will be loaded for each set of corresponding entries.

All of the job command file keywords for specifying job structure such as **total_tasks**, **tasks_per_node**, **node=min,max** and **blocking** are supported by the **ll_start_job_ext** interface but users should ensure that they understand the LoadLeveler model that is created for each combination when constructing the adapter usage list for **ll_start_job_ext**. Jobs that are submitted with **node=number** and **tasks_per_node** result in more regular LoadLeveler models and are easier to create adapter usage lists for.

In the following example, it is assumed that the step found to be dispatched will run on one machine with two tasks, each task using one switch adapter window for MPI communication. The name of the machine to run on is contained in the variable **use_machine (char*)**, the names of the switch adapters are contained in **use_adapter_1 (char *)** and **use_adapter_2 (char *)** and the adapter windows on those adapters in **use_window_1 (int)** and **use_window_2 (int)**, respectively. Further more, each adapter will be allocated 1M of memory.

If the network adapters that the external scheduler assigns to the job allocate communication buffers in rCxt blocks instead of bytes (the Switch Network Interface for HPS is an example of such a network adapter), the **api_rcxtblocks** field of **adapterUsage** should be used to specify the number of rCxt blocks to assign instead of the **mem** field.

```

LL_start_job_info_ext *start_info;
char * pChar;
LL_element * step;
LL_element * job;
int rc;
char * submit_host;
char * step_id;

start_info = (LL_start_job_info_ext *) (malloc(sizeof(LL_start_job_info_ext)));
if(start_info == NULL)
{
    fprintf(stderr, "Out of memory.\n");
    return;
}

/* Create a NULL terminated list of target machines. Each task
/* must have an entry in this list and the entries for tasks on the
/* same machine must be sequential. For example, if a job is to run
/* on two machines, A and B, and three tasks are to run on each
/* machine, the list would be: AAABBB
/* Any specifications on the job when it was submitted such as
/* nodes, total_tasks or tasks_per_node must be explicitly queried
/* and honored by the external scheduler in order to take effect.
/* They are not automatically enforced by LoadLeveler when an
/* external scheduler is used.
/*
/* In this example, the job will only be run on one machine
/* with only one task so the machine list consists of only 1 machine
/* (plus the terminating NULL entry)
start_info->nodeList = (char **) malloc(2*sizeof(char *));
if (!start_info->nodeList)
{
    fprintf(stderr, "Out of memory.\n");
    return;
}

start_info->nodeList[0] = strdup(use_machine);

```

Using an external scheduler

```
start_info->nodeList[1] = NULL;

/* Retrieve information from the job to populate the start_info structure
/* In the interest of brevity, the success of the ll_get_data() is not tested. In a real application it should be

/* The version number is set from the header that is included when the application using the API is compiled. This allows for checking that the application was compiled with a version of the API that is compatible with the version in the library when the application is run.
start_info->version_num = LL_PROC_VERSION;

/* Get the first step of the job to start
ll_get_data(job,LL_JobGetFirstStep,&step);
if(step==NULL)
{
    printf("No step to start\n");
    return;
}

/* In order to set the submitting host, cluster number and proc number in the start_info structure, we need to parse it out of the step id

/* First get the submitting host and save it
ll_get_data(job,LL_JobSubmitHost,&submit_host);
start_info->StepId.from_host = strdup(submit_host);
free(submit_host);

rc = ll_get_data(step, LL_StepID, &step_id);

/* The step id format is submit_host.jobno.stepno . Because the submit host is a dotted string of indeterminate length, the simplest way to detect where the job number starts is to retrieve the submit host from the job and skip forward its length in the step id.

pChar = step_id+strlen(start_info->StepId.from_host)+1;
/* The next segment is the cluster or job number
pChar = strtok(pChar,".");
start_info->StepId.cluster=atoi(pChar);
/* The last token is the proc or step number
pChar = strtok(NULL,".");
start_info->StepId.proc = atoi(pChar);
free(step_id);

/* For each protocol (eg. MPI or LAPI) on each task, we need to specify which adapter to use, whether a window is being used (subsystem = "US") or not (subsystem="IP"). If a window is used, the window ID and window buffer size must be specified.
/* The adapter usage entries for the protocols of a task must be sequential and the set of entries for tasks on the same node must be sequential. For example the twelve entries for a job where each task uses one window for MPI and one for LAPI with three tasks per node and running on two nodes would be laid out as:
/* 1: MPI window for 1st task running on 1st node
/* 2: LAPI window for 1st task running on 1st node
/* 3: MPI window for 2nd task running on 1st node
/* 4: LAPI window for 2nd task running on 1st node
/* 5: MPI window for 3rd task running on 1st node
/* 6: LAPI window for 3rd task running on 1st node
/* 7: MPI window for 1st task running on 2nd node
/* 8: LAPI window for 1st task running on 2nd node
/* 9: MPI window for 2nd task running on 2nd node
```

```

/* 10: LAPI window for 2nd task running on 2nd node */
/* 11: MPI window for 3rd task running on 2nd node */
/* 12: LAPI window for 3rd task running on 2nd node */
/* An improperly ordered adapter usage list may cause the job not to */
/* be started or, if started, incorrect execution of the job */
/*
/*
/* This example starts the job with two tasks on one machine, using */
/* one switch adapter window on each task. The protocol is forced */
/* to MPI and a fixed window size of 1M is used. An actual external */
/* scheduler application would check the steps requirements and its */
/* adapter requirements of the step with ll_get_data */
/*
start_info->adapterUsageCount = 2;
start_info->adapterUsage =
    (LL_ADAPTER_USAGE *)malloc((start_info->adapterUsageCount)
                               * sizeof(LL_ADAPTER_USAGE));

start_info->adapterUsage[0].dev_name = use_adapter_1;
start_info->adapterUsage[0].protocol = "MPI";
start_info->adapterUsage[0].subsystem = "US";
start_info->adapterUsage[0].wid = use_window_1;
start_info->adapterUsage[0].mem = 1048577;

start_info->adapterUsage[1].dev_name = use_adapter_2;
start_info->adapterUsage[1].protocol = "MPI";
start_info->adapterUsage[1].subsystem = "US";
start_info->adapterUsage[1].wid = use_window_2;
start_info->adapterUsage[1].mem = 1048577;

if ((rc = ll_start_job_ext(start_info)) != API_OK)
{
    printf("Error %d returned attempting to start Job Step %s.%d.%d on %s\n",
           rc,
           start_info->StepId.from_host,
           start_info->StepId.cluster,
           start_info->StepId.proc,
           start_info->nodeList[0]
    );
}
else
{
    printf("ll_start_job_ext() invoked to start job step: "
           "%s.%d.%d on machine: %s.\n\n",
           start_info->StepId.from_host, start_info->StepId.cluster,
           start_info->StepId.proc, start_info->nodeList[0]);
}
free(start_info->nodeList[0]);
free(start_info);

```

Finally, when the step and job element are no longer in use, `ll_free_objs()` and `ll_deallocate()` should be called on the query element.

Example: Changing scheduler types

You can toggle between the default LoadLeveler scheduler and other types of schedulers by using the `SCHEDULER_TYPE` keyword. Changes to `SCHEDULER_TYPE` will not take effect at reconfiguration. The administrator must stop and restart or recycle LoadLeveler when changing `SCHEDULER_TYPE`. A combination of changes to `SCHEDULER_TYPE` and some other keywords may terminate LoadLeveler.

Changing scheduler types

The following example illustrates how you can toggle between the default LoadLeveler scheduler and an external scheduler, such as the Extensible Argonne Scheduling sYstem (EASY), developed by Argonne National Laboratory and available as public domain code.

If you are running the default LoadLeveler scheduler, perform the following steps to switch to an external scheduler:

1. In the configuration file, set **SCHEDULER_TYPE = API**
2. On the central manager machine:
 - Issue **llctl -g stop** and **llctl -g start**, or
 - Issue **llctl -g recycle**

If you are running an external scheduler, this is how you can re-enable the LoadLeveler scheduling algorithm:

1. In the configuration file, set **SCHEDULER_TYPE = LL_DEFAULT**
2. On the central manager machine:
 - Issue **llctl -g stop** and **llctl -g start**, or
 - Issue **llctl -g recycle**

Preempting and resuming jobs

The BACKFILL scheduler allows LoadLeveler jobs to be preempted so that a higher priority job step can run. Administrators may specify not only preemption rules for job classes, but also the method that LoadLeveler uses to preempt jobs. The BACKFILL scheduler supports various methods of preemption.

LoadLeveler for Linux does not support use of the suspend method under the BACKFILL scheduler.

Use Table 27 to find more information about preemption.

Table 27. Roadmap of tasks for using preemption

Subtask	Associated instructions (see . . .)
Learn about types of preemption and what it means for preempted jobs	“Overview of preemption”
Prepare the LoadLeveler environment and jobs for preemption	“Planning to preempt jobs” on page 117
Configure LoadLeveler to use preemption	“Steps for configuring a scheduler to preempt jobs” on page 120

Overview of preemption

LoadLeveler supports two types of preemption:

- System-initiated preemption
 - Automatically enforced by LoadLeveler, except for job steps running under a reservation.
 - Governed by the PREEMPT_CLASS rules defined in the global configuration file.
 - When resources required by an incoming job are in use by other job steps, all or some of those job steps in certain classes may be preempted according to the PREEMPT_CLASS rules.

- An automatically preempted job step will be resumed by LoadLeveler when resources become available and conditions such as `START_CLASS` rules are satisfied.
- An automatically preempted job step cannot be resumed using `llpreempt` command or `ll_preempt` subroutine.
- User-initiated preemption
 - Manually initiated by LoadLeveler administrators using `llpreempt` command or `ll_preempt` subroutine.
 - A manually preempted job step cannot be resumed automatically by LoadLeveler.
 - A manually preempted job step can be resumed using `llpreempt` command or `ll_preempt` subroutine. Issuing this command or subroutine, however, does not guarantee that the job step will successfully be resumed. A manually preempted job step that was resumed through these interfaces competes for resources with system-preempted job steps, and will be resumed only when resources become available.
 - All steps in a set of coscheduled job steps will be preempted if one or more steps in the set are preempted.
 - A coscheduled step will not be resumed until all steps in the set of coscheduled job steps can be resumed.

For the BACKFILL scheduler only, administrators may select which method LoadLeveler uses to preempt and resume jobs. The suspend method is the default behavior, and is the preemption method LoadLeveler uses for any external schedulers that support preemption. For more information about preemption methods, see “Planning to preempt jobs.”

For a preempted job to be resumed after system- or user-initiated preemption occurs through a method other than suspend, the `restart` keyword in the job command file must be set to `yes`. Otherwise, LoadLeveler vacates the job step and removes it from the cluster.

In order to determine the preempt type and preempt method to use when a coscheduled step preempts another step, an order of precedence for preempt types and preempt methods has been defined. All steps in the preempting coscheduled step will be examined and the preempt type and preempt method having the highest precedence will be used. The order of precedence for preempt type will be ALL, ENOUGH. The precedence order for preempt method will be remove, vacate, system hold, user hold, suspend.

When coscheduled steps are running, if one step is preempted as a result of a system initiated preemption, then all coscheduled steps will be preempted. This implies that more resource than necessary might be preempted when one of the steps being preempted is a coscheduled step.

Planning to preempt jobs

Consider the following points when planning to use preemption:

- **Avoiding circular preemption under the BACKFILL scheduler**

BACKFILL scheduling enables job preemption using rules specified with the `PREEMPT_CLASS` keyword. When you are setting up the preemption rules, make sure that you do not create a circular preemption path. Circular preemption causes a job class to preempt itself after applying the preemption rules recursively. For example, the following keyword definitions set up circular preemption rules on `Class_A`:

Preempting and resuming jobs

```
PREEMPT_CLASS[Class_A] = ALL { Class_B }
PREEMPT_CLASS[Class_B] = ALL { Class_C }
PREEMPT_CLASS[Class_C] = ENOUGH { Class_A }
```

Another example of circular preemption involves **allclasses**:

```
PREEMPT_CLASS[Class_A] = ENOUGH {allclasses}
PREEMPT_CLASS[Class_B] = ALL {Class_A}
```

In this instance, **allclasses** means all classes except Class_A, any additional preemption rule preempting Class_A causes circular preemption.

- **Understanding implied START_CLASS values**

Using the "ALL" value in the PREEMPT_CLASS keyword places implied restrictions on when a job can start. For example,

```
PREEMPT_CLASS[Class_A] = ALL {Class_B Class_C}
```

tells LoadLeveler two things:

1. If a new Class_A job is about to run on a node set, then preempt all Class_B and Class_C jobs on those nodes
2. If a Class_A job is running on a node set, then do not start any Class_B or Class_C jobs on those nodes

This PREEMPT_CLASS statement also implies the following START_CLASS expressions:

1. START_CLASS[Class_B] = (Class_A < 1)
2. START_CLASS[Class_C] = (Class_A < 1)

LoadLeveler adds all implied START_CLASS expressions to the START_CLASS expressions specified in the configuration file. This overrides any existing values for START_CLASS.

For example, if the configuration file contains the following statements:

```
PREEMPT_CLASS[Class_A] = ALL {Class_B Class_C}
START_CLASS[Class_B] = (Class_A < 5)
START_CLASS[Class_C] = (Class_C < 3)
```

When LoadLeveler runs through the configuration process, the PREEMPT_CLASS statement on the first line generates the two implied START_CLASS statements. When the implied START_CLASS statements get added in, the user specified START_CLASS statements are overridden and the resulting START_CLASS statements are effectively equivalent to:

```
START_CLASS[Class_B] = (Class_A < 1)
START_CLASS[Class_C] = (Class_C < 3) && (Class_A < 1)
```

Note: LoadLeveler's central manager (CM) uses these effective expressions instead of the original statements specified in the configuration file. The output from **llclass -l** displays the original customer specified START_CLASS expressions.

- **Selecting the preemption method under the BACKFILL scheduler**

Use Table 28 on page 119 and Table 29 on page 119 to determine which preemption you want to use for jobs running under the BACKFILL scheduler. You may define one or more of the following:

- A default preemption method to be used for all job classes, by setting the **DEFAULT_PREEMPT_METHOD** keyword in the configuration file.
- A specific preemption method for one or more classes or job steps, by using an option on:
 - The **PREEMPT_CLASS** statement in the configuration file.
 - The **llpreempt** command, **ll_preempt** subroutine or **ll_preempt_jobs** subroutine.

Notes:

1. LoadLeveler for Linux does not support the suspend method of preemption.
2. Process tracking must be enabled in order to use the suspend method to preempt a job. To configure LoadLeveler for process tracking, see “Tracking job processes” on page 64.
3. For a preempted job to be resumed after system- or user-initiated preemption occurs through a method other than suspend and remove, the **restart** keyword in the job command file must be **set** to **yes**. Otherwise, LoadLeveler vacates the job step and removes it from the cluster.

Table 28. Preemption methods for which LoadLeveler automatically resumes preempted jobs

Preemption method (abbreviation)	LoadLeveler resumes preempted job:		
	At this time	At this location	At this processing point
Suspend (su)	When preempting job completes	On the same nodes	At the point of suspension
Vacate (vc)	When nodes are available	Any nodes that meet job requirements	At the beginning or at the last successful checkpoint

Table 29. Preemption methods for which administrator or user intervention is required

Preemption method (abbreviation)	Required intervention	LoadLeveler resumes preempted job:	
		At this location	At this processing point
Remove (rm)	Administrator or user must resubmit the preempted job	Any nodes that meet job requirements, when they are available	At the beginning or at the last successful checkpoint
System Hold (sh)	Administrator must release the preempted job		
User Hold (uh)	User must release the preempted job		

- **Understanding how LoadLeveler treats resources held by jobs to be preempted**

When a job step is running, it may be holding the following resources:

- Processors
- Scheduling slots
- Real memory
- ConsumableCpus and ConsumableMemory
- Communication switches, if the **PREEMPTION_TYPE** keyword is set to **FULL** in the configuration file.

When LoadLeveler suspends preemptable jobs running under the BACKFILL scheduler, certain resources held by those jobs do not become available for the preempting jobs. These resources include ConsumableVirtualMemory and floating resources. Under the BACKFILL scheduler only, LoadLeveler releases these resources when you select a preemption method other than suspend. For all preemption methods other than suspend, LoadLeveler treats all job-step resources as available when it preempts the job step.

- **Understanding how LoadLeveler processes multiple entries for the same keywords**

If there are multiple entries for the same keyword in either a configuration file or an administration file, the last entry wins. For example, the following statements are all valid specifications for the same keyword **START_CLASS**:

Preempting and resuming jobs

```
START_CLASS [Class_B] = (Class_A < 1)
START_CLASS [Class_B] = (Class_B < 1)
START_CLASS [Class_B] = (Class_C < 1)
```

However, all three statements identify Class_B as the incoming class. LoadLeveler resolves these statements according to the "last one wins" rule. Because of that, the actual value used for the keyword is (Class_C < 1).

Steps for configuring a scheduler to preempt jobs

Before you begin:

- To define rules for starting and preempting jobs, you need to know certain details about the job characteristics and workload at your installation, including:
 - Which jobs require the same resources, or must be run on the same machines, and so on. This knowledge allows you to group specific jobs into a class.
 - Which jobs or classes have higher priority than others. This knowledge allows you to define which job classes can preempt other classes.
- To correctly configure LoadLeveler to preempt jobs, you might need to refer to the following information:
 - "Choosing a scheduler" on page 42.
 - "Planning to preempt jobs" on page 117.
 - Chapter 12, "Configuration file reference," on page 243.
 - Chapter 13, "Administration file reference," on page 301.
 - "llctl - Control LoadLeveler daemons" on page 409.

Perform the following steps to configure a scheduler to preempt jobs:

1. In the configuration file, use the **SCHEDULER_TYPE** keyword to define the type of LoadLeveler or external scheduler you want to use. Of the LoadLeveler schedulers, only the BACKFILL scheduler supports preemption.
Rule: If you select the BACKFILL or API scheduler, you must set the **PREEMPTION_SUPPORT** configuration keyword to either **full** or **no_adapter**.
2. (Optional) In the configuration file, use the **DEFAULT_PREEMPT_METHOD** to define the default method that the BACKFILL scheduler should use for preempting jobs.
Alternatives: You also may set the preemption method through:
 - The **PREEMPT_CLASS** keyword or on the LoadLeveler preemption command or APIs, which override the setting for the **DEFAULT_PREEMPT_METHOD** keyword.
 - The LoadLeveler GUI by selecting **Admin ► Preempt**.
3. For either the BACKFILL or API scheduler, to preempt by the suspend method requires that you set the **PROCESS_TRACKING** configuration keyword to **true**.
4. In the configuration file, use the **PREEMPT_CLASS** and **START_CLASS** to define the preemption and start policies for job classes.
5. In the administration file, use the **max_total_tasks** keyword to define the maximum number of tasks that may be run per user, group, or class.
6. On the central manager machine:
 - Issue **llctl -g stop** and **llctl -g start**, or
 - Issue **llctl -g recycle**

When you are done with this procedure, you can use the **llq** command to determine whether jobs are being preempted and resumed correctly. If not, use the LoadLeveler logs to trace the actions of each daemon involved in preemption to determine the problem.

Configuring LoadLeveler to support reservations

Under the BACKFILL scheduler only, LoadLeveler allows authorized users to make reservations, which specify a time period during which specific node resources are reserved for use by particular users or groups. Normally, jobs wait to be dispatched until the resources they require become available. Through the use of reservations, wait time can be reduced because only jobs bound to the reservation may use the node resources as soon as the reservation period begins.

Use Table 30 to find additional information about reservations.

Table 30. Roadmap of reservation tasks for administrators

Subtask	Associated instructions (see . . .)
Learn how reservations work in the LoadLeveler environment	<ul style="list-style-type: none"> • “Overview of reservations” on page 23 • “Understanding the reservation life cycle” on page 198
Configuring a LoadLeveler cluster to support reservations	<ul style="list-style-type: none"> • “Steps for configuring reservations in a LoadLeveler cluster” • “Examples: Reservation keyword combinations in the administration file” on page 123 • “Collecting accounting data for reservations” on page 126
Working with reservations: <ul style="list-style-type: none"> • Creating reservations • Submitting jobs under a reservation • Managing reservations 	“Working with reservations” on page 197
Correctly coding and using administration and configuration keywords	<ul style="list-style-type: none"> • Chapter 13, “Administration file reference,” on page 301 • Chapter 12, “Configuration file reference,” on page 243

Steps for configuring reservations in a LoadLeveler cluster

Before you begin:

- You need to know that only the BACKFILL scheduler supports the use of reservations. For information about configuring the BACKFILL scheduler, see “Choosing a scheduler” on page 42.
- You need to decide:
 - Which users will be allowed to create reservations.
 - How many reservations users may own, and how long a duration for their reservations will be allowed.
 - Which nodes will be used for reservations.
 - How much setup time is required before the reservation period starts.
 - Whether accounting data for reservations is to be saved.
- For examples of possible reservation keyword combinations, see “Examples: Reservation keyword combinations in the administration file” on page 123.
- For details about specific keyword syntax and use:
 - In the administration file, see Chapter 13, “Administration file reference,” on page 301.
 - In the configuration file, see Chapter 12, “Configuration file reference,” on page 243.

Configuring LoadLeveler for reservations

Perform the following steps to configure reservations:

1. In the administration file, modify the user or group stanzas to authorize users to create reservations. You may grant the ability to create reservations to an individual user, a group of users, or a combination of users and groups. To do so, define the following keywords in the appropriate user or group stanzas:
 - **max_reservations**, to set the maximum number of reservations that a user or group may have.
 - (Optional) **max_reservation_duration**, to set the maximum amount of time for the reservation period.

Tip: To quickly set up and use reservations, use one of the following examples:

- To allow every user to create a reservation, add `max_reservations=1` to the default user stanza. Then every administrator or user may create a reservation, as long as the number of reservations has not reached the limit for a LoadLeveler cluster.
- To allow a specific group of users to make 10 reservations, add `max_reservations=10` to the group stanza for that LoadLeveler group. Then every user in that group may create a reservation, as long as the number of reservations has not reached the limit for that group or for a LoadLeveler cluster.

See the **max_reservations** description in Chapter 13, “Administration file reference,” on page 301 for more information about setting this keyword in the user or group stanza.

2. In the administration file, modify the machine stanza of each machine that may be reserved. To do so, set the **reservation_permitted** keyword to **true**.

Tip: If you want to allow every machine to be reserved, you do not have to set this keyword; by default, any LoadLeveler machine may be reserved. If you want to prevent particular machines from being reserved, however, you must define a machine stanza for that machine and set the **reservation_permitted** keyword to **false**.

3. In the global configuration file, set reservation policy by specifying values for the following keywords:
 - **MAX_RESERVATIONS** to specify the maximum number of reservations per cluster.

Rule: The total number of reservations supported in a LoadLeveler cluster is 10.

- **RESERVATION_CAN_BE_EXCEEDED** to specify whether LoadLeveler will be permitted to schedule job steps bound to a reservation when their expected end times exceed the reservation end time.

The default for this keyword is **TRUE**, which means that LoadLeveler will schedule these bound job steps even when they are expected to continue running beyond the time at which the reservation ends. Whether these job steps run and successfully complete depends on resource availability, which is not guaranteed after the reservation ends. In addition, these job steps become subject to preemption rules after the reservation ends.

Tip: You might want to set this keyword value to **FALSE** to prevent users from binding long-running jobs to run under reservations of short duration.

- **RESERVATION_MIN_ADVANCE_TIME** to define the minimum time between the time at which a reservation is created and the time at which the reservation is to start.

Tip: To reduce the impact to the currently running workload, consider changing the default for this keyword, which allows reservations to begin as soon as they are created. You may, for example, require reservations to be

Configuring LoadLeveler for reservations

made at least one day (1440 minutes) in advance, by specifying `RESERVATION_MIN_ADVANCE_TIME=1440` in the global configuration file.

- **RESERVATION_PRIORITY** to define whether LoadLeveler administrators may reserve nodes on which running jobs are expected to end after the start time for the reservation.

Tip: The default for this keyword is `NONE`, which means that LoadLeveler will not reserve a node on which running jobs are expected to end after the start time for the reservation. If you want to allow LoadLeveler administrators to reserve specific nodes regardless of the expected end times of job steps currently running on the node, set this keyword value to `HIGH`. Note, however, that setting this keyword value to `HIGH` might increase the number of job steps that must be preempted when LoadLeveler sets up the reservation, and many jobs might remain in Preempted state. This also applies to Blue Gene job steps.

This keyword value applies only for LoadLeveler administrators; other reservation owners do not have this capability.

- **RESERVATION_SETUP_TIME** to define the amount of time LoadLeveler uses to prepare for a reservation before it is to start.
4. (Optional) In the global configuration file, set controls for the collection of accounting data for reservations:
 - To turn on accounting for reservations, add the `A_RES` flag to the `ACCT` keyword.
 - To specify a file other than the default history file to contain the data, use the `RESERVATION_HISTORY` keyword.

To learn how to collect accounting data for reservations, see “Collecting accounting data for reservations” on page 126.

5. If LoadLeveler is already started, to process the changes you made in the preceding steps, issue the command `llctl -g reconfig`

Tip: If you have changed the value of only the `RESERVATION_PRIORITY` keyword, issue the command `llctl reconfig` only on the central manager node.

Result: The new keyword values take effect immediately, but they do not change the attributes of existing reservations.

When you are done with this procedure, you may perform additional tasks described in “Working with reservations” on page 197.

Examples: Reservation keyword combinations in the administration file

The following examples demonstrate LoadLeveler behavior when the `max_reservations` and `max_reservation_duration` keywords are set. The examples assume that only the user and group stanzas listed exist in the LoadLeveler administration file.

- **Example 1:** Assume the administration file contains the following stanzas:

```
default: type = user
        maxjobs = 10

group2: type = group
        include_users = rich dave steve

rich: type = user
      default_group = group2
```

This example shows that, by default, no one is allowed to make any reservations. No one, including LoadLeveler administrators, is permitted to make any reservations unless the `max_reservations` keyword is used.

Configuring LoadLeveler for reservations

- **Example 2:** Assume the administration file contains the following stanzas:

```
default: type = user
        maxjobs = 10

group2: type = group
        include_users = rich dave steve

rich: type = user
      default_group = group2
      max_reservations = 5
```

This example shows how permission to make reservations can be granted to a specific user through the user stanza only. Because the **max_reservations** keyword is not used in any group stanza, by default, the group stanzas neither grant permissions nor put any restrictions on reservation permissions. User Rich can make reservations in any group (group2, No_Group, Group_A, and so on), whether or not the group stanzas exist in the LoadLeveler administration file. The total number of reservations user Rich can own at any given time is limited to five.

- **Example 3:** Assume the administration file contains the following stanzas:

```
default: type = user
        maxjobs = 10

group2: type = group
        include_users = rich dave steve
        max_reservations = 5

rich: type = user
      default_group = group2
```

This example shows how permission to make reservations can be granted to a group of users through the group stanza only. Because the **max_reservations** keyword is not used in any user stanza, by default, the user stanzas neither grant nor deny permission to make reservations. All users in group2 (Rich, Dave and Steve) can make reservations, but they must make reservations in group2 because other groups do not grant the permission to make reservations. The total number of reservations the users in group2 can own at any given time is limited to five.

- **Example 4:** Assume the administration file contains the following stanzas:

```
default: type = user
        maxjobs = 10

group2: type = group
        include_users = rich dave steve
        max_reservations = 5

rich: type = user
      default_group = group2
      max_reservations = 0
```

This example shows how permission to make reservations can be granted to a group of users except one specific user. Because the **max_reservations** keyword is set to zero in the user stanza for Rich, he does not have permission to make any reservation, even though all other users in group2 (Dave and Steve) can make reservations.

- **Example 5:** Assume the administration file contains the following stanzas:

```
default: type = group
        max_reservations = 0

default: type = user
        max_reservations = 0
```

Configuring LoadLeveler for reservations

```
group2: type = group
        include_users = rich dave steve
        max_reservations = 5
```

```
rich: type = user
      default_group = group2
      max_reservations = 5
```

```
dave: type = user
      max_reservations = 2
```

This example shows how permission to make reservations can be granted to specific user and group pairs. Because the **max_reservations** keyword is set to zero in both the default user and group stanza, no one has permission to make any reservation unless they are specifically granted permission through both the user and group stanza. In this example:

- User Rich can own at any time up to five reservations in group2 only.
- User Dave can own at any time up to two reservations in group2 only.

The total number of reservations they can own at any given time is limited to five. No other combination of user or group pairs can make any reservations.

- **Example 6:** Assume the administration file contains the following stanzas:

```
default: type = user
        max_reservations = 1
```

This example permits any user to make one reservation in any group, until the number of reservations reaches the maximum number allowed in the LoadLeveler cluster.

- **Example 7:** Assume the administration file contains the following stanzas:

```
default: type = group
        max_reservations = 0
```

```
default: type = user
        max_reservations = 0
```

```
group1: type = group
        max_reservations = 6
        max_reservation_duration = 1440
```

```
carol: type = user
      default_group = group1
      max_reservations = 4
      max_reservation_duration = 720
```

```
dave: type = user
      default_group = group1
      max_reservations = 4
      max_reservation_duration = 2880
```

In this example, two users, Carol and Dave, are members of group1. Neither Carol nor Dave belong to any other group with a group stanza in the LoadLeveler administration file, although they may use any string as the name of a LoadLeveler group and belong to it by default.

Because the **max_reservations** keyword is set to zero in the default group stanza, reservations can be made only in group1, which has an allotment of six reservations. Each reservation can have a maximum duration of 1440 minutes (24 hours).

Considering only the user-stanza attributes for reservations:

- User Carol can make up to four reservations with each having a maximum duration of 720 minutes (12 hours).
- User Dave can make up to four reservations with each having a maximum duration of 2880 minutes (48 hours).

Configuring LoadLeveler for reservations

If there are no reservations in the system and user Carol wants to make four reservations, she may do so. Each reservation can have a maximum duration of no more than 720 minutes. If Carol attempts to make a reservation with a duration greater than 720 minutes, LoadLeveler will not make the reservation because it exceeds the duration allowed for Carol.

Assume that Carol has created four reservations, and user Dave now wants to create four reservations:

- The number of reservations Dave may make is limited by the state of Carol's reservations and the maximum limit on reservations for group1. If the four reservations Carol made are still being set up, or are active, active shared or waiting, LoadLeveler will restrict Dave to making only two reservations at this time.
- Because the value of **max_reservation_duration** for the group is more restrictive than **max_reservation_duration** for user Dave, LoadLeveler enforces the group value, 1440 minutes.

If Dave belonged to another group that still had reservations available, then he could make reservations under that group, assuming the maximum number of reservations for the cluster had not been met. However, in this example, Dave cannot make any further reservations because they are allowed in group1 only.

Collecting accounting data for reservations

LoadLeveler can collect accounting data for reservations, which are set periods of time during which node resources are reserved for the use of particular users or groups. To enable recording of reservation information, specify the following keywords in the configuration file:

- To turn on accounting for reservations, add the **A_RES** flag to the **ACCT** keyword.
- To specify a file other than the default history file to contain the data, use the **RESERVATION_HISTORY** keyword.

See Chapter 12, "Configuration file reference," on page 243 for details about the **ACCT** and **RESERVATION_HISTORY** keywords.

When these keyword values are set and a reservation ends or is canceled, LoadLeveler records the following information:

- The reservation ID
- The time at which the reservation was created
- The user ID of the reservation owner
- The name of the owning group
- Requested and actual start times
- Requested and actual duration
- Actual time at which the reservation ended or was canceled
- Whether the reservation was created with the **SHARED** or **REMOVE_ON_IDLE** options
- A list of users and a list of groups that were authorized to use the reservation
- The number of reserved nodes
- The names of reserved nodes

This reservation information is appended in a single line to the reservation history file for the reservation. The format of reservation history data is:

```
| Reservation ID!Reservation Creation Time!Owner!Owning Group!Start Time! \  
| Actual Start Time!Duration!Actual Duration!Actual End Time!SHARED(yes|no)! \  
| REMOVE_ON_IDLE(yes|no)!Users!Groups!Number of Nodes!Nodes!BG C-nodes! \  
| BG Connection!BG Shape!Number of BG BPs!BG BPs
```

In reservation history data:

Configuring LoadLeveler for reservations

- The unit of measure for start times and end times is the number of seconds since January 1, 1970.
- The unit of time for durations is seconds.

The following is an example of a reservation history file entry:

```
| bg1dd1.rchland.ibm.com.68.r!1150242970!ezhong!group1!1150243200!1150243200! \  
| 300!300!1150243500!no!no!yang!fvt,dev!1!bg1dd1!0!!0!  
| bg1dd1.rchland.ibm.com.54.r!1150143472!ezhong!No_Group!1153612800!0!60!0! \  
| 1150243839!no!no!!0!32!MESH!0x0x0!1!R010(J115)  
| bg1dd1.rchland.ibm.com.70.r!1150244654!ezhong!No_Group!1150244760!1150244760! \  
| 60!60!1150244820!yes!yes!user1,user2!group1,group2!0!512!MESH!1x1x1!1!R010
```

To collect the reservation information stored in the history file, use one of the following:

- The **llacctmrg** command with the **-R** option. For **llacctmrg** command syntax, see “llacctmrg - Collect machine history files” on page 387.
- The LoadLeveler GUI. From the Machines window, use the LoadLeveler GUI by selecting one or more machines, then selecting **Admin ► Collect Reservation Data** and specifying the directory to contain the merged file. If you do not enter a directory, LoadLeveler uses the directory specified in the **GLOBAL_HISTORY** keyword in the configuration file.

```
| To format reservation history data contained in a file, use the sample script  
| llreshist.pl in directory ${RELEASEDIR}/samples/llres/.
```

Steps for integrating LoadLeveler with AIX Workload Manager

Another administrative setup task you must consider is whether you want to enforce resource usage of **ConsumableCPUs** and **ConsumableMemory**. If you want to control these resources, AIX Workload Manager (WLM) can be integrated with LoadLeveler to balance workloads at the machine level.

WLM is not supported in LoadLeveler for Linux.

Workload balancing is done by assigning relative priorities to job processes. These job priorities prevent one job from monopolizing system resources when that resource is under contention.

To integrate LoadLeveler and WLM, perform the following steps:

1. Define **ConsumableCpus**, **ConsumableMemory**, or both as consumable resources in the **SCHEDULE_BY_RESOURCES** global configuration keyword. This enables the LoadLeveler scheduler to consider these consumable resources.
2. Define **ConsumableCpus**, **ConsumableMemory**, or both in the **ENFORCE_RESOURCE_USAGE** global configuration keyword. This enables enforcement of these consumable resources by AIX WLM.
3. Define **hard**, **soft** or **shares** in the **ENFORCE_RESOURCE_POLICY** configuration keyword. This defines what policy is used by LoadLeveler when setting WLM class resource entitlements.
4. (Optional) Set the **ENFORCE_RESOURCE_MEMORY** configuration keyword to **true**. This setting allows AIX WLM to limit the real memory usage of a WLM class as precisely as possible. When a class exceeds its limit, all processes in the class are killed.

LoadLeveler with AIX Workload Manager

Rule: `ConsumableMemory` must be defined in the `ENFORCE_RESOURCE_USAGE` keyword in the global configuration file, or LoadLeveler does not consider the `ENFORCE_RESOURCE_MEMORY` keyword to be valid.

Tips:

- When set to true, the `ENFORCE_RESOURCE_MEMORY` keyword overrides the policy set through the `ENFORCE_RESOURCE_POLICY` keyword for `ConsumableMemory` only. The `ENFORCE_RESOURCE_POLICY` keyword value still applies for `ConsumableCpus`.
 - `ENFORCE_RESOURCE_MEMORY` may be set in either the global or the local configuration file. In the global configuration file, this keyword sets the default value for all the machines in the LoadLeveler cluster. If the keyword also is defined in a local file, the local setting overrides the global setting.
5. Using the `resources` keyword in a machine stanza in the administration file, define the CPU and real memory machine resources available for user jobs.
 - The `ConsumableCpus` reserved word accepts a count value of "all." This indicates that the initial resource count will be obtained from the Startd machine update value for CPUs.
 - If no resources are defined for a machine, then no enforcement will be done on that machine.
 - If the count specified by the administrator is greater than what the Startd update indicates, the initial count value will be reduced to match what the Startd reports.
 - If the count specified by the administrator is less than what the Startd update indicates, the WLM resource shares assigned to a job will be adjusted to represent that difference and a WLM softlimit will be defined for each WLM class. For example, if the administrator defines 8 CPUs on a 16 CPU machine, then a job requesting 4 CPUs will get a share of 4 and a softlimit of 50%.
 - Use caution when determining the amount of real memory available for user jobs. A certain percentage of a machine's real memory will be dedicated to the Default and System WLM classes and will not be included in the calculation of real memory available for users jobs. Start LoadLeveler with the `ENFORCE_RESOURCE_USAGE` keyword enabled and issue `wlmstat -v -m`. Look at the `npg` column to determine how much memory is being used by these classes.
 6. Decide if all jobs should have their CPU or real memory resources enforced and then define the `ENFORCE_RESOURCE_SUBMISSION` global configuration keyword.
 - If the value specified is true, LoadLeveler will check all jobs at submission time for the `resources` keyword. The job's `resources` keyword needs to have the same resources specified as the `ENFORCE_RESOURCE_USAGE` keyword in order to be submitted.
 - If the value specified is false, no checking will be done and jobs submitted without the `resources` keyword will not have resources enforced and may interfere with other jobs whose resources are enforced.
 - To support existing job command files without the `resources` keyword, the `default_resources` keyword in the class stanza can be defined. The `default_resources` keyword needs to be defined in the default interactive class to support interactive jobs.

For more information on the `ENFORCE_RESOURCE_USAGE` and the `ENFORCE_RESOURCE_SUBMISSION` keywords, see "Defining usage policies for consumable resources" on page 56.

LoadLeveler support for checkpointing jobs

Checkpointing is a method of periodically saving the state of a job step so that if the step does not complete it can be restarted from the saved state. When checkpointing is enabled, checkpoints can be initiated from within the application at major milestones, or by the user, administrator or LoadLeveler external to the application. Both serial and parallel job steps can be checkpointed.

Once a job step has been successfully checkpointed, if that step terminates before completion, the checkpoint file can be used to resume the job step from its saved state rather than from the beginning. When a job step terminates and is removed from the LoadLeveler job queue, it can be restarted from the checkpoint file by submitting a new job and setting the **restart_from_ckpt = yes** job command file keyword. When a job is terminated and remains on the LoadLeveler job queue, such as when a job step is vacated, the job step will automatically be restarted from the latest valid checkpoint file. A job can be vacated as a result of flushing a node, issuing checkpoint and hold, stopping or recycling LoadLeveler or as the result of a node crash.

To find out more about checkpointing jobs, use the information in Table 31.

Table 31. Roadmap of tasks for checkpointing jobs

Subtask	Associated instructions (see . . .)
Preparing the LoadLeveler environment for checkpointing and restarting jobs	<ul style="list-style-type: none"> • “Checkpoint keyword summary” • “Planning considerations for checkpointing jobs” on page 130 • “AIX checkpoint and restart limitations” on page 131 • “Naming checkpoint files and directories” on page 134
Checkpointing and restarting jobs	<ul style="list-style-type: none"> • “Checkpointing a job” on page 214 • “Removing old checkpoint files” on page 135
Correctly specifying configuration and administration file keywords	<ul style="list-style-type: none"> • Chapter 12, “Configuration file reference,” on page 243 • Chapter 13, “Administration file reference,” on page 301

Checkpoint keyword summary

The following is a summary of keywords associated with the checkpoint and restart function.

- **Configuration file keywords**
 - CKPT_CLEANUP_INTERVAL
 - CKPT_CLEANUP_PROGRAM
 - CKPT_EXECUTE_DIR
 - MAX_CKPT_INTERVAL
 - MIN_CKPT_INTERVAL

For more information about these keywords, see Chapter 12, “Configuration file reference,” on page 243.

- **Administration file keywords**
 - ckpt_dir
 - ckpt_time_limit

For more information about these keywords, see Chapter 13, “Administration file reference,” on page 301.

- **Job command file keywords**

Checkpointing jobs

- `checkpoint`
- `ckpt_dir`
- `ckpt_execute_dir`
- `ckpt_file`
- `ckpt_time_limit`
- `restart_from_ckpt`

For more information about these keywords, see “Job command file keyword descriptions” on page 336.

Planning considerations for checkpointing jobs

Review the following guidelines before you submit a checkpointing job:

- **Plan for jobs that you will restart on different nodes**

If you plan to migrate jobs (restart jobs on a different node or set of nodes), you should understand the difference between writing checkpoint files to a local file system versus a global file system (such as AFS or GPFS). The `ckpt_file`, and `ckpt_dir` keywords in the job command and configuration files allow you to write to either type of file system. If you are using a local file system, before restarting the job from checkpoint, make certain that the checkpoint files are accessible from the machine on which the job will be restarted.

- **Reserve adequate disk space**

A checkpoint file requires a significant amount of disk space. The checkpoint will fail if the directory where the checkpoint file is written does not have adequate space. For serial jobs, one checkpoint file will be created. For parallel jobs, one checkpoint file will be created for each task. Since the old set of checkpoint files are not deleted until the new set of files are successfully created, the checkpoint directory should be large enough to contain two sets of checkpoint files. You can make an accurate size estimate only after you have run your job and noticed the size of the checkpoint file that is created.

- **Plan for staging executables**

If you want to stage the executable for a job step, use the `ckpt_execute_dir` keyword to define the directory where LoadLeveler will save the executable. This directory cannot be the same as the current location of the executable file, or LoadLeveler will not stage the executable.

You may define the `ckpt_execute_dir` keyword in either the configuration file or the job command file. To decide where to define the keyword, use the information in Table 32.

Table 32. Deciding where to define the directory for staging executables

If the <code>ckpt_execute_dir</code> keyword is defined in:	Then the following information applies:
The configuration file only	<ul style="list-style-type: none">• LoadLeveler stages the executable file in a new subdirectory of the specified directory. The name of the subdirectory is the job step ID.• The user is the owner of the subdirectory and has permission 700.• If the user issues the <code>llckpt</code> command with the <code>-k</code> option, LoadLeveler deletes the staged executable.• LoadLeveler will delete the subdirectory and the staged executable when the job step ends.

Table 32. Deciding where to define the directory for staging executables (continued)

If the <code>ckpt_execute_dir</code> keyword is defined in:	Then the following information applies:
The job command file only	<ul style="list-style-type: none"> • LoadLeveler stages the executable file in the directory specified in the job command file. • The user is the owner of the file and has execute permission for it. • The user is responsible for deleting the staged file after the job step ends.
Both the configuration and job command files	
Neither file (the keyword is not defined)	LoadLeveler does not stage the executable file for the job step.

- **Set your checkpoint file size to the maximum**

To make sure that your job can write a large checkpoint file, assign your job to a job class that has its file size limit set to the maximum (unlimited). In the administration file, set up a class stanza for checkpointing jobs with the following entry:

```
file_limit = unlimited,unlimited
```

This statement specifies that there is no limit on the maximum size of a file that your program can create.

- **Choose a unique checkpoint file name**

To prevent another job step from writing over your checkpoint file with another checkpoint file, make certain that your checkpoint file name is unique. The `ckpt_dir` and `ckpt_file` keywords give you control over the location and name of these files.

For mode information, see “Naming checkpoint files and directories” on page 134.

AIX checkpoint and restart limitations

- The following items cannot be checkpointed:
 - Programs that are being run under:
 - The dynamic probe class library (DPCL).
 - Any debugger.
 - MPI programs that are *not* compiled with `mpcc_r`, `mpCC_r`, `mpxlf_r`, `mpxlf90_r`, or `mpxlf95_r`.
 - Processes that use:
 - Extended `shmat` support
 - Pinned shared memory segments.
 - Sets of processes in which any process is running a `setuid` program when a checkpoint occurs.
 - Sets of processes if any process is running a `setgid` program when a checkpoint occurs.
 - Interactive parallel jobs for which POE input or output is a pipe.
 - Interactive parallel jobs for which POE input or output is redirected, unless the job is submitted from a shell that had the CHECKPOINT environment variable set to `yes` before the shell was started. If POE is run from inside a shell script and is run in the background, the script must be started from a shell started in the same manner for the job to be checkpointable.
 - Interactive POE jobs for which the `su` command was used prior to checkpointing or restarting the job.
- The node on which a process is restarted must have:

Checkpointing jobs

- The same operating system level (including PTFs). In addition, a restarted process may not load a module that requires a system call from a kernel extension that was not present at checkpoint time.
- The same switch type as the node where the checkpoint occurred.

If any threads in a process were bound to a specific processor ID at checkpoint time, that processor ID must exist on the node where that process is restarted.

- If the LoadLeveler cluster contains nodes running a mix of 32-bit and 64-bit kernels then applications must be checkpointed and restarted on the same set of nodes. For more information, see “llckpt - Checkpoint a running job step” on page 400 and the `restart_on_same_nodes` keyword description.
- For a parallel job, the number of tasks and the task geometry (the tasks that are common within a node) must be the same on a restart as it was when the job was checkpointed.
- Any regular file open in a process when it is checkpointed must be present on the node where that process is restarted, including the executable and any dynamically loaded libraries or objects.
- If any process uses sockets or pipes, user callbacks should be registered to save data that may be “in flight” when a checkpoint occurs, and to restore the data when the process is resumed after a checkpoint or restart. Similarly, any user shared memory in a parallel task should be saved and restored.
- A checkpoint operation will not begin on a process until each user thread in that process has released all pthread locks, if held. This can potentially cause a significant delay from the time a checkpoint is issued until the checkpoint actually occurs. Also, any thread of a process that is being checkpointed that does not hold any pthread locks and tries to acquire one will be stopped immediately. There are no similar actions performed for atomic locks (`_check_lock` and `_clear_lock`, for example).
- Atomic locks must be used in such a way that they do not prevent the releasing of pthread locks during a checkpoint. For example, if a checkpoint occurs and thread 1 holds a pthread lock and is waiting for an atomic lock, and thread 2 tries to acquire a different pthread lock (and does not hold any other pthread locks) before releasing the atomic lock that is being waited for in thread 1, the checkpoint will hang.
- A process must not hold a pthread lock when creating a new process (either implicitly using `popen`, for example, or explicitly using `fork`) if releasing the lock is contingent on some action of the new process. Otherwise, a checkpoint could occur which would cause the child process to be stopped before the parent could release the pthread lock causing the checkpoint operation to hang.
- The checkpoint operation will hang if any user pthread locks are held across:
 - Any collective communication calls in MPI or LAPI
 - Calls to `mpc_init_ckpt` or `mp_init_ckpt`
- Processes cannot be profiled at the time a checkpoint is taken.
- There can be no devices other than TTYs or `/dev/null` open at the time a checkpoint is taken.
- Open files must either have an absolute pathname that is less than or equal to `PATHMAX` in length, or must have a relative pathname that is less than or equal to `PATHMAX` in length from the current directory at the time they were opened. The current directory must have an absolute pathname that is less than or equal to `PATHMAX` in length.
- Semaphores or message queues that are used within the set of processes being checkpointed must only be used by processes within the set of processes being checkpointed. This condition is not verified when a set of processes is

checkpointed. The checkpoint and restart operations will succeed, but inconsistent results can occur after the restart.

- The processes that create shared memory must be checkpointed with the processes using the shared memory if the shared memory is ever detached from all processes being checkpointed. Otherwise, the shared memory may not be available after a restart operation.
- The ability to checkpoint and restart a process is not supported for B1 and C2 security configurations.
- A process can only checkpoint another process if it can send a signal to the process. In other words, the privilege checking for checkpointing processes is identical to the privilege checking for sending a signal to the process. A privileged process (the effective user ID is 0) can checkpoint any process. A set of processes can only be checkpointed if each process in the set can be checkpointed.
- A process can only restart another process if it can change its entire privilege state (real, saved, and effective versions of user ID, group ID, and group list) to match that of the restarted process. A set of processes can only be restarted if each process in the set can be restarted.
- The only DCE function supported is DCE credential forwarding by LoadLeveler using the DCE_AUTHENTICATION_PAIR configuration keyword. DCE credential forwarding is for the sole purpose of DFS™ access by the application.
- The following functions will return ENOTSUP if called in a job that has enabled checkpointing:
 - clock_getcpuclockid()
 - clock_getres()
 - clock_gettime()
 - clock_nanosleep()
 - clock_settime()
 - mlock()
 - mlockall()
 - mq_close()
 - mq_getattr()
 - mq_notify()
 - mq_open()
 - mq_receive()
 - mq_send()
 - mq_setattr()
 - mq_timedreceive()
 - mq_timedsend()
 - mq_unlink()
 - munlock()
 - munlockall()
 - nanosleep()
 - pthread_barrier_destroy()
 - pthread_barrier_init()
 - pthread_barrier_wait()
 - pthread_barrierattr_destroy()
 - pthread_barrierattr_getpshared()
 - pthread_barrierattr_init()
 - pthread_barrierattr_setpshared()
 - pthread_condattr_getclock()
 - pthread_condattr_setclock()
 - pthread_getcpuclockid()
 - pthread_mutex_getprioceiling()

Checkpointing jobs

- pthread_mutex_setprioceiling()
- pthread_mutex_timedlock()
- pthread_mutexattr_getprioceiling()
- pthread_mutexattr_getprotocol()
- pthread_mutexattr_setprioceiling()
- pthread_mutexattr_setprotocol()
- pthread_rwlock_timedrdlock()
- pthread_rwlock_timedwrlock()
- pthread_setschedprio()
- pthread_spin_destroy()
- pthread_spin_init()
- pthread_spin_lock()
- pthread_spin_trylock()
- pthread_spin_unlock()
- sched_get_priority_max()
- sched_get_priority_min()
- sched_getparam()
- sched_getscheduler()
- sched_rr_get_interval()
- sched_setparam()
- sched_setscheduler()
- sem_close()
- sem_destroy()
- sem_getvalue()
- sem_init()
- sem_open()
- sem_post()
- sem_timedwait()
- sem_trywait()
- sem_unlink()
- sem_wait()
- shm_open()
- shm_unlink()
- timer_create()
- timer_delete()
- timer_getoverrun()
- timer_gettime()
- timer_settime()

Naming checkpoint files and directories

At checkpoint time, a checkpoint file and potentially an error file will be created. For jobs which are enabled for checkpoint, a control file may be generated at the time of job submission. The directory which will contain these files must pre-exist and have sufficient space and permissions for these files to be written. The name and location of these files will be controlled through keywords in the job command file or the LoadLeveler configuration. The file name specified is used as a base name from which the actual checkpoint file name is constructed. To prevent another job step from writing over your checkpoint file, make certain that your checkpoint file name is unique. For serial jobs and the master task (POE) of parallel jobs, the checkpoint file name will be *<basename>.Tag*. For a parallel job, a checkpoint file is created for each task. The checkpoint file name will be *<basename>.Taskid.Tag*.

The tag is used to differentiate between a current and previous checkpoint file. A control file may be created in the checkpoint directory. This control file contains

information LoadLeveler uses for restarting certain jobs. An error file may also be created in the checkpoint directory. The data in this file is in a machine readable format. The information contained in the error file is available in mail, LoadLeveler logs or is output of the checkpoint command. Both of these files are named with the same base name as the checkpoint file with the extensions *.cntl* and *.err*, respectively.

Naming checkpoint files for serial and batch parallel jobs

The following describes the order in which keywords are checked to construct the full path name for a serial or batch checkpoint file:

- Base name for the checkpoint file name
 1. The **ckpt_file** keyword in the job command file
 2. The default file name [*< jobname.>*]*<job_step_id>*.ckpt

Where:

jobname

The *job_name* specified in the Job Command File. If *job_name* is not specified, it is omitted from the default file name

job_step_id

Identifies the job step that is being checkpointed

- Checkpoint Directory Name
 1. The **ckpt_file** keyword in the job command file, if it contains a "/" as the first character
 2. The **ckpt_dir** keyword in the job command file
 3. The **ckpt_dir** keyword specified in the class stanza of the LoadLeveler admin file
 4. The default directory is the initial working directory

Note that two or more job steps running at the same time cannot both write to the same checkpoint file, since the file will be corrupted.

Naming checkpointing files for interactive parallel jobs

The following describes the order in which keywords and variables are checked to construct the full path name for the checkpoint file for an interactive parallel job.

- Checkpoint File Name
 1. The value of the MP_CKPTFILE environment variable within the POE process
 2. The default file name, poe.ckpt.<pid>
- Checkpoint Directory Name
 1. The value of the MP_CKPTFILE environment variable within the POE process, if it contains a full path name.
 2. The value of the MP_CKPTDIR environment variable within the POE process.
 3. The initial working directory.

Note: The keywords **ckpt_dir** and **ckpt_file** are not allowed in the command file for an interactive session. If they are present, they will be ignored and the job will be submitted.

Removing old checkpoint files

To keep your system free of checkpoint files that are no longer necessary, LoadLeveler provides two keywords to help automate the process of removing these files:

- **CKPT_CLEANUP_PROGRAM**
- **CKPT_CLEANUP_INTERVAL**

Checkpointing jobs

Both keywords must contain valid values to automate this process. For information about configuration file keyword syntax and other details, see Chapter 12, “Configuration file reference,” on page 243.

LoadLeveler scheduling affinity support

Enabling scheduling affinity allows LoadLeveler jobs to utilize the memory and adapter affinity features available in IBM eServers equipped with System p5 or System p4 CPU architecture. The Resource Set (RSet) functionality available in AIX 5L is made available for jobs to use in order to take advantage of these affinity features. Once the scheduling affinity feature has been configured, users can request scheduling affinity options for their jobs as a requirement or as a preference.

Memory affinity is a special purpose option for improving performance on IBM System p5 or System p4 machines. These machines contain Multiple Chip Modules (MCMs), each containing multiple processors. System memory is attached to these MCMs. While any processor can access all of the memory in the system, a processor has faster access and higher bandwidth when addressing memory that is attached to its own MCM rather than memory attached to the other MCMs in the system. The concept of affinity also applies to the I/O subsystem. The processors running on CPUs from an MCM have faster access to the adapters attached to the I/O slots of that MCM. I/O affinity will be referred to as adapter affinity in this topic. For more information on memory and adapter affinity, see *AIX Performance Management Guide*.

An RSet contains bit maps for CPU and memory pool resources. The RSet APIs available in AIX 5L can be used to attach RSets to processes. Attaching an RSet to a process limits the process to only using the resources contained in the RSet. One of the main uses of RSets is to limit the application processes to run only on the processors contained in a single MCM and hence to benefit from memory affinity. For more information on RSets, see *AIX System Management Guide: Operating System and Devices*.

Configuring LoadLeveler to use scheduling affinity

Taking advantage of scheduling affinity requires certain changes to LoadLeveler configuration files for machines, and in some cases, LoadLeveler administration files. The steps for enabling LoadLeveler to use scheduling affinity are:

Configure one or more machines to enable scheduling affinity

This is accomplished through the use of the **RSET_SUPPORT** configuration file keyword. Machines which are configured with this keyword indicate the ability to service jobs requesting or requiring scheduling affinity. Enable **RSET_SUPPORT** with one of these values:

- Choose **RSET_MCM_AFFINITY** to allow jobs specifying `#@rset = RSET_MCM_AFFINITY` to run on this node. The **RSET_MCM_AFFINITY** option enables scheduling affinity. When this option is specified, LoadLeveler will create and attach RSets to task processes so that the RSet CPUs will be from same MCM..
- Choose **RSET_CONSUMABLE_CPUS** allow jobs specifying `#@rset = RSET_CONSUMABLE_CPUS` to run on this node. The **RSET_CONSUMABLE_CPUS** option is used to indicate that all of the tasks need to be attached to RSets with a number of CPUs equal to the number of **ConsumableCPUs** requested by the job. The CPUs will be selected such that the same CPU will not be shared among different

tasks. Using this option requires LoadLeveler to be configured to use **ConsumableCPUs**. The difference between this option and the **RSET_MCM_AFFINITY** option is that when this option is specified, CPUs are selected regardless of their location with respect to an MCM.

- Choose **RSET_USER_DEFINED** to allow jobs specifying `#@rset = RSET_USER_DEFINED` to run on this node. The **RSET_USER_DEFINED** option enables scheduling affinity, allowing users more control over scheduling affinity parameters by allowing the use of user-defined RSets. Through the use of user-defined RSets, users can utilize new RSet features before a LoadLeveler implementation is released. This option also allows users to specify a different number of CPUs in their RSets depending on the needs of each task.

Note: If you do not plan to create your own RSets and only want to take advantage of memory or adapter affinity, we suggest using the **RSET_MCM_AFFINITY** option. This allows LoadLeveler to dynamically allocate CPU and adapter resources to meet the scheduling affinity requirement of the job.

See “Configuration file keyword descriptions” on page 244 for more information on the **RSET_SUPPORT** keyword.

Configure LoadLeveler to recognize ConsumableCPUs

ConsumableCPUs must be enabled on a machine if the **RSET_SUPPORT** configuration file keyword is specified with a value of **RSET_CONSUMABLE_CPUS** in global configuration file or local configuration file for the same machine.

The **RSET_MCM_AFFINITY** option for keyword **RSET_SUPPORT** can be used with or without **ConsumableCPUs** enabled. The CPU allocation to task RSets and task MCM selection will be slightly different in these two scenarios. If **ConsumableCPUs** is not specified, all CPUs from an MCM will be included in the task RSet and the MCM where the least number of tasks is running will be selected for the next task. If **ConsumableCPUs** is enabled, the task RSet will have the **ConsumableCPUs** number of CPUs in it and for the next task central manager will select an MCM based on number of unused CPUs available on MCMs.

ConsumableCPUs can be specified in several ways:

- If a list of CPUs is specified, only CPUs in the list are made available to LoadLeveler. CPUs specified in this manner must be on nodes that support RSets.
- If the number, *n*, of CPUs available for consumption is specified, the first *n* CPUs on nodes supporting RSets are marked available to LoadLeveler.
- If the reserved word **all** is specified, all CPUs on nodes supporting RSets are marked available to LoadLeveler.

For more information on specifying **ConsumableCPUs**, see the **resource** keyword description in “Administration file keyword descriptions” on page 306.

Linux CPU affinity support

Linux CPU affinity support allows users to allocate a CPU on a compute node for the exclusive use of a job. This support is only available if the compute node is running the Linux kernel 2.6.

Linux CPU affinity support

Table 33 illustrates what level of CPU affinity will be supported on each LoadLeveler platform:

Table 33. Linux platforms for CPU affinity

CPU type	Operating system	Type of CPU affinity support
POWER	SLES 9, SLES 10, and RHEL4 on POWER platforms	LOGICAL only
Intel x86	SLES 9, SLES 10, and RHEL4 on Intel IA-32 32-bit platforms	PHYSICAL and LOGICAL
AMD Opteron and Intel EM64T	SLES 9, SLES 10, and RHEL4 on AMD Opteron 64-bit platforms	PHYSICAL and LOGICAL
ANY	RHEL 3	NONE

To enable exclusive allocation of CPUs to jobs, you must set the following configuration keyword to a value of either **LOGICAL** or **PHYSICAL**:

```
ALLOC_EXCLUSIVE_CPU_PER_JOB = LOGICAL|PHYSICAL
```

For more information, see the `ALLOC_EXCLUSIVE_CPU_PER_JOB` configuration keyword.

Assigning Linux CPU affinity to application processes

The following steps describe how Linux CPU affinity is assigned to application processes:

1. When Linux CPU affinity support is enabled on a compute node, LoadLeveler detects the number of processors available on the node at startup.
 2. Select the **LOGICAL** or **PHYSICAL** option:
 - If the **LOGICAL** option is selected, each processor unit on the node will be assigned only one job to run. **LOGICAL** implies to use all available processors to run jobs. For example, a node with two Intel x86 processors with hyperthreading turned ON will be treated as a node with four processors. A node with two dual-core AMD Opteron processors will be treated as a node with four processors.
 - If the **PHYSICAL** option is selected, one physical processor package will be assigned only one job to run. The **PHYSICAL** option is meant to allow a user to treat hyperthreaded processors and multicore processors as a single unit so that a job has dedicated computing resources. For example, a node with two Intel x86 processors with hyperthreading turned ON, will be treated as a node with two physical processors. Similarly, a node with two dual-core AMD Opteron processors will be treated as a node with two physical processors.
- Note:** **LOGICAL** and **PHYSICAL** options will be treated identically on pSeries servers. In other words, if a pSeries node has n physical processors and $2n$ logical processors, the number of available processors will always be assumed to be $2n$.
3. When a job begins running, its CPU affinity is set to one of the available processors. As long as the job is running, LoadLeveler will not set CPU affinity of any other job to this processor.
 4. When a job completes, the processor on which the job was running will become available and LoadLeveler might assign it to the next job.

LoadLeveler multicluster support

To provide a more scalable runtime environment and more efficient workload balancing, you may configure a LoadLeveler multicluster environment. A LoadLeveler multicluster environment consists of two or more LoadLeveler clusters, grouped together through network connections that allow the clusters to share resources. These clusters may be AIX, Linux, or mixed clusters.

Within a LoadLeveler multicluster environment:

- The **local cluster** is the cluster from which the user submits jobs or issues commands.
- A **remote cluster** is a cluster that accepts job submissions and commands from the local cluster.
- A **local gateway Schedd** is a Schedd within the local cluster serving as an inbound point from some remote cluster, an outbound point to some remote cluster, or both.
- A **remote gateway Schedd** is a Schedd within a remote cluster serving as an inbound point from the local cluster, an outbound point to the local cluster, or both.
- A **local central manager** is the central manager in the same cluster as the local gateway Schedd.
- A **remote central manager** is the central manager in the same cluster as a remote gateway Schedd.

A LoadLeveler multicluster environment addresses scalability and workload balancing issues by providing the ability to:

- Distribute workload among LoadLeveler clusters when jobs are submitted.
- Easily access multiple LoadLeveler cluster resources.
- Display information about the multicluster.
- Monitor and control operations in a multicluster.
- Transfer idle jobs from one cluster to another.
- Transfer user input and output files between clusters.
- Enable LoadLeveler to operate in a secure environment where clusters are separated by a firewall.

Table 34 shows the multicluster support subtasks with a pointer to the associated instructions:

Table 34. Multicluster support subtasks and associated instructions

Subtask	Associated instructions (see . . .)
Configure a LoadLeveler multicluster	“Configuring a LoadLeveler multicluster” on page 140
Submit and monitor jobs in a LoadLeveler multicluster	“Submitting and monitoring jobs in a LoadLeveler multicluster” on page 205

Table 35. Multicluster support related topics

Related topics	Additional information (see . . .)
Administration file: Cluster stanzas	“Defining clusters” on page 93
Administration file: Cluster keywords	“Administration file keyword descriptions” on page 306
Configuration file: Cluster keywords	“Configuration file keyword descriptions” on page 244

LoadLeveler multicluster support

Table 35. Multicluster support related topics (continued)

Related topics	Additional information (see . . .)
Job command file: Cluster keywords	"Job command file keyword descriptions" on page 336
Commands and APIs	Chapter 16, "Commands," on page 385 or Chapter 17, "Application programming interfaces (APIs)," on page 503
Diagnosis and messages	<i>TWS LoadLeveler: Diagnosis and Messages Guide</i>

Configuring a LoadLeveler multicluster

Table 36 lists the subtasks for configuring a LoadLeveler multicluster:

Table 36. Subtasks for configuring a LoadLeveler multicluster

Subtask	Associated instructions (see . . .)
Configure the LoadLeveler multicluster environment	<ul style="list-style-type: none">• "Steps for configuring a LoadLeveler multicluster" on page 141• "Steps for securing communications within a LoadLeveler multicluster" on page 143
Display information about the LoadLeveler multicluster environment	<ul style="list-style-type: none">• Use the llstatus command:<ul style="list-style-type: none">– With the -X option to display information about machines in the multicluster.– With the -C option to display information defined in cluster stanzas in the administration file.• Use the llclass command with the -X option to display information about classes on any cluster (local or remote).• Use the llq command with the -X option to display information about jobs on any cluster (local or remote).

Table 36. Subtasks for configuring a LoadLeveler multicluster (continued)

Subtask	Associated instructions (see . . .)
Monitor and control operations in the LoadLeveler multicluster environment	<p>Existing LoadLeveler user commands accept the -X option for a multicluster environment.</p> <p>Rules:</p> <ul style="list-style-type: none"> • Administrator only commands are not applicable in a multicluster environment. • The options -x, -W, -s, and -p cannot be specified together with the -X option on the llmodify command. • The options -x and -w cannot be specified together with the -X option on the llq command. • The -X option on the following commands is restricted to a single cluster: <ul style="list-style-type: none"> – llcancel – llckpt – llhold – llmodify – llprio • The following commands are not applicable in a multicluster environment: <ul style="list-style-type: none"> – llacctmrg – llchres – lldbconvert – llextrPD – llinit – llmkres – llqres – llrmres – llrunscheduler – llsummary

Steps for configuring a LoadLeveler multicluster

The primary task for configuring a LoadLeveler multicluster environment is to enable communication between gateway Schedd daemons on all of the clusters in the multicluster. To do so requires defining each Schedd daemon as either local or remote, and defining the inbound and outbound hosts with which the daemon will communicate.

Before you begin: You need to know that:

- A single machine may be defined as an inbound or outbound host, or as both.
- A single cluster must belong to only one multicluster.
- A single multicluster must consist of 10 or fewer clusters.
- Clusters must have unique host names within the multicluster network domain space.
- The inbound Schedd becomes the **schedd_host** of all remote jobs it receives.

Perform the following steps to configure a LoadLeveler multicluster:

1. In the administration file, define one cluster stanza for each cluster in the LoadLeveler multicluster environment.

Rules:

- You must define one cluster as the local cluster.
- You must code the following required cluster-stanza keywords and variable values:

LoadLeveler multicluster support

```
cluster_name: type=cluster
outbound_hosts = hostname[(cluster_name)]
inbound_hosts = hostname[(cluster_name)]
```

- If you want to allow users to submit remote jobs to the local cluster, the list of inbound hosts must include the name of the inbound Schedd and the cluster you are defining as remote or you must specify the name of an inbound Schedd without any cluster specification so that it defaults to being an inbound Schedd for all clusters.
 - If the configuration file keyword **SCHEDD_STREAM_PORT** for any cluster is set to use a port other than the default value of 9605, you must set the **inbound_schedd_port** keyword in the cluster stanza for that cluster.
2. (Optional) If the local cluster wants to provide job distribution where users allow LoadLeveler to select the appropriate cluster for job submission based on administration defined objectives, then define an installation exit to be executed at submit time using the **CLUSTER_METRIC** configuration keyword. You can use the LoadLeveler data access APIs in this exit to query other clusters for information about possible metrics, such as the number of jobs in a specified job class, the number of jobs in the idle queue, or the number of free nodes in the cluster. For more detailed information, see “**CLUSTER_METRIC**” on page 250.

Tip: LoadLeveler provides a set of sample exits for you to use as models. These samples are in the `$(RELEASEDIR)/samples/llcluster` directory.
 3. (Optional) If the local cluster wants to perform user mapping on jobs arriving from remote clusters, define the **CLUSTER_USER_MAPPER** configuration keyword. For more information, see “**CLUSTER_USER_MAPPER**” on page 252.
 4. (Optional) If the local cluster wants to perform job filtering on jobs received from remote clusters, define the **CLUSTER_REMOTE_JOB_FILTER** configuration keyword. For more information, see “**CLUSTER_REMOTE_JOB_FILTER**” on page 251.
 5. Notify LoadLeveler daemons by issuing the **llctl** command with either the **reconfig** or **recycle** keyword. Otherwise, LoadLeveler will not process the modifications you made to the administration file.

Additional considerations:

- Remote jobs are subjected to the same configuration checks as locally submitted jobs. Examples include account validation, class limits, include lists, and exclude lists.
- Remote jobs will be processed by the local **submit_filter** prior to submission to a remote cluster.
- Any tracker program specified in the API parameters will be invoked upon the scheduling cluster nodes.
- If a step is enabled for checkpoint and the **ckpt_execute_dir** is not specified, LoadLeveler will not copy the executable to the remote cluster, the user must ensure that executable exists on the remote cluster. If the executable is not in a shared file system, the executable can be copied to the remote cluster using the **cluster_input_file** job command file keyword.
- If the job command file is also the executable and the job is submitted or moved to a remote cluster, the **\$(executable)** variable will contain the full path name of the executable on the local cluster from which it came. This differs from the behavior on the local cluster, where the **\$(executable)** variable will be the command line argument passed to the **llsubmit** command. If you only want the file name, use the **\$(base_executable)** variable.

Steps for securing communications within a LoadLeveler multicluster

Configuring LoadLeveler to use the OpenSSL library enables it to operate in a secure environment where clusters are separated by a firewall.

Perform the following steps to configure LoadLeveler to use OpenSSL in a multicluster environment:

1. Install SSL using the standard platform installation process.
2. Ensure a link exists from the installed SSL library to:
 - a. `/usr/lib/libssl.so` for 32-bit Linux platforms.
 - b. `/usr/lib64/libssl.so` for 64-bit Linux platforms.
 - c. `/usr/lib/libssl.a` for AIX platforms.
3. Create the SSL authorization keys by invoking the `llclusterauth` command with the `-k` option on all local gateway schedds.

Result: LoadLeveler creates a public key, a private key, and a security certificate for each gateway node.
4. Distribute the public keys to remote gateway schedds on other secure clusters. This is done by exchanging the public keys with the other clusters you wish to communicate with.
 - for AIX, public keys can be found in the `/var/LoadL/ssl/id_rsa.pub` file.
 - for Linux, public keys can be found in the `/var/opt/LoadL/ssl/id_rsa.pub` file.
5. Copy the public keys of the clusters you wish to communicate with into the `authorized_keys` directory on your inbound Schedd nodes.
 - for AIX, `/var/LoadL/ssl/authorized_keys`
 - for Linux, `/var/opt/LoadL/ssl/authorized_keys`
 - The authorization key files can be named anything within the `authorized_keys` directory.
6. Define the cluster stanzas within the LoadLeveler administration file, using the `multicluster_security = SSL` keyword. Define the keyword `ssl_cipher_list` if a specific OpenSSL cipher encryption method is desired. Use `secure_schedd_port` to define the port number to be used for secure inbound transactions to the cluster.
7. Notify LoadLeveler daemons by issuing the `llctl -g` command with the `recycle` keyword. Otherwise, LoadLeveler will not process the modifications you made to the administration file.
8. Configure firewalls to accept connections to the `secure_schedd_port` numbers you defined in the administration file.

LoadLeveler Blue Gene support

Blue Gene is a massively parallel system based on a scalable cellular architecture which exploits a very large number of tightly interconnected compute nodes (C-nodes). Each C-node is based on system-on-a-chip technology, and is comprised of two PowerPC® 440 processors.

To take advantage of Blue Gene support, you must be using the BACKFILL scheduler and the Blue Gene system must be at a release 2 or higher.

LoadLeveler Blue Gene support

While LoadLeveler Blue Gene support is available on all platforms, Blue Gene software is only supported on IBM POWER servers running SLES 9. This limitation currently restricts LoadLeveler Blue Gene support to SLES 9 on IBM POWER servers.

Terms you should know:

- **Compute nodes**, also called C-nodes, are system-on-a-chip nodes that execute at most a single job at a time. All the C-nodes are interconnected in a three-dimensional toroidal pattern. Each C-node has a unique address and location in the three-dimensional toroidal space. Compute nodes execute the jobs' tasks. Compute nodes run a minimal custom operating system called BLRTS.
- **Front End Nodes (FEN)** are machines from which users and administrators interact with Blue Gene. Applications are compiled on and submitted for execution in the Blue Gene core from FENs. User interactions with applications, including debugging, are also performed from the FENs.
- The **Service Node** is dedicated hardware that runs software to control and manage the Blue Gene system.
- **I/O nodes** are special nodes that connect the compute nodes to the outside world. I/O nodes allow processes that are executing in the compute nodes to perform I/O operations, such as accessing files, and to communicate with the job management system. Each I/O node serves anywhere from 8 to 64 C-nodes, depending on the physical configuration.
- **mpirun** is a program that is executed partly on the Front End Node, and partly on the Service Node. mpirun controls and monitors the parallel Blue Gene job. The mpirun program is executed by the user program that is run on the FEN by LoadLeveler.
- A **base partition (BP)** is a group of compute nodes connected in a 3D rectangular pattern and their controlled I/O nodes. A base partition is one of the basic allocation units for jobs. For example, an allocation for the job will require at least one base partition, unless an allocation requests a small partition, in which case sub base partition allocation is possible.
- A **small partition** is a group of C-nodes which are part of one base partition. Valid small partitions have size of 32 or 128 C-nodes.
- A **partition** is a group of base partitions, switches, and switch states allocated to a job. A partition is predefined or is created on demand to execute a job. Partitions are physically (electronically) isolated from each other (for example, messages cannot flow outside an allocated partition). A partition can have the topology of a mesh or a torus.
- The **Control System** is a component that serves as the interface to the Blue Gene system. It contains persistent storage with configuration and status information on the entire system. It also provides various services to perform actions on the Blue Gene system, such as launching a job.
- A **node card** is a group of 32 compute nodes within a base partition. This is the minimal allocation size for a partition.
- A **quarter** is a group of 4 node cards. This is a logical grouping of node cards within a base partition. A quarter, which is 128 compute nodes, is the next smallest allowed allocation size for a partition after a node card.
- A **switch state** is a set of internal switch connections which physically "wire" the partition. A switch has a number of incoming and outgoing wires. An internal switch connection physically connects one incoming wire with one outgoing wire, setting up a communication path between base partitions.

For more information on the Blue Gene system and Blue Gene terminology refer to the Blue Gene documentation listed in "IBM System Blue Gene Solution documentation" on page xv.

Table 37 lists the Blue Gene subtasks with a pointer to the associated instructions:

Table 37. Blue Gene subtasks and associated instructions

Subtask	Associated instructions (see . . .)
Configure LoadLeveler Blue Gene support	“Configuring LoadLeveler Blue Gene support” on page 145
Submit and monitor Blue Gene jobs	“Submitting and monitoring Blue Gene jobs” on page 208

Table 38 lists the Blue Gene related topics and associated information:

Table 38. Blue Gene related topics and associated information

Related topic	Associated information (see . . .)
Configuration file: Blue Gene keywords	“Configuration file keyword descriptions” on page 244
Job command file: Blue Gene keywords	“Job command file keyword descriptions” on page 336
Commands and APIs	Chapter 16, “Commands,” on page 385 or Chapter 17, “Application programming interfaces (APIs),” on page 503
Diagnosis and messages	<i>TWS LoadLeveler: Diagnosis and Messages Guide</i>

Configuring LoadLeveler Blue Gene support

Table 39 lists the subtasks for configuring LoadLeveler Blue Gene support along with a pointer to the associated instructions:

Table 39. Blue Gene configuring subtasks and associated instructions

Subtask	Associated instructions (see . . .)
Configuring LoadLeveler Blue Gene support	“Steps for configuring LoadLeveler Blue Gene support”
Display information about the Blue Gene system	<ul style="list-style-type: none"> Use the llstatus command with the -b option to display information about the Blue Gene system. The llstatus command can also be used with the -B option to display information about Blue Gene base partitions. Using llstatus with the -P option can be used to display information about Blue Gene partitions.
Display information about Blue gene jobs	<ul style="list-style-type: none"> Use the llsummary command with the -I option to display job resource information. Use the llq command with the -b option to display information about all Blue Gene jobs.

Steps for configuring LoadLeveler Blue Gene support

The primary task for configuring LoadLeveler Blue Gene support consists of setting up the environment of the **LoadL_negotiator** daemon, the environment of any process that will run Blue Gene jobs, and the LoadLeveler configuration file.

Perform the following steps to configure LoadLeveler Blue Gene support:

1. Configure the **LoadL_negotiator** daemon to run on a node which has access to the Blue Gene Control System.

LoadLeveler Blue Gene support

2. Enable Blue Gene support by setting the **BG_ENABLED** configuration file keyword to **true**.
3. (Optional) Set any of the following additional Blue Gene related configuration file keywords which your setup requires:
 - **BG_ALLOW_LL_JOBS_ONLY**
 - **BG_CACHE_PARTITIONS**
 - **BG_MIN_PARTITION_SIZE**

See “Configuration file keyword descriptions” on page 244 for more information on these keywords.

4. Set the required environment variables for the **LoadL_negotiator** daemon and any process that will run Blue Gene jobs. You can use global profiles to set the necessary environment variables for all users. Follow these steps to set environment variables for a LoadLeveler daemon:
 - a. Add required environment variable settings to global profile.
 - b. Set the environment as the administrator before invoking **llctl start** on the central manager node.
 - c. Build a shell script which sets the required environments and starts LoadLeveler, which can be invoked using **rsh** remotely.

Note: Using the **llctl -h** or **llctl -g** command to start the central manager remotely will not carry the environment variables from the login session to the LoadLeveler daemons on the remote nodes.

- Specify the full path name of the bridge configuration file by setting the **BRIDGE_CONFIG_FILE** environment variable. For details on the contents of the bridge configuration file, see the *Blue Gene/L: System Administration* book.

Example:

For ksh:

```
export BRIDGE_CONFIG_FILE=/var/bluegene/config/bridge.cfg
```

For csh:

```
setenv BRIDGE_CONFIG_FILE=/var/bluegene/config/bridge.cfg
```

- Specify the full path name of the file containing the data required to access the Blue Gene Control System database by setting the **DB_PROPERTY** environment variable. For details on the contents of the database property file, see the *Blue Gene/L: System Administration* book.

Example:

For ksh:

```
export DB_PROPERTY=/var/bluegene/config/db.cfg
```

For csh:

```
setenv DB_PROPERTY=/var/bluegene/config/db.cfg
```

- Specify the hostname of the machine running the Blue Gene control system by setting the **MMCS_SERVER_IP** environment variable. For details on the use of this environment variable, see the *Blue Gene/L: System Administration* book.

Example:

For ksh:

```
export MMCS_SERVER_IP=bluegene.ibm.com
```

For csh:

```
setenv MMCS_SERVER_IP=bluegene.ibm.com
```

Blue Gene advance reservation support

Advance reservation supports Blue Gene resources including the Blue Gene compute nodes. A front end node (FEN), which is used to start a Blue Gene job is not part of the Blue Gene resources. A Blue Gene reservation only reserves Blue Gene resources and a Blue Gene job step bound to a reservation uses the reserved Blue Gene resources and shares a FEN outside the reservation.

Jobs using Blue Gene resources can be submitted to a Blue Gene reservation to run. A Blue Gene job step can also be used to select what Blue Gene resources to reserve to make sure the reservation will have enough Blue Gene resources to run the Blue Gene job step.

For more information about advance reservations, see “Overview of reservations” on page 23.

Blue Gene fair share scheduling support

Fair share scheduling has been extended to Blue Gene resources as well. The **FAIR_SHARE_TOTAL_SHARES** keyword in **LoadL_config** and the **fair_shares** keyword for the user and group stanza in **LoadL_admin** apply to both the CPU resources and the Blue Gene resources. When a Blue Gene job step ends, both the CPU utilization and the Blue Gene resource utilization data will be collected. The elapsed job running time multiplied by the number of C-nodes allocated to the job step (the **Size Allocated** field in the **llq -l** output) will be counted as the amount of Blue Gene resource used. The used shares of the Blue Gene resources are independent of the used shares of the CPU resources and are made available through the LoadLeveler variables **UserUsedBgShares** and **GroupUsedBgShares**. LoadLeveler variable **JobIsBlueGene** will indicate whether a job step is a Blue Gene job step or not. LoadLeveler administrators have flexibility in specifying the behavior of fair share scheduling by using these variables in the SYSPRIO expression. The **llfs** command and the related APIs can also handle requests related to the Blue Gene resources.

For more information about fair share scheduling, see “Using fair share scheduling” on page 148.

Blue Gene heterogeneous memory support

The LoadLeveler job command file has a **bg_requirements** keyword that can be used to specify the requirements that a Blue Gene base partition must meet to execute the job step. The Blue Gene compute nodes (C-nodes) in the same base partition have the same amount of physical memory. The C-nodes in different base partitions might have different amounts of physical memory. The **bg_requirements** job command file keyword allows users to specify the memory requirement on the Blue Gene C-nodes.

The **bg_requirements** keyword works like the **requirements** keyword, but it can only support memory requirements and applies only to Blue Gene base partitions. For a Blue Gene job step, the **requirements** keyword value applies to the front end node needed by the job step and the **bg_requirements** keyword value applies to the Blue Gene base partitions needed by the job step.

Using fair share scheduling

Fair share scheduling in LoadLeveler provides a way to divide resources in a LoadLeveler cluster among users or groups of users. To fairly share cluster resources, LoadLeveler can be configured to allocate a proportion of the resources to each user or group and to let job priorities be adjusted based on how much of the resources have been used and when they were used. Generally speaking, LoadLeveler should be configured so that job priorities decrease for a user or group that has recently used more resources than the allocated proportion and job priorities should increase for a user or group that has not run any jobs recently.

Administrators can configure the behavior of fair share scheduling through a set of configuration keywords. They can also query fair share information, save a snapshot of historic data, reset and restore fair share scheduling, and perform other functions by using the LoadLeveler `llfs` command, the GUI, and the corresponding APIs.

Fair share scheduling also includes Blue Gene resources (see “Blue Gene fair share scheduling support” on page 147 for more information).

Note: The time of day clocks on all of the nodes in the cluster *must* be synchronized in order for fair share scheduling to work properly.

For more information, see the following:

- “llfs - Fair share scheduling queries and operations” on page 422
- Corresponding APIs:
 - “ll_fair_share subroutine” on page 593
 - “Data access API” on page 522
- Keywords:
 - fair_shares
 - FAIR_SHARE_INTERVAL
 - FAIR_SHARE_INTERVAL
- SYSPRIO expression

Fair share scheduling keywords

The `FAIR_SHARE_TOTAL_SHARES` global configuration file keyword is used to specify the total number of shares that each type of resource is divided into. The `fair_shares` keyword in a user or group stanza in the administration file specifies how many shares the user or group is allocated. The ratio of the `fair_shares` keyword value in a user or group stanza over the `FAIR_SHARE_TOTAL_SHARES` keyword value defines the resource usage proportion for the user or group. For example, if a user is allocated one third of the cluster resources, then the ratio of the user’s `fair_share` value over the `FAIR_SHARE_TOTAL_SHARES` keyword value should be one third.

The LoadLeveler SYSPRIO expression can be configured to let job priorities change to achieve the specified resource usage proportions. Besides changing job priorities, fair share scheduling does not change in any way how LoadLeveler schedules jobs. If a job can be scheduled to run, it will be run regardless of whether the owner and the LoadLeveler group of the job has any shares allocated or not. No matter how many shares are allocated to a user, if the user does not submit any jobs to run, then the resource usage proportion for that user cannot be achieved and other users might be able to use more than their allocated proportions.

Note: The sum of all allocated shares for users or groups does not have to equal the value of the **FAIR_SHARE_TOTAL_SHARES** keyword. The share allocation can be used as a way to prevent a single user from consuming too much of the cluster resources and as a way to share the resources as fairly as possible.

When the value of the **FAIR_SHARE_TOTAL_SHARES** keyword is greater than 0, fair share scheduling is on, which means that resource usage data is collected when every job ends, regardless of the **fair_shares** values for any user or group. The collected usage data is converted to used shares for each user and group. The **llfs** command can be used to display the allocated and used shares. Turning fair share scheduling on does not mean that job priorities are affected by fair share scheduling. You have to configure the **SYSPRIO** expression to let fair share scheduling affect job priorities in a way that suits your needs. By default, the value of the **FAIR_SHARE_TOTAL_SHARES** keyword is 0 and fair share scheduling is disabled.

There is a built-in decay mechanism for the historic resource usage data that is collected when jobs end, that is, the initial resource usage value becomes smaller and smaller as times goes by. This decay mechanism allows the most recent resource usage to have more impact on fair share scheduling. The **FAIR_SHARE_INTERVAL** global configuration file keyword is used to specify how fast the decay is. The shorter the interval, the faster the historic data decays. A resource usage value decays to 5% of its initial value after an elapsed time period of the same length as the **FAIR_SHARE_INTERVAL** value. Generally, the interval should be at least several times larger than the typical job running time in the cluster to get stable results. A value should be chosen corresponding to how long the historic resource usage data should have an impact on the current job priorities.

The LoadLeveler **SYSPRIO** expression is used to calculate job priorities. A set of LoadLeveler variables including some related to fair share scheduling can be used in the **SYSPRIO** expression in the global configuration file. You can define the **SYSPRIO** expression to let fair share scheduling influence the job priorities in a way that is suitable to your needs. For more information, see the **SYSPRIO** expression in Chapter 12, "Configuration file reference."

When the **GroupTotalShares**, **GroupUsedShares**, **UserTotalShares**, **UserUsedShares**, **UserUsedBgShares**, **GroupUsedBgShares**, and **JobIsBlueGene** and their corresponding user-defined variables are used, you must use the **NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL** global configuration keyword to specify a time interval at which the job priorities will be recalculated using the most recent share usage information.

You can add the following user-defined variables to the **LoadL_config** global configuration file to make it easier to specify fair share scheduling in the **SYSPRIO** expressions:

- **GroupRemainingShares** = (GroupTotalShares - GroupUsedShares)
- **GroupHasShares** = (\$(GroupRemainingShares) > 0)
- **GroupSharesExceeded** = (\$(GroupRemainingShares) <= 0)
- **UserRemainingShares** = (UserTotalShares - UserUsedShares)
- **UserHasShares** = (\$(UserRemainingShares) > 0)
- **UserSharesExceeded** = (\$(UserRemainingShares) <= 0)
- **UserRemainingBgShares** = (UserTotalShares - UserUsedBgShares)
- **UserHasBgShares** = (\$(UserRemainingBgShares) > 0)
- **UserBgSharesExceeded** = (\$(UserRemainingBgShares) <= 0)

LoadLeveler fair share scheduling support

- **GroupRemainingBgShares** = (GroupTotalShares - GroupUsedBgShares)
- **GroupHasBgShares** = (\$(GroupRemainingBgShares) > 0)
- **GroupBgSharesExceeded** = (\$(GroupRemainingBgShares) <= 0)
- **JobIsNotBlueGene** = ! JobIsBlueGene

If fair share scheduling is not turned on, either because the **FAIR_SHARE_INTERVAL** keyword value is not positive or because the scheduler type is not **BACKFILL**, then the variables will have the following values:

```
GroupTotalShares: 0
GroupUsedShares: 0
$(GroupRemainingShares): 0
$(GroupHasShares): 0
$(GroupSharesExceeded): 1
UserUsedBgShares: 0
$(UserRemainingBgShares): 0
$(UserHasBgShares): 0
$(UserBgSharesExceeded): 1
```

If a user has the **fair_shares** keyword set to 10 in its user stanza and the user has used up 8 CPU shares and 3 Blue Gene shares, then the variables will have the following values:

```
UserTotalShares: 10
UserUsedShares: 8
$(UserRemainingShares): 2
$(UserHasShares): 1
$(UserSharesExceeded): 0
UserUsedBgShares: 3
$(UserRemainingBgShares): 7
$(UserHasBgShares): 1
$(UserBgSharesExceeded): 0
```

If a group has the **fair_shares** keyword set to 10 in its group stanza and the group has used up 15 CPU shares and 0 Blue Gene shares, then the variables will have the following values:

```
GroupTotalShares: 10
GroupUsedShares: 15
$(GroupRemainingShares): -5
$(GroupHasShares): 0
$(GroupSharesExceeded): 1
GroupUsedBgShares: 0
$(GroupRemainingBgShares): 10
$(GroupHasBgShares): 1
$(GroupBgSharesExceeded): 0
```

The values of the following variables for a Blue Gene job step:

```
JobIsBlueGene: 1
$(JobIsNotBlueGene): 0
```

The values of the following variables for a non-Blue Gene job step:

```
JobIsBlueGene: 0
$(JobIsNotBlueGene): 1
```

Reconfiguring fair share scheduling keywords

LoadLeveler configuration and administration files can be modified to assign new values to various keywords. After files have been modified, issue the **llctl -g reconfig** command to read in the new keyword values. All new keywords introduced for fair share scheduling become effective right after reconfiguration.

Reconfiguring when the Schedd daemons are up

To avoid any inconsistency, change the value of the `FAIR_SHARE_INTERVAL` keyword while the central manager and all **Schedd** daemons are up, then do the reconfiguration. After the reconfiguration, the following will happen:

- All historic fair share scheduling data will be decayed to the current time using the old value.
- The old value is replaced with the new value
- The new value will be used from here on

Notes:

1. You must have the same value for the `FAIR_SHARE_INTERVAL` keyword in the central manager and the **Schedd** daemons because the `FAIR_SHARE_INTERVAL` keyword determines the rate of decay for the historic fair share data and the same value on the daemons maintains the data consistency.
2. There are some LoadLeveler configuration parameters that require restarting LoadLeveler with `llctl recycle` for changes to take effect. You can use `llctl recycle` when changing fair share parameters also. The effect will be the same as using `llctl reconfig` because when the Schedd machine shuts down normally, the fair share scheduling data will be decayed to the time of the shutdown and it will be saved.

Reconfiguring when the Schedd daemons are down

If the value for the `FAIR_SHARE_INTERVAL` keyword has to be changed while a **Schedd** daemon is down, the following will happen when the **Schedd** daemon is restarted:

- All historic fair share scheduling data will be read in from the disk files in the `$(SPOOL)` directory with no change.
- When a new job ends, the historic fair share scheduling data for the owner and the LoadLeveler group of the job will be updated using the new value and then sent to the central manager. The new value is used effectively from the time the data was last updated before the **Schedd** went down, not from the time of the reconfiguration as it would normally be.

Example: three groups share a LoadLeveler cluster

For purposes of this example, we will assume the following:

- Three groups of users share a LoadLeveler cluster and each group is to have one third of the resources
- Historic data will have significant impact for about 10 days
- Groups with unused shares will have much higher job priorities than the groups which have used up their shares

To setup for fair share scheduling with these assumptions, an administrator could update the `LoadL_config` global configuration file as follows:

```
FAIR_SHARE_TOTAL_SHARES = 99

FAIR_SHARE_INTERVAL = 240

NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL = 300

GroupRemainingShares = ( GroupTotalShares - GroupUsedShares )

GroupHasShares = ( $(GroupRemainingShares) > 0 )

SYSPRIO : 10000000 * $(GroupHasShares) - QDate
```

LoadLeveler fair share scheduling support

In the admin file **LoadL_admin**, add:

```
chemistry: type = group

    include_users = harold mark kim enci george charlie

    fair_shares = 33

physics: type = group

    include_users = cnyang gchen newton roy

    fair_shares = 33

math: type = group

    include_users = rich dave chris popco

    fair_shares = 33
```

When user rich in the math group wants to submit a job, the following keyword can be put into the job command file so that the job will have high priority through the math group:

```
#@group=math
```

If user rich has a job that does not need to be run right away or as soon as possible (can be run at any time), then he should run the job in a LoadLeveler group with no shares allocated (for example, the **No_Group** group). Because the group **No_Group** has no shares allocated to it in this example, $$(GroupHasShares)$ has a value of 0 and the job priority will be lower than those jobs whose group has unused shares. The job will be run when all higher priority jobs are done or when it is used to backfill a higher priority job (will be run whenever it can be scheduled).

Example: two thousand students share a LoadLeveler cluster

For purposes of this example, we will assume the following:

- A university has 2000 students who share a LoadLeveler cluster and every student is to have the same number of shares of the resources.
- Historic data will have significant impact for about 7 days (because **FAIR_SHARE_INTERVAL** is not specified and the default value is 7 days).
- A student with unused shares is to have somewhat higher job priorities and let the priorities decrease as the number of used shares increase.

The **LoadL_config** global configuration file should contain the following:

```
FAIR_SHARE_TOTAL_SHARES = 10000

NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL = 600

UserRemainingShares = ( UserTotalShares - UserUsedShares )

SYSPRIO : 100000 * $(UserRemainingShares) - QDate
```

In the **LoadL_admin** admin file, add

```
default: type = user

    fair_shares = 5
```

Note: The value **fair_shares = 5** is the result of dividing the total shares into the number of students ($10000 \div 2000$). The number of students can be more or

less than 2000, but the same configuration parameters still prevent a single user from using too much cluster resources in a short time period.

We can see from the `SYSPRIO` expression that the larger the number of unused shares for a student and the earlier the job is submitted, the higher the priority is for the student's job.

Querying Information about fair share scheduling

The `llfs` command, the GUI, and the data access API can be used to query information about fair share scheduling. The `llfs` command without any options displays the allocated and used shares for all users and LoadLeveler groups having run one or more jobs in the cluster to completion. The `-u` and `-g` options can show the allocated and used shares for any user or LoadLeveler group regardless of whether they have run any jobs in the cluster. In either case, the user or group need not have any `fair_shares` allocated in the `LoadL_admin` administration file for the usage to be reported by the `llfs` command.

Resetting fair share scheduling

The `llfs -r` command option (or the GUI option **Reset historic data**), by default, will start fair share scheduling from the beginning, which means that all the previous historic data will be lost. This command will not be run unless all `Schedd` daemons are up and running.

In case a `Schedd` daemon is down when this command option is being run, the request will not be processed. To manually reset fair share scheduling, bring down the LoadLeveler cluster, remove all fair share data files (`fair_share_queue.dir` and `fair_share_queue.pag`) in the `$(SPOOL)` directory and then restart the LoadLeveler cluster.

Saving historic data

The LoadLeveler central manager holds the complete historic fair share data when it is up. Every `Schedd` holds a portion of the historic fair share data and the data is stored on disk in the `$(SPOOL)` directory. When the central manager is restarted, it receives the historic fair share data from every `Schedd`. If a `Schedd` machine is down temporarily and the central manager remains up, the data in the central manager is not affected. In case a `Schedd` machine is permanently damaged and the central manager restarts, the central manager will not be able to get all of the historic fair share data because the data stored on the damaged `Schedd` is lost. If the value of `FAIR_SHARE_INTERVAL` is very large, many days of data on the damaged `Schedd` could be lost. To reduce the loss of data, the historic fair share data in the central manager can be saved to disk periodically. Recovery can be done using the latest saved data when a `Schedd` machine is permanently out of service. The `llfs -s` command, the GUI, or the `ll_fair_share` API can be used to save a snapshot of the historic data in the central manager to a file.

Restoring saved historic data

You can use the `llfs -r` command option, the GUI, or the `ll_fair_share` API to restore fair share scheduling to a previously saved state. For the file name, specify a file you saved previously using `llfs -s`.

If the central manager goes down and restarts again, the historic data stored in an out of service `Schedd` machine is not reported to the central manager. If the `Schedd`

LoadLeveler fair share scheduling support

machine will not be brought back to service at all, then the administrator can consider restoring fair share scheduling to a state corresponding to the latest saved file.

Procedure for recovering a job spool

The **llmovespool** command is intended for recovery purposes only. Jobs being managed by a down Schedd are unable to clean up resources or move to completion. These jobs need their job records transferred to another Schedd. The **llmovespool** command moves the job records from the spool of one managing Schedd to another managing Schedd in the local cluster. All moved jobs retain their original job identifiers.

It is very important that the Schedd that created the job records to be moved is not running during the move operation. Jobs within the job queue database will be unrecoverable if the job queue is updated during the move by any process other than the **llmovespool** command.

The **llmovespool** command operates on a set of job records, these records are updated as the command executes. When a job is successfully moved, the records for that job are deleted. Job records that are not moved because of a recoverable failure, like the original Schedd not being fenced, may have the **llmovespool** command executed against them again. It is very important that a Schedd never reads the job records from the spool being moved. Jobs will be unrecoverable if more than one Schedd is considered to be the managing Schedd.

The procedure for recovering a job spool is:

1. Move the files located in the spool directory to be transferred to another directory before entering the **llmovespool** command in order to guarantee that no other Schedd process is updating the job records.
2. Add the statement **schedd_fenced=true** to the machine stanza of the original Schedd node in order to guarantee that the central manager ignores connections from the original managing Schedd, and to prevent conflicts from arising if the original Schedd is restarted after the **llmovespool** command has been run. See the **schedd_fenced=true** keyword in Chapter 13, "Administration file reference," on page 301 for more information.
3. Reconfigure the central manager node so that it recognizes that the original Schedd is "fenced".
4. Issue the **llmovespool** command providing the spool directory where the job records are stored. The command displays a message that the transfer has started and reports status for each job as it is processed. For more information about the **llmovespool** command, see "llmovespool - Move job records" on page 442. For more information about the **ll_move_spool** API, see "ll_move_spool subroutine" on page 634.

Chapter 7. Using LoadLeveler's GUI to perform administrator tasks

The end user can perform many tasks more efficiently and faster using the graphical user interface (GUI) but there are certain tasks that end users cannot perform unless they have the proper authority. If you are defined as a LoadLeveler administrator in the LoadLeveler configuration file then you are immediately granted administrative authority and can perform the administrative tasks discussed in this topic. To find out how to grant someone administrative authority, see "Defining LoadLeveler administrators" on page 41.

You can access LoadLeveler administrative commands using the **Admin** pull-down menu on both the Jobs window and the Machines window of the GUI. The **Admin** pull-down menu on the Jobs window corresponds to the command options available in the **llhold**, **llfavoruser**, and **llfavorjob** commands. The **Admin** pull-down menu on the Machines window corresponds to the command options available in the **llctl** command.

The main window of the GUI has three sub-windows: one for job status with pull-down menus for job-related commands, one for machine status with pull-down menus for machine-related commands, and one for messages and logs (see "The LoadLeveler main window" on page 378 in the Chapter 15, "Graphical user interface (GUI) reference," on page 377). There are a variety of facilities available that allow you to sort and select the items displayed.

Job-related administrative actions

You access the administrative commands that act on jobs through the **Admin** pull-down menu in the Jobs window of the GUI.

You can perform the following tasks with this menu:

Favor Users Allows you to favor users. This means that you can select one or more users whose jobs you want to move up in the job queue. This corresponds to the **llfavoruser** command.

Select **Admin** from the Jobs window

Select **Favor User**

▲The **Order by User** window appears.

Type in

The name of the user whose jobs you want to favor.

Press **OK**

Unfavor Users

Allows you to unfavor users. This means that you want to unfavor the user's jobs which you previously favored. This corresponds to the **llfavoruser** command.

Select **Admin** from the Jobs window

Select **Unfavor User**

▲The **Order by User** window appears.

Administrative uses of the GUI

- Type in**
The name of the user for whom you want to unfavor their jobs.
- Press** OK
- Favor Jobs** Allows you to select a job that you want to favor. This corresponds to the **llfavorjob** command.
- Select** One or more jobs from the Jobs window
- Select** **Admin** from the Jobs window
- Select** **Favor Job**
- ▲The selected jobs are favored.
- Press** OK
- Unfavor Jobs** Allows you select a job that you want to unfavor. This corresponds to the **llfavorjob** command.
- Select** One or more jobs from the Jobs window
- Select** **Admin** from the Jobs window
- Select** **Unfavor Job**
- ▲Unfavors the jobs that you previously selected.
- Syshold** Allows you to place a system hold on a job. This corresponds to the **llhold** command.
- Select** A job from the Jobs window
- Select** **Admin** pull-down menu from the Jobs window
- Select** **Syshold** to place a system hold on the job.
- Release From Hold** Allows you to release the system hold on a job. This corresponds to the **llhold** command.
- Select** A job from the Jobs window
- Select** **Admin** pull-down menu from the Jobs window
- Select** **Release From Hold** to release the system hold on the job.
- Preempt** Available when using the BACKFILL or external schedulers. Preempt allows you to place the selected jobs in preempted state. This action corresponds to the **llpreempt** command.
- Select** One or more jobs from the Jobs window
- Select** **Admin** pull-down menu from the Jobs window
- Select** **Preempt**
- Resume Preempted Job** Available only when using the BACKFILL or external schedulers. Resume Preempted Job allows you to remove user-initiated preemption (initiated using the Preempt menu option or the **llpreempt** command) from the selected jobs. This action corresponds to the **llpreempt -r** command.
- Select** One or more jobs from the Jobs window
- Select** **Admin** pull-down menu from the Jobs window

Select Resume Preempted Job

Prevent Preempt

Available only when using the BACKFILL or API scheduler. Prevent Preempt allows you to place the selected running job into a non-preemptable state. When the BACKFILL or API scheduler is in use, this is equivalent to the **lmodify -p nopreempt** command.

Select One job from the Jobs window

Select Admin pull-down menu from the Jobs window

Select Prevent Preempt

Allow Preempt

Available only when using the BACKFILL or API scheduler, Allow Preempt makes the unpreemptable job preemptable again. When the BACKFILL or API scheduler is in use, this is equivalent to the **lmodify -p preempt** command.

Select One or more jobs from the Jobs window

Select Admin pull-down menu from the Jobs window

Select Allow Preempt

Extend Wallclock Limits

Allows you to extend the wallclock limits by the number of minutes specified. This corresponds to the **lmodify -W** command.

Select Admin pull-down window from the Jobs window

Select Extend Wallclock Limit

▲The Extend Wallclock Limits window appears.

Type in

The number of minutes to extend the wallclock limit.

Press OK

Modify Job Priority

Allows you to modify the system priority of a job step. This corresponds to the **lmodify -s** command.

Select Admin pull-down window from the Jobs window

Select Modify Job Priority

▲The Modify Job Priority window appears.

Type in

An integer value for system priority.

Press OK

Move to another cluster

Allows you to move an idle job from the local cluster to another. This menu items appears only when a multicluster environment is configured. It corresponds to the **lmovejob** command.

Select Admin pull-down window from the Jobs window

Select Modify Job Priority

▲The Move Job to Another Cluster window appears.

Select The name of the target cluster.

Press OK

Machine-related administrative actions

You access the administrative commands that act on machines using the **Admin** pull-down menu in the Machines window of the GUI.

Using the GUI pull-down menu, you can perform the tasks described in this topic.

- Start All** Starts LoadLeveler on all machines listed in machine stanzas beginning with the central manager. Submit-only machines are skipped. Use this option when specifying alternate central managers in order to ensure the primary central manager starts before any alternate central manager attempts to serve as central manager.
- Select** **Admin** from the Machines window.
- Select** **Start All**
- Start LoadLeveler** Allows you to start LoadLeveler on selected machines.
- Select** One or more machines on which you want to start LoadLeveler.
- Select** **Admin** from the Machines window.
- Select** **Start LoadLeveler**
- Start Drained** Allows you to start LoadLeveler with **startd** drained on selected machines.
- Select** One or more machines on which you want **startd** drained.
- Select** **Admin** from the Machines window.
- Select** **Start Drained**
- Stop LoadLeveler** Allows you to stop LoadLeveler on selected machines.
- Select** One or more machines on which you want to stop LoadLeveler.
- Select** **Admin** from the Machines window.
- Select** **Stop LoadLeveler**.
- Stop All** Stops LoadLeveler on all machines listed in machine stanzas. Submit-only machines are skipped.
- Select** **Admin** from the Machines window.
- Select** **Stop All**
- Reconfig** Forces all daemons to reread the configuration files
- Select** The machine on which you want to operate. To reconfigure this **xloadl** session, choose **reconfig** but do not select a machine.
- Select** **Admin** from the Machines window.
- Select** **reconfig**
- Recycle** Stops all LoadLeveler daemons and restarts them.

Select The machine on which you want to operate.

Select **Admin** from the Machines window.

Select **recycle**

Configuration Tasks

Starts Configuration Tasks wizard

Select **Admin** from the Machines window.

Select **Config Tasks**

Note: Use the invoking script **lltg** to start the wizard outside of **xloadl**. This option will appear on the pull-down only if the **LoadL.tguides** filesset is installed.

Drain

Allows no more LoadLeveler jobs to begin running on this machine but it does allow running jobs to complete.

Select The machine on which you want to operate.

Select **Admin** from the Machines window.

Select **drain**.

A cascading menu allows you to select either **daemons**, **Schedd**, **startd**, or **startd by class**. If you select **daemons**, both the **startd** and the **Schedd** on the selected machine will be drained. If you select **Schedd**, only the **Schedd** on the selected machine will be drained. If you select **startd**, only the **startd** on the selected machine will be drained. If you select **startd by class**, a window appears which allows you to select classes to be drained.

Flush

Terminates running jobs on this host and sends them back to the system queue to await redispach. No new jobs are redispached to this machine until **resume** is issued. Forces a checkpoint if jobs are enabled for checkpointing.

Select The machine on which you want to operate.

Select **Admin** from the Machines window.

Select **flush**

Suspend

Suspends all jobs on this host.

Select The machine on which you want to operate.

Select **Admin** from the Machines window.

Select **suspend**

Resume

Resumes all jobs on this machine.

Select The machine on which you want to operate.

Select **Admin** from the Machines window

Select **resume**

A cascading menu allows you to select either **daemons**, **Schedd**, **startd**, or **startd by class**. If you select **daemons**, both machines will be resumed. If you select **Schedd**, only the **Schedd** on the selected machine will be resumed. If you select **startd**, only the **startd** on the selected machine will

Administrative uses of the GUI

be resumed. If you select **startd by class**, a window appears which allows you to select classes to be resumed.

Purge Schedd Allows you to purge (remove) all of the jobs scheduled by the Schedd on the selected machines. To use this option you must first specify `schedd_fenced=true` in the machine stanza for this machine and reconfigure the central manager. For more information on using this option, see “How do I recover resources allocated by a Schedd machine?” on page 660.

Select One or more machines whose Schedd is down and will be down long enough to necessitate that you recover the resources allocated to jobs scheduled by that Schedd

Select **Admin** pull-down menu from the Machines window

Select
Purge Schedd

Press **OK**

Capture Data Collects information on the machines selected.

Select The machine on which you want to operate.

Select **Admin** from the Machines window.

Select **Capture Data**.

Collect Account Data

Collects accounting data on the machines selected.

Select The machine on which you want to operate.

Select **Admin** from the Machines window.

Select **Collect Account Data**.

A window appears prompting you to enter the name of the directory in which you want the collected data stored.

Collect Reservation Data

Collects reservation data on the machines selected.

Select The machine on which you want to operate.

Select **Admin** from the Machines window.

Select **Collect Reservation Data**.

A window appears prompting you to enter the name of the directory in which you want the collected data stored.

Create Account Report

Creates an accounting report for you.

Select **Admin → Create Account Report...**

Note: If you want to receive an extended accounting report, select the **extended** cascading button.

A window appears prompting you to enter the following information:

- A short, long, or extended version of the output. The short version is the default.
- The user ID
- The class name

- The LoadL (LoadLeveler) group name
- The UNIX group name
- The Allocated host
- The job ID
- The report Type
- The section
- A start and end date for the report. If no date is specified, the default is to report all of the data in the report.
- The name of the input data file.
- The name of the output data file. This is the same as stdout.

Press **OK**

The window closes and you return to the main window. The report appears in the Messages window if no output data file was specified.

Move Spool Moves the job records from the spool of one managing Schedd to another managing Schedd in the local cluster. This is intended for recovery purposes only.

Select One Schedd machine from the Machines window.

Select **Admin** from the Machines window.

Select **Move Spool**

A window is displayed prompting you to enter the directory containing the job records to be moved.

Press **OK**

Version Displays version and release data for LoadLeveler on the machines selected in an information window.

Select The machine on which you want to operate.

Select **Admin** from the Machines window.

Select **version**

Fair Share Scheduling

Provides fair share scheduling functions (see “llfs - Fair share scheduling queries and operations” on page 422).

Select **Admin** from the Machines window.

Select **Fair Share Scheduling**

A cascading menu allows you to select one of the following:

- **Show**

Displays fair share scheduling information for all users or for specified users and groups.

- **Save historic data**

Saves fair share scheduling information into the directory specified.

- **Restore historic data**

Restores fair share scheduling data to a state corresponding to a file previously saved by Save historic data or the **llfs -s** command.

Administrative uses of the GUI

- **Reset historic data**
Erases all historic CPU data to reset fair share scheduling.

Part 3. Submitting and managing TWS LoadLeveler jobs

After an administrator installs IBM Tivoli Workload Scheduler (TWS) LoadLeveler and customizes the environment, general users can build and submit jobs to exploit the many features of the TWS LoadLeveler runtime environment.

The topics listed Table 40 will help you learn about building and submitting jobs:

Table 40. Learning about building and submitting jobs

To learn about:	Read the following:
Creating and submitting serial and parallel jobs	Chapter 8, "Building and submitting jobs," on page 165
Controlling and monitoring TWS LoadLeveler jobs	Chapter 9, "Managing submitted jobs," on page 211
Ways to control or monitor TWS LoadLeveler operations by using the TWS LoadLeveler commands, GUI, and APIs	<ul style="list-style-type: none">• Chapter 16, "Commands," on page 385• Chapter 10, "Example: Using commands to build, submit, and manage jobs," on page 215• Chapter 11, "Using LoadLeveler's GUI to build, submit, and manage jobs," on page 217• Chapter 17, "Application programming interfaces (APIs)," on page 503

Chapter 8. Building and submitting jobs

Table 41 lists the tasks that general users perform to run LoadLeveler jobs.

Table 41. Roadmap of user tasks for building and submitting jobs

To learn about:	Read the following:
Building jobs	<ul style="list-style-type: none">• “Building a job command file”• “Editing job command files” on page 171• “Defining resources for a job step” on page 171• “Working with coscheduled job steps” on page 171• “Using bulk data transfer” on page 173• “Preparing a job for checkpoint/restart” on page 174• “Preparing a job for preemption” on page 177
Submitting jobs	<ul style="list-style-type: none">• “Submitting a job command file” on page 177• “llsubmit - Submit a job” on page 494
Working with parallel jobs	“Working with parallel jobs” on page 178
Working with reserved node resources and the jobs that use them	“Working with reservations” on page 197
Correctly specifying job command file keywords	Chapter 14, “Job command file reference,” on page 333

Building a job command file

Before you can submit a job or perform any other job related tasks, you need to build a job command file. A job command file describes the job you want to submit, and can include LoadLeveler keyword statements. For example, to specify a binary to be executed, you can use the **executable** keyword, which is described later in this topic. To specify a shell script to be executed, the **executable** keyword can be used; if it is not used, LoadLeveler assumes that the job command file itself is the executable.

The job command file can include the following:

- LoadLeveler keyword statements: A *keyword* is a word that can appear in job command files. A *keyword statement* is a statement that begins with a LoadLeveler keyword. These keywords are described in “Job command file keyword descriptions” on page 336.
- Comment statements: You can use comments to document your job command files. You can add comment lines to the file as you would in a shell script.
- Shell command statements: If you use a shell script as the executable, the job command file can include shell commands.
- LoadLeveler variables: See “Job command file variables” on page 372 for more information.

You can build a job command file either by using the Build a Job window on the GUI or by using a text editor.

Using multiple steps in a job command file

To specify a stream of job steps, you need to list each job step in the job command file. You must specify one **queue** statement for each job step. Also, the executables for all job steps in the job command file must exist when you submit the job. For most keywords, if you specify the keyword in a job step of a multi-step job, its value is inherited by all proceeding job steps. Exceptions to this are noted in the keyword description.

LoadLeveler treats all job steps as independent job steps unless you use the **dependency** keyword. If you use the **dependency** keyword, LoadLeveler determines whether a job step should run based upon the exit status of the previously run job step.

For example, Figure 19 contains two separate job steps. Notice that step1 is the first job step to run and that step2 is a job step that runs only if step1 exits with the correct exit status.

```
# This job command file lists two job steps called "step1"
# and "step2". "step2" only runs if "step1" completes
# with exit status = 0. Each job step requires a new
# queue statement.
#
# @ step_name = step1
# @ executable = executable1
# @ input = step1.in1
# @ output = step1.out1
# @ error = step2.err1
# @ queue
# @ dependency = (step1 == 0)
# @ step_name = step2
# @ executable = executable2
# @ input = step2.in1
# @ output = step2.out1
# @ error = step2.err1
# @ queue
```

Figure 19. Job command file with multiple steps

In Figure 19, step1 is called the *sustaining* job step. step2 is called the *dependent* job step because whether or not it begins to run is dependent upon the exit status of step1. A single sustaining job step can have more than one dependent job steps and a dependent job step can also have job steps dependent upon it.

In Figure 19, each job step has its own **executable**, **input**, **output**, and **error** statements. Your job steps can have their own separate statements, or they can use those statements defined in a previous job step. For example, in Figure 20 on page 167, step2 uses the **executable** statement defined in step1:

```

# This job command file uses only one executable for
# both job steps.
#
# @ step_name = step1
# @ executable = executable1
# @ input = step1.in1
# @ output = step1.out1
# @ error = step1.err1
# @ queue
# @ dependency = (step1 == 0)
# @ step_name = step2
# @ input = step2.in1
# @ output = step2.out1
# @ error = step2.err1
# @ queue

```

Figure 20. Job command file with multiple steps and one executable

Examples: Job command files

- **Example 1: Generating multiple jobs with varying outputs**

To run a program several times, varying the initial conditions each time, you could can multiple LoadLeveler scripts, each specifying a different input and output file as described in Figure 22 on page 169. It would probably be more convenient to prepare different input files and submit the job only once, letting LoadLeveler generate the output files and do the multiple submissions for you.

Figure 21 illustrates the following:

- You can refer to the LoadLeveler name of your job symbolically, using **\$(jobid)** and **\$(stepid)** in the LoadLeveler script file.
- **\$(jobid)** refers to the job identifier.
- **\$(stepid)** refers to the job step identifier and increases after each **queue** command. Therefore, you only need to specify input, output, and error statements once to have LoadLeveler name these files correctly.

Assume that you created five input files and each input file has different initial conditions for the program. The names of the input files are in the form **longjob.in.x**, where *x* is 0–4.

Submitting the LoadLeveler script shown in Figure 21 results in your program running five times, each time with a different input file. LoadLeveler generates the output file from the LoadLeveler job step IDs. This ensures that the results from the different submissions are not merged.

```

# @ executable = longjob
# @ input = longjob.in.$(stepid)
# @ output = longjob.out.$(jobid).$(stepid)
# @ error = longjob.err.$(jobid).$(stepid)
# @ queue

```

Figure 21. Job command file with varying input statements

To submit the job, type the command:

```
llsubmit longjob.cmd
```

LoadLeveler responds by issuing the following:

Building a job command file

submit: The job "ll6.23" with 5 job steps has been submitted.

Table 42 lists the standard input files, standard output files, and standard error files for the five job steps:

Table 42. Standard files for the five job steps

Job Step	Standard Input	Standard Output	Standard Error
ll6.23.0	longjob.in.0	longjob.out.23.0	longjob.err.23.0
ll6.23.1	longjob.in.1	longjob.out.23.1	longjob.err.23.1
ll6.23.2	longjob.in.2	longjob.out.23.2	longjob.err.23.2
ll6.23.3	longjob.in.3	longjob.out.23.3	longjob.err.23.3
ll6.23.4	longjob.in.4	longjob.out.23.4	longjob.err.23.4

• Example 2: Using LoadLeveler variables in a job command file

Figure 22 on page 169 shows how you can use LoadLeveler variables in a job command file to assign different names to input and output files. This example assumes the following:

- The name of the machine from which the job is submitted is `lltest1`
- The user's home directory is `/u/rhclark` and the current working directory is `/u/rhclark/OSL`
- LoadLeveler assigns a value of 122 to `$(jobid)`.

In Job Step 0:

- LoadLeveler creates the subdirectories `oslsslv_out` and `oslsslv_err` if they do not exist at the time the job step is started.

In Job Step 1:

- The character string `rhclark` denotes the home directory of user `rhclark` in `input`, `output`, `error`, and `executable` statements.
- The `$(base_executable)` variable is set to be the "base" portion of the `executable`, which is `oslsslv`.
- The `$(host)` variable is equivalent to `$(hostname)`. Similarly, `$(jobid)` and `$(stepid)` are equivalent to `$(cluster)` and `$(process)`, respectively.

In Job Step 2:

- This job step is executed only if the return codes from Step 0 and Step 1 are both equal to zero.
- The initial working directory for Step 2 is explicitly specified.

```

# Job step 0 =====
# The names of the output and error files created by this job step are:
#
#   output: /u/rhclark/OSL/oslsslv_out/11test1.122.0.out
#   error : /u/rhclark/OSL/oslsslv_err/11test1_122_0_err
#
# @ job_name = OSL
# @ step_name = step_0
# @ executable = oslsslv
# @ arguments = -maxmin=min -scale=yes -alg=dual
# @ environment = OSL_ENV1=20000; OSL_ENV2=500000
# @ requirements = (Arch == "R6000") && (OpSys == "AIX53")
# @ input = test01.mps.$(stepid)
# @ output = $(executable)_out/$(host).$(jobid).$(stepid).out
# @ error = $(executable)_err/$(host)_$(jobid)_$(stepid)_err
# @ queue
#
# Job step 1 =====
# The names of the output and error files created by this job step are:
#
#   output: /u/rhclark/OSL/oslsslv_out/11test1.122.1.out
#   error : /u/rhclark/OSL/oslsslv_err/11test1_122_1_err
#
# @ step_name = step_1
# @ executable = rhclark/$(job_name)/oslsslv
# @ arguments = -maxmin=max -scale=no -alg=primal
# @ environment = OSL_ENV1=60000; OSL_ENV2=500000; \
#               OSL_ENV3=70000; OSL_ENV4=800000;
# @ input = rhclark/$(job_name)/test01.mps.$(stepid)
# @ output = rhclark/$(job_name)/$(base_executable)_out/$(hostname).$(cluster).$(process).out
# @ error = rhclark/$(job_name)/$(base_executable)_err/$(hostname)_$(cluster)_$(process)_err
# @ queue
#
# Job step 2 =====
# The names of the output and error files created by this job step are:
#
#   output: /u/rhclark/OSL/oslsslv_out/11test1.122.2.out
#   error : /u/rhclark/OSL/oslsslv_err/11test1_122_2_err
#
# @ step_name = OSL
# @ dependency = (step_0 == 0) && (step_1 == 0)
# @ comment = oslsslv
# @ initialdir = /u/rhclark/$(step_name)
# @ arguments = -maxmin=min -scale=yes -alg=dual
# @ environment = OSL_ENV1=300000; OSL_ENV2=500000
# @ input = test01.mps.$(stepid)
# @ output = $(comment)_out/$(host).$(jobid).$(stepid).out
# @ error = $(comment)_err/$(host)_$(jobid)_$(stepid)_err
# @ queue

```

Figure 22. Using LoadLeveler variables in a job command file

• Example 3: Using the job command file as the executable

The name of the sample script shown in Figure 23 on page 170 is `run_spice_job`. This script illustrates the following:

- The script does not contain the **executable** keyword. When you do not use this keyword, LoadLeveler assumes that the script is the executable. (Since the name of the script is `run_spice_job`, you can add the **executable = run_spice_job** statement to the script, but it is not necessary.)
- The job consists of four job steps (there are 4 **queue** statements). The **spice3f5** and **spice2g6** programs are invoked at each job step using different input data files:
 - **spice3f5**: Input for this program is from the file `spice3f5_input_x` where `x` has a value of 0, 1, and 2 for job steps 0, 1, and 2, respectively. The name of this file is passed as the first argument to the script. Standard output and standard error data generated by **spice3f5** are directed to the file `spice3f5_output_x`. The name of this file is passed as second argument to

Building a job command file

the script. In job step 3, the names of the input and output files are **spice3f5_input_benchmark1** and **spice3f5_output_benchmark1**, respectively.

- **spice2g6**: Input for this program is from the file **spice2g6_input_x**. Standard output and standard error data generated by **spice2g6** together with all other standard output and standard error data generated by this script are directed to the files **spice_test_output_x** and **spice_test_error_x**, respectively. In job step 3, the name of the input file is **spice2g6_input_benchmark1**. The standard output and standard error files are **spice_test_output_benchmark1** and **spice_test_error_benchmark1**.

All file names that are not fully qualified are relative to the initial working directory **/home/load1/spice**. LoadLeveler will send the job steps 0 and 1 of this job to a machine for that has a real memory of 64 MB or more for execution. Job step 2 most likely will be sent to a machine that has more than 128 MB of real memory and has the ESSL library installed since these preferences have been stated using the LoadLeveler **preferences** keyword. LoadLeveler will send job step 3 to the machine **115.pok.ibm.com** for execution because of the explicit requirement for this machine in the **requirements** statement.

```
#!/bin/ksh
# @ job_name = spice_test
# @ account_no = 99999
# @ class = small
# @ arguments = spice3f5_input_${stepid} spice3f5_output_${stepid}
# @ input = spice2g6_input_${stepid}
# @ output = ${job_name}_output_${stepid}
# @ error = ${job_name}_error_${stepid}
# @ initialdir = /home/load1/spice
# @ requirements = ((Arch == "R6000") && \
#                 (OpSys == "AIX53") && (Memory > 64))
# @ queue
# @ queue
# @ preferences = ((Memory > 128) && (Feature == "ESSL"))
# @ queue
# @ class = large
# @ arguments = spice3f5_input_benchmark1 spice3f5_output_benchmark1
# @ requirements = (Machine == "115.pok.ibm.com")
# @ input = spice2g6_input_benchmark1
# @ output = ${job_name}_output_benchmark1
# @ error = ${job_name}_error_benchmark1
# @ queue
OS_NAME=`uname`

case $OS_NAME in
  AIX)
    echo "Running $OS_NAME version of spice3f5" > $2
    AIX_bin/spice3f5 < $1 >> $2 2>&1
    echo "Running $OS_NAME version of spice2g6"
    AIX_bin/spice2g6
    ;;
  *)
    echo "spice3f5 for $OS_NAME is not available" > $2
    echo "spice2g6 for $OS_NAME is not available"
    ;;
esac
```

Figure 23. Job command file used as the executable

Editing job command files

After you build a job command file, you can edit it using the editor of your choice. You may want to change the name of the executable or add or delete some statements.

When you create a job command file, it is considered the job executable unless you specify otherwise by using the **executable** keyword in the job command file. LoadLeveler copies the executable to the spool directory unless the **checkpoint** keyword was set to **yes** or **interval**. Jobs that are to be checkpointed cannot be moved to the spool directory. Do not make any changes to the executable while the job is still in the queue—it could affect the way that job runs.

Defining resources for a job step

The LoadLeveler user may use the **resources** keyword in the job command file to specify the resources to be consumed by each task of a job step. If the **resources** keyword is specified in the job command file, it overrides any **default_resources** specified by the administrator for the job step's class.

For example, the following job requests one CPU and one FRM license for each of its tasks:

```
resources = ConsumableCpus(1) FRMLicense(1)
```

If this were specified in a serial job step, one CPU and one FRM license would be consumed while the job step runs. If this were a parallel job step, then the number of CPUs and FRM licenses consumed while the job step runs would depend upon how many tasks were running on each machine. For more information on assigning tasks to nodes, see "Task-assignment considerations" on page 180.

Working with coscheduled job steps

LoadLeveler allows you to specify that a group of two or more steps within a job are to be coscheduled. Coscheduled steps are dispatched at the same time.

Submitting coscheduled job steps

The **coschedule = true** keyword has been added in the job command file to specify which steps within a job are to be coscheduled. All steps within a job with the **coschedule** keyword set to true will be coscheduled. The coscheduled steps will continue to be stored as individual steps in both memory and in the job queue, but when performing certain operations, such as scheduling, the steps will be managed as a single entity. An operation initiated on one of the coscheduled steps will cause the operation to be performed on all other steps (unless the coscheduling dependency between steps is broken).

Determining priority for coscheduled job steps

Coscheduled steps are supported only with the BACKFILL scheduler. The LoadLeveler BACKFILL scheduler will only dispatch the set of coscheduled steps when enough resource is available for all steps in the set to start. If the set of coscheduled steps cannot be started immediately, but enough resource will be available in the future, then the resource for all the steps will be reserved. In this case, only one of the coscheduled steps will be designated as a top dog, but enough resources will be reserved for all coscheduled steps and all the steps will be dispatched when the top dog step is started. The coscheduled step with the

Building a job command file

highest priority in the current job queue will be designated as the primary coscheduled step and all other steps will be secondary coscheduled steps. The primary coscheduled step will determine when the set of coscheduled steps will be scheduled. The priority for all other coscheduled steps is ignored.

Supporting preemption of coscheduled job steps

Preemption of coscheduled steps is supported with the following restrictions:

- In order for a step S to be preemptable by a coscheduled step, all steps in the set of coscheduled steps must be able to preempt step S.
- In order for a step S to preempt a coscheduled step, all steps in the set of coscheduled steps must be preemptable by step S.
- The set of job steps available for preemption will be the same for all coscheduled steps. Any resource made available by preemption for one coscheduled step will be available to all other coscheduled steps.

To determine the preempt type and preempt method to use when a coscheduled step preempts another step, an order of precedence for preempt types and preempt methods has been defined. All steps in the preempting coscheduled step are examined and the preempt type and preempt method having the highest precedence are used. The order of precedence for preempt type will be **ALL** and **ENOUGH**. The precedence order for preempt method is:

- Remove
- Vacate
- System Hold
- User hold
- Suspend

For more information about preempt types and methods, see “Planning to preempt jobs” on page 117.

When coscheduled steps are running, if one step is preempted as a result of a system-initiated preemption, then all coscheduled steps are preempted. When determining an optimal preempt set, the BACKFILL scheduler does not consider coscheduled steps as a single entity. All coscheduled steps are in the initial preempt set, but the final preempt set might not include all coscheduled steps, if the scheduler determines the resources of some coscheduled steps are not necessary to start the preempting job step. This implies that more resource than necessary might be preempted when a coscheduled step is in the set of steps to be preempted because regardless of whether or not all coscheduled steps are in the preempt set, if one coscheduled step is preempted, then all coscheduled steps will be preempted.

Coscheduled job steps and commands and APIs

Commands and APIs that operate on job steps are impacted by coscheduled steps. For the **llbind**, **llcancel**, **llhold**, and **llpreempt** commands, even if all coscheduled steps are not in the list of targeted steps, the requested operation is performed on all coscheduled steps.

For the **llmkres** and **llchres** commands, a coscheduled job step cannot be specified when using the **-j** or **-f** flags. For the **llckpt** command, you cannot specify a coscheduled job step using the **-u** flag.

Termination of coscheduled steps

If a coscheduled step is dispatched but cannot be started and is rejected by the **startd** daemon or the starter process, then all coscheduled steps are rejected. If a running step is removed or vacated by LoadLeveler as a result of a system related failure, then all coscheduled steps are removed or vacated. If a running step is vacated as a result of the VACATE expression evaluating to true for the step, then all coscheduled steps are vacated.

Using bulk data transfer

On AIX systems with device drivers and network adapters that support remote direct-memory access (RDMA), LoadLeveler supports bulk data transfer for jobs that use either the Internet or User Space communication protocol mode. For jobs using the Internet protocol (IP jobs), LoadLeveler does not monitor or control the use of bulk transfer. For User Space jobs that request bulk transfer, however, LoadLeveler creates a consumable RDMA resource requirement. Machines with network adapters that support RDMA are automatically given an RDMA consumable resource with an available amount of four. Each step that requests bulk transfer consumes one RDMA resource on each machine on which that step runs.

The RDMA resource is similar to user-defined consumable resources except in one important way: A user-specified resource requirement is consumed by every task of the job assigned to a machine, whereas the RDMA resource is consumed once on a machine no matter how many tasks of the job are running on the machine. Other than that exception, LoadLeveler handles the RDMA resource as it does all other consumable resources. LoadLeveler displays RDMA resources in the output of the following commands:

- `llq -l`
- `llstatus -l`
- `llstatus -R`
- `llsummary -l`

Bulk transfer is supported only on systems where the device driver of the network adapters supports RDMA. To determine which systems will support bulk transfer, use the `llstatus` command with either the `-l` or `-R` flag to display machines with adapters that support RDMA (supporting machines will have an RDMA resource listed in the command output).

Under certain conditions, LoadLeveler displays a total count of RDMA resources as less than four:

- If jobs that LoadLeveler does not manage use RDMA, the amount of available RDMA resource reported to the Negotiator is reduced by the amount consumed by the unmanaged jobs.
- In rare situations, LoadLeveler jobs can fail to release their adapter resources before reporting to the Negotiator that they have completed. In these situations, the amount of available RDMA reported to the Negotiator is reduced by the amount consumed by the unreleased adapter resources. When the adapter resources are eventually released, the RDMA resource they consumed becomes available again.

These conditions do not require corrective action.

You do not need to perform specific job-definition tasks to enable bulk transfer for LoadLeveler jobs that use the IP network protocol. LoadLeveler cannot affect

Using bulk data transfer

whether IP communication uses bulk transfer; the implementation of IP where the job runs determines whether bulk transfer is supported.

To enable User Space jobs to use bulk data transfer, however, **all** of the following tasks must be completed. If you omit one or more of these steps, the job will run but will not be able to use bulk transfer.

- A LoadLeveler administrator must update the LoadLeveler configuration file to include the value `RDMA` in the `SCHEDULE_BY_RESOURCES` list.

Example:

```
SCHEDULE_BY_RESOURCES = RDMA others
```

- Users must request bulk transfer for their LoadLeveler jobs, using one of the following methods:

- Specifying the `bulkxfer` keyword in the LoadLeveler job command file.

Example:

```
#@ bulkxfer=yes
```

If users specify this keyword for jobs that use the IP communication protocol, LoadLeveler ignores the `bulkxfer` keyword.

- Specifying a `POE` line command parameter on interactive jobs.

Example:

```
poe_job -use_bulk_xfer=yes
```

- Specifying an environment variable on interactive jobs.

Example:

```
export MP_USE_BULK_XFER=yes
poe_job
```

- Because LoadLeveler honors the bulk transfer request only for LAPI or MPI jobs, users must ensure that the **network** keyword in the job command file specifies the `MPI`, `LAPI`, or `MPI_LAPI` protocol for user space communication.

Examples:

```
network.MPI =sn_single,not_shared,US,HIGH
network.MPI_LAPI =sn_single,not_shared,US,HIGH
```

Preparing a job for checkpoint/restart

LoadLeveler has the ability to checkpoint your entire job step, and to allow a job step to restart from the last checkpoint. When a job step is checkpointed, the entire state of each process of that job step is saved by the operating system. On AIX, this checkpoint capability is built in to the base operating system.

Use the information in Table 43 on page 175 to correctly configure your job for checkpointing.

Table 43. Checkpoint configurations

To specify that:	Do this:
<p>Your job is checkpointable</p>	<ul style="list-style-type: none"> • Add either one of the following two options to your job command file: <ol style="list-style-type: none"> 1. checkpoint = yes This enables your job to checkpoint in any of the following ways: <ul style="list-style-type: none"> – The application can initiate the checkpoint. This is only available on AIX. – Checkpoint from a program which invokes the ll_ckpt API. – Checkpoint using the llckpt command. – As the result of a flush command. OR 2. checkpoint = interval This enables your job to checkpoint in any of the following ways: <ul style="list-style-type: none"> – The application can initiate the checkpoint. This is only available on AIX. – Checkpoint from a program which invokes the ll_ckpt API. – Checkpoint using the llckpt command. – Checkpoint automatically taken by LoadLeveler. – As the result of a flush command. • If you would like your job to checkpoint itself, use the API ll_init_ckpt in your serial application, or mpc_init_ckpt for parallel jobs to cause the checkpoint to occur. This is only available on AIX.
<p>Your job step's executable is to be copied to the execute node</p>	<p>Add the ckpt_execute_dir keyword to the job command file.</p>

Preparing a job for checkpoint/restart

Table 43. Checkpoint configurations (continued)

To specify that:	Do this:
<p>LoadLeveler automatically checkpoints your job at preset intervals</p>	<ol style="list-style-type: none"> <li data-bbox="691 258 1419 882"> <p>Add the following option to your job command file:</p> <p>checkpoint = interval</p> <p>This enables your job to checkpoint in any of the following ways:</p> <ul style="list-style-type: none"> Checkpoint automatically at preset intervals Checkpoint initiated from user application. This is only available on AIX. Checkpoint from a program which invokes the ll_ckpt API Checkpoint using the llckpt command As the result of a flush command <li data-bbox="691 577 1419 882"> <p>The system administrators must set the following two keywords in the configuration file to specify how often LoadLeveler should take a checkpoint of the job. These two keywords are:</p> <p>MIN_CKPT_INTERVAL = number Where <i>number</i> specifies the initial period, in seconds, between checkpoints taken for running jobs.</p> <p>MAX_CKPT_INTERVAL = number Where <i>number</i> specifies the maximum period, in seconds, between checkpoints taken for running jobs.</p> <p>The time between checkpoints will be increased after each checkpoint within these limits as follows:</p> <ul style="list-style-type: none"> The first checkpoint is taken after a period of time equal to the MIN_CKPT_INTERVAL has passed. The second checkpoint is taken after LoadLeveler waits <i>twice as long</i> (MIN_CKPT_INTERVAL X 2) The third checkpoint is taken after LoadLeveler waits twice as long again (MIN_CKPT_INTERVAL X 4) before taking the third checkpoint. <p>LoadLeveler continues to double this period until the value of MAX_CKPT_INTERVAL has been reached, where it stays for the remainder of the job.</p> <p>A minimum value of 900 (15 minutes) and a maximum value of 7200 (2 hours) are the defaults.</p> <p>You can set these keyword values globally in the global configuration file so that all machines in the cluster have the same value, or you can specify a different value for each machine by modifying the local configuration files.</p>
<p>Your job will not be checkpointed</p>	<p>Add the following option to your job command file:</p> <ul style="list-style-type: none"> checkpoint = no <p>This will disable checkpoint.</p>

Table 43. Checkpoint configurations (continued)

To specify that:	Do this:
Your job has successfully checkpointed and terminated. The job has left the LoadLeveler job queue and you want LoadLeveler to restart your executable from an existing checkpoint file.	<ol style="list-style-type: none"> 1. Add the following option to your job command file: <ul style="list-style-type: none"> • restart_from_ckpt = yes 2. On AIX, specify the name of the checkpoint file by setting the following job command file keywords to specify the directory and file name of the checkpoint file to be used: <ul style="list-style-type: none"> • ckpt_dir • ckpt_file <p>When the job command file is submitted, a new job will be started that uses the specified checkpoint file to restart the previously checkpointed job.</p> <p>The job command file which was used to submit the original job should be used to restart from checkpoint. The only modifications to this file should be the addition of restart_from_ckpt = yes and ensuring ckpt_dir and ckpt_file point to the appropriate checkpoint file.</p>
Your job has successfully checkpointed. The job has been vacated and remains on the LoadLeveler job queue.	<p>When the job restarts, if a checkpoint file is available, the job will be restarted from that file.</p> <p>If a checkpoint file is not available upon restart, the job will be started from the beginning.</p>

Preparing a job for preemption

Depending on various configuration options, LoadLeveler may preempt your job so that a higher priority job step can run. Administrators may:

- Configure LoadLeveler or external schedulers to preempt jobs through various methods.
- Specify preemption rules for job classes.
- Manually preempt your job using LoadLeveler interfaces.

To ensure that your job can be resumed after preemption, set the **restart** keyword in the job command file to **yes**.

Submitting a job command file

After building a job command file, you can submit it for processing either to a machine in the LoadLeveler cluster or one outside of the cluster. (See “Querying multiple LoadLeveler clusters” on page 65 for information on submitting a job to a machine outside the cluster.) You can submit a job command file either by using the GUI or the **llsubmit** command.

When you submit a job, LoadLeveler assigns a job identifier and one or more step identifiers.

The LoadLeveler job identifier consists of the following:

machine name

The name of the machine which assigned the job identifier.

Submitting a job command file

jobid A number given to a group of job steps that were initiated from the same job command file.

The LoadLeveler step identifier consists of the following:

job identifier

The job identifier.

stepid A number that is unique for every job step in the job you submit.

If a job command file contains multiple job steps, every job step will have the same jobid and a unique stepid.

For an example of submitting a job, see Chapter 10, “Example: Using commands to build, submit, and manage jobs,” on page 215.

In a multicluster environment, job and step identifiers are assigned by the local cluster and are retained by the job regardless of what cluster the job runs in.

Submitting a job using a submit-only machine

You can submit jobs from submit-only machines. Submit-only machines allow machines that do not run LoadLeveler daemons to submit jobs to the cluster. You can submit a job using either the submit-only version of the GUI or the `llsubmit` command.

To install submit-only LoadLeveler, follow the procedure in the *TWS LoadLeveler: Installation Guide*.

In addition to allowing you to submit jobs, the submit-only feature allows you to cancel and query jobs from a submit-only machine.

Working with parallel jobs

LoadLeveler allows you to schedule parallel batch jobs that have been written using the following:

- On AIX 5L and Linux:
 - IBM Parallel Environment (PE)
 - MPICH, which is an open-source, portable implementation of the Message-Passing Interface Standard developed by Argonne National Laboratory
 - MPICH-GM, which is a port of MPICH on top of Myrinet GM code
- On Linux:
 - MVAPICH, which is a high performance implementation of MPI-1 over InfiniBand based on MPICH support for PE is available in this release of LoadLeveler for Linux

Scheduler support for parallel jobs

Several LoadLeveler job command language keywords are associated with parallel jobs. Whether a keyword is appropriate is dependent upon the type of LoadLeveler scheduler you are running.

Table 44 on page 179 shows you the parallel keywords supported by LoadLeveler’s `LL_DEFAULT`, `BACKFILL`, and `API` schedulers. If your administrator disabled the `LL_DEFAULT` LoadLeveler scheduler to run an external scheduler, see “Replacing

the default LoadLeveler scheduling algorithm with an external scheduler” on page 106 for an explanation of which keywords are supported.

Table 44. Parallel keywords supported by the LL_DEFAULT, BACKFILL, and API schedulers

Keywords supported by the LL_DEFAULT scheduler	Keywords supported by the BACKFILL and API schedulers
<ul style="list-style-type: none"> • max_processors • min_processors • Adapter requirement 	<ul style="list-style-type: none"> • network • node • node_usage • tasks_per_node • total_tasks • task_geometry • blocking

These keywords are used in the examples in this topic, and are described in more detail in “Job command file keyword descriptions” on page 336.

Step for controlling whether LoadLeveler copies environment variables to all executing nodes

You may specify that LoadLeveler is to copy, either to all executing nodes or to only the master executing node, the environment variables that are specified in the environment job command file statement for a parallel job.

Before you begin: You need to know:

- Whether Parallel Environment (PE) will be used to run the parallel job; if so, then LoadLeveler does not have to copy the application environment to the executing nodes.
- How to correctly specify the **env_copy** keyword. For information about keyword syntax and other details, see the **env_copy** keyword description.
- To specify whether LoadLeveler is to copy environment variables to only the master node, or to all executing nodes, use the **#@ env_copy** keyword in the job command file.

Alternative: You can use the Job Builder window in the LoadLeveler GUI to specify a value for this keyword.

Ensuring that parallel jobs in a cluster run on the correct levels of PE and LoadLeveler software

If support for parallel POE jobs is required, users must be aware that when LoadLeveler uses Parallel Environment for parallel job submission, that the PE software requires the same level of PE to be used throughout the parallel job. Different levels of PE cannot be mixed. For example, PE 4.3 supports only LoadLeveler 3.4, and PE 4.2 supports only LoadLeveler 3.3. Therefore, a POE parallel job cannot run some of its tasks on LoadLeveler 3.4 machines and the remaining tasks on LoadLeveler 3.3 machines.

The **requirements** keyword of the job command file can be used to ensure that all the tasks of a POE job run on compatible levels of PE and LoadLeveler software in a cluster. Here are three examples showing different ways this can be done:

1. If the following requirements statement is included in the job command file, LoadLeveler’s central manager will select only 3.4 or higher machines with the appropriate OpSys level for this job step.

```
# @ requirements = (LL_Version >= "3.4") && (OpSys == "AIX53")
```

Working with parallel jobs

The requirements expression should contain the OpSys specification because the llsubmit command automatically adds the OpSys of the submitting machine to the other job requirements unless an OpSys requirement has already been explicitly specified.

- If a requirements statement such as the following is specified, the tasks of a POE job will see a consistent environment when "hostname1" and "hostname2" run the same levels of PE and LoadLeveler software.

```
# @ requirements = (Machine == { "hostname1" "hostname2" }) && (OpSys == "AIX53")
```

- If the mixed cluster has been partitioned into 3.4 and 3.3 LoadLeveler pools, then you may use a requirements statement similar to one of the two following statements to select machines running the same levels of software.

```
• # @ requirements = (Pool == 33) && (OpSys == "AIX53")
```

```
• # @ requirements = (Pool == 32) && (OpSys == "AIX52")
```

Here, it is assumed that all the 3.4 machines in this mixed cluster are assigned to pool 33 and all 3.3 machines are assigned to pool 32. A LoadLeveler administrator can use the **pool_list** keyword of the machine stanza of the LoadLeveler administration file to assign machines to pools.

If a statement such as # @ executable = /bin/poe is specified in a job command file, and if the job is intended to be run on 3.3 machines, then it is important that the job be submitted from a 3.3 machine. When the "executable" keyword is used, LoadLeveler will copy the associated binary on the submitting machine and send it to a running machine for execution. In this example, the POE program will fail if the submitting and the running machines are at different software levels. In a mixed cluster, this problem can be circumvented by not using the **executable** keyword in the job command file. By omitting this keyword, the job command file itself is the shell script that will be executed. If this script invokes a local version of the POE binary then there is no compatibility problem at run time.

Task-assignment considerations

You can use the keywords listed in Table 45 to specify how LoadLeveler assigns tasks to nodes. With the exception of unlimited blocking, each of these methods prioritizes machines in an order based on their **MACHPRIO** expressions. Various task assignment keywords can be used in combination, and others are mutually exclusive.

Table 45. Valid combinations of task assignment keywords are listed in each column

Keyword	Valid Combinations							
total_tasks	X	X						
tasks_per_node			X	X				
node = <min, max>			X					
node = <number>	X			X				
min_processors					X		X	
max_processors						X	X	
task_geometry								X
blocking		X						

The following examples show how each allocation method works. For each example, consider a 3-node SP with machines named "N1," "N2," and "N3". The

machines' order of priority, according to the values of their MACHPRIO expressions, is: N1, N2, N3. N1 has 4 initiators available, N2 has 6, and N3 has 8.

node and total_tasks

When you specify the **node** keyword with the **total_tasks** keyword, the assignment function will allocate all of the tasks in the job step evenly among however many nodes you have specified. If the number of **total_tasks** is not evenly divisible by the number of nodes, then the assignment function will assign any larger groups to the first nodes on the list that can accept them. In this example, 14 tasks must be allocated among 3 nodes:

```
# @ node=3
# @ total_tasks=14
```

Table 46 shows the machine, available initiators, and assigned tasks:

Table 46. *node and total_tasks*

Machine	Available Initiators	Assigned Tasks
N1	4	4
N2	6	5
N3	8	5

The assignment function divides the 14 tasks into groups of 5, 5, and 4, and begins at the top of the list, to assign the first group of 5. The assignment function starts at N1, but because there are only 4 available initiators, cannot assign a block of 5 tasks. Instead, the function moves down the list and assigns the two groups of 5 to N2 and N3, the assignment function then goes back and assigns the group of 4 tasks to N1.

node and tasks_per_node

When you specify the **node** keyword with the **tasks_per_node** keyword, the assignment function will assign tasks in groups of the specified value among the specified number of nodes.

```
# @ node = 3
# @ tasks_per_node = 4
```

blocking

When you specify **blocking**, tasks are allocated to machines in groups (blocks) of the specified number (blocking factor). The assignment function will assign one block at a time to the machine which is next in the order of priority until all of the tasks have been assigned. If the total number of tasks are not evenly divisible by the blocking factor, the remainder of tasks are allocated to a single node. The **blocking** keyword must be specified with the **total_tasks** keyword. For example:

```
# @ blocking = 4
# @ total_tasks = 17
```

Where **blocking** specifies that a job's tasks will be assigned in blocks, and **4** designates the size of the blocks. Table 47 shows how a blocking factor of 4 would work with 17 tasks:

Table 47. *Blocking*

Machine	Available Initiators	Assigned Tasks
N1	4	4
N2	6	5

Working with parallel jobs

Table 47. Blocking (continued)

Machine	Available Initiators	Assigned Tasks
N3	8	8

The assignment function first determines that there will be 4 blocks of 4 tasks, with a remainder of one task. Therefore, the function will allocate the remainder with the first block that it can. N1 gets a block of four tasks, N2 gets a block, plus the remainder, then N3 gets a block. The assignment function begins again at the top of the priority list, and N3 is the only node with enough initiators available, so N3 ends up with the last block.

unlimited blocking

When you specify unlimited blocking, the assignment function will allocate as many jobs as possible to each node; the function prioritizes nodes primarily by how many initiators each node has available, and secondarily on their MACHPRIO expressions. This method allows you to allocate tasks among as few nodes as possible. To specify unlimited blocking, specify "unlimited" as the value for the blocking keyword. The `total_tasks` keyword must also be specified with unlimited blocking. For example:

```
# @ blocking = unlimited
# @ total_tasks = 17
```

Table 48 lists the machine, available initiators, and assigned tasks for unlimited blocking:

Table 48. Unlimited blocking

Machine	Available Initiators	Assigned Tasks
N3	8	8
N2	6	6
N1	4	3

The assignment function begins with N3 (because N3 has the most initiators available), and assigns 8 tasks, N2 takes six, and N1 takes the remaining 3.

task_geometry

The `task_geometry` keyword allows you to specify which tasks run together on the same machines, although you cannot specify which machines. In this example, the `task_geometry` keyword groups 7 tasks to run on 3 nodes:

```
# @ task_geometry = {(5,2)(1,3)(4,6,0)}
```

The entire `task_geometry` expression must be enclosed within braces. The task IDs for each node must be enclosed within parenthesis, and must be separated by commas. The entire range of task IDs that you specify must begin with zero, and must end with the task ID which is one less than the total number of tasks. You can specify the task IDs in any order, but you cannot skip numbers (the range of task IDs must be complete). Commas may only appear between task IDs, and spaces may only appear between nodes and task IDs.

Submitting jobs that use striping

When communication between parallel tasks occurs only over a single device such as `en0`, the application and the device are gated by each other. The device must wait for the application to fill a communication buffer before it transmits the buffer and the application must wait for the device to transmit and empty the buffer before it can refill the buffer. Thus the application and the device must wait for each other and this wastes time.

The technique of striping refers to using two or more communication paths to implement a single communication path as perceived by the application. As the application sends data, it fills up a buffer on one device. As that buffer is transmitted over the first device, the application's data begins filling up a second buffer and the application perceives no delay in being able to write. When the second buffer is full, it begins transmission over the second device and the application moves on to the next device. When all devices have been used, the application returns to the first device. Much, if not all of the buffer on the first device has been transmitted while the application wrote to the buffers on the other devices so the application waits for a minimal amount of time or possibly does not wait at all.

LoadLeveler supports striping in two ways. When multiple switch planes or networks are present, striping over them is indicated by requesting **sn_all** (multiple networks).

If multiple adapters are present on the same network and the communication subsystem, such as LAPI, supports striping over multiple adapters on the same network, specifying the **instances** keyword on the network statement requests striping over adapters on the same network. The **instances** keyword specifies the number of adapters on a single network to stripe on. It is possible to stripe over multiple networks and over multiple adapters on each network by specifying both **sn_all** and a value for **instances** greater than one. For HPS adapters, only machines that are connected to both networks are considered for **sn_all** jobs.

- **User space striping:** When **sn_all** is specified on a network statement with **US** mode, LoadLeveler commits an equivalent set of adapter resources (adapter windows and memory) on each of the networks present in the system to the job on each node where the job runs. The communication subsystem is initialized to indicate that it should use the user space communication protocol on all the available switch adapters to service communication requests on behalf of the application.
- **IP striping:** When the **sn_all** device is specified on a network statement with the **IP** mode, LoadLeveler attempts to locate the striped IP address associated with the switch adapters, known as the multi-link address. If it is successful, it passes the multi-link address to POE for use. If multi-link addresses are not available, LoadLeveler instructs POE to use the IP address of one of the switch adapters. The IP address that is used is different each time a choice has to be made in an attempt to balance the adapter use. Multi-link addresses must be configured on the system prior to running LoadLeveler and they are specified with the **multilink_address** keyword on the switch adapter stanza in the administration file. If a multi-link address is specified for a node, LoadLeveler assigns the multi-link address and multi-link IP name to the striping adapter on that node. If a multi-link address is not present on a node, the **sn_all** adapter associated with the node will not have an IP address or IP name. If not all of the nodes of a system have multi-link addresses but some do, LoadLeveler will only dispatch jobs that request IP striping to nodes that have multi-link addresses.

Jobs that request striping (both user space and IP) can be submitted to nodes with only one switch adapter. In that situation, the result is the same as if the job requested no striping.

Note: When configured, a multi-link address is associated with the virtual **m10** device. The IP address of this device is the multi-link address. The **llextrpd** program will create a stanza for the **m10** device that will appear similar to Ethernet or token ring adapter stanzas except that it will include the **multilink_list** keyword that lists the adapters it performs

Working with parallel jobs

striping over. As with any other device with an IP address, the **m10** device can be requested in IP mode on the network statement. Doing so would yield a comparable effect to requesting **sn_all** IP except that no checking would be performed by LoadLeveler to ensure the associated adapters are actually working. Thus it would be possible to dispatch a job that requested communication over **m10** only to have the job fail because the switch adapters that **m10** stripes over were down.

- **Striping over one network:** If the **instances** keyword is specified on a network statement with a value greater than one, LoadLeveler allocates multiple sets of resources for the protocol using as many sets as the **instances** keyword specified. For User Space jobs, these sets are adapter windows and memory. For IP jobs, these sets are IP addresses. If multiple adapters exist on each node on the same network, then these sets of adapter resources will be distributed among all the available adapters on the same network. Even though LoadLeveler will allocate resources to support striping over a single network, the communication subsystem must be capable of exploiting these resources in order for them to be used.

Understanding striping over multiple networks

Striping over multiple networks involves establishing a communication path using one or more of the available communication networks or switch fabrics. How those paths are established depends on the network adapter that is present. For the SP Switch2 family of adapters, it is not necessary to acquire communication paths among all tasks on all fabrics as long as there is at least one fabric over which all tasks can communicate. However, each adapter on a machine, if it is available, must use exactly the same adapter resources (window and memory amount) as the other adapters on that machine. Switch Network Interface for HPS adapters are not required to use exactly the same resources on each network, but in order for a machine to be selected, there must be an available communication path on all networks.

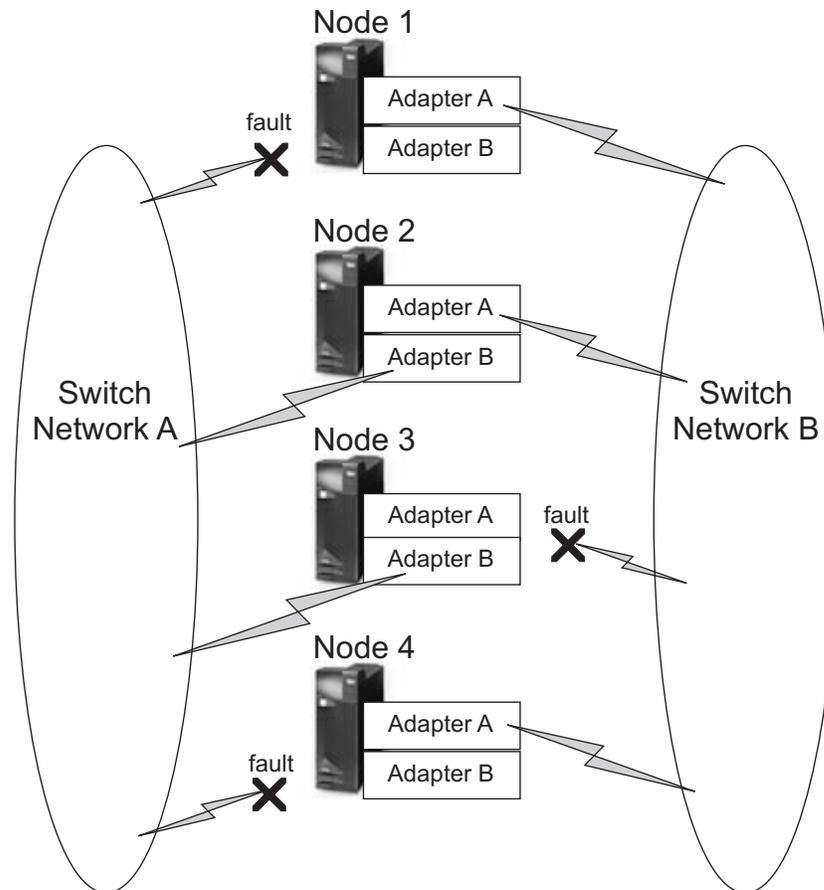


Figure 24. Striping over multiple networks

Consider these sample scenarios using the network configuration as shown in Figure 24 where the adapters are from the SP Switch2 family:

- If a three node job requests striping over networks, it will be dispatched to Node 1, Node 2 and Node 4 where it can communicate on Network B as long as the adapters on each machine have a common window free and sufficient memory available. It cannot run on Node 3 because that node only has a common communication path with Node 2, namely Network A.
- If a three node job does not request striping, it will not be run because there are not enough adapters connected to Network A to run the job. Notice both the adapter connected to Network A on Node 1 and the adapter connected to Network A on Node 4 are both at fault. SP Switch2 family adapters can only use the adapter connected to Network A for non-striped communication.
- If a three node job requests striped IP and some but not all of the nodes have multi-linked addresses, the job will only be dispatched to the nodes that have the multi-link addresses.

Consider these sample scenarios using the network configuration as shown in Figure 24 where the adapters are Switch Network Interface for HPS adapters:

- If a three node job requests striping over networks, it will not be dispatched because there are not three nodes that have active connections to both networks.
- If a three node job does not request striping, it can be run on Node 1, Node 2, and Node 4 because they have an active connection to network B.

Working with parallel jobs

- If a three node job requests striped IP and some but not all of the nodes have multi-linked addresses, the job will only be dispatched to the nodes that have the multi-link addresses.

Note that for all adapter types, adapters are allocated to a step that requests striping based on what the node knows is the available set of networks or fabrics. LoadLeveler expects each node to have the same knowledge about available networks. If this is not true, it is possible for tasks of a step to be assigned adapters which cannot communicate with tasks on other nodes.

Similarly, LoadLeveler expects all adapters that are identified as being on the same Network ID or fabric ID to be able to communicate with each other. If this is not true, such as when LoadLeveler operates with multiple, independent sets of networks, other attributes of the Step, such as the requirements expression, must be used to ensure that only nodes from a single network set are considered for the step.

As you can see from these scenarios, LoadLeveler will find enough nodes on the same communication path to run the job. If enough nodes connected to a common communication path cannot be found, no communication can take place and the job will not run.

Understanding striping over a single network

Striping over a single network is only supported by Switch Network Interface for HPS adapters.

Figure 25 shows a network configuration where the adapters support striping over a single network.

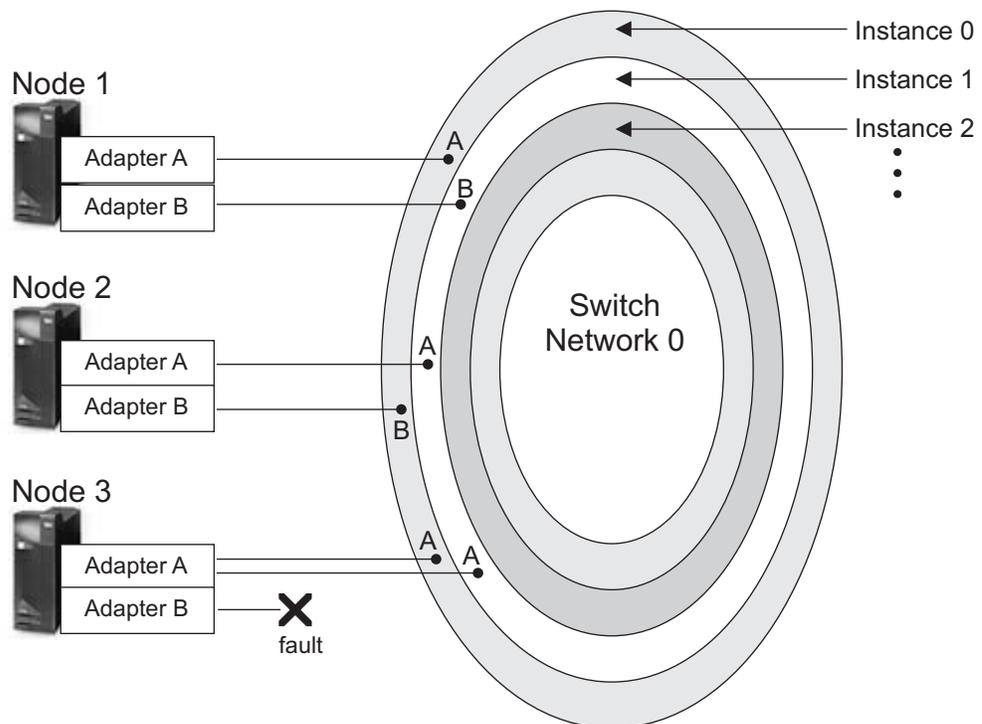


Figure 25. Striping over a single network

Both Adapter A and Adapter B on a node are connected to Network 0. The entire oval represents the physical network and the concentric ovals (shaded differently) represent the separate communication paths created for a job by the **instances** keyword on the network statement. In this case a three node job requests two instances for communication. On Node 1, adapter A is used for instance 0 and adapter B is used for instance 1. There is no requirement to use the same adapter for the same instance so on Node 2, adapter B was used for instance 0 and adapter A for instance 1.

On Node 3, where a fault is keeping adapter B from connecting to the network, adapter A is used for both instance 0 and instance 1 and Node 3 is available for the job to use.

The network itself does not impose any limitation on the total number of communication paths that can be active at a given time for either a single job or all the jobs using the network. As long as nodes with adapter resources are available, additional communication paths can be created.

Examples: Requesting striping in network statements

You request that a job be run using striping with the **network** statement in your job command file. The default when instances is not specified for a job in the **network** statement is controlled by the class stanza keyword for **sn_all**. For more information on the **network** and **max_protocol_instances** statements, see the keyword descriptions in “Job command file keyword descriptions” on page 336.

Shown here are examples of IP and user space network modes:

- **Example 1: Requesting striping using IP mode**

To submit a job using IP striping, your network statement would look like this:

```
network.MPI = sn_all,,IP
```

- **Example 2: Requesting striping using user space mode**

To submit a job using user space striping, your network statement would look like this:

```
network.MPI = sn_all,,US
```

- **Example 3: Requesting striping over a single network**

To request IP striping over multiple adapter on a single network, the network statement would look like this:

```
network.MPI = sn_single,,IP,,instances=2
```

If the nodes on which the job runs have two or more adapters on the same network, two different IP addresses will be allocated to each task for MPI communication. If only one adapter exists per network, the same IP address will be used twice for each task for MPI communication.

- **Example 4: Requesting striping over multiple networks and multiple adapters on the same network**

To submit a user space job that will stripe MPI communication over multiple adapters on all networks present in the system the network statement would look like this:

```
network.MPI = sn_all,,US,,instances=2
```

If, on a node where the job runs, there are two adapters on each of the two networks, one adapter window would be allocated from each adapter for MPI communication by the job. If only one network were present with two adapters, one adapter window from each of the two adapters would be used. If two

Working with parallel jobs

networks were present but each only had one adapter on it, two adapter windows from each adapter would be used to satisfy the request for two instances.

Running interactive POE jobs

POE will accept LoadLeveler job command files; however, you can still set the following environment variables to define specific LoadLeveler job attributes before running an interactive POE job:

LOADL_ACCOUNT_NO

The account number associated with the job.

LOADL_INTERACTIVE_CLASS

The class to which the job is assigned.

MP_TASK_AFFINITY

The affinity preferences requested for the job.

For information on other POE environment variables, see *IBM Parallel Environment for AIX; Operation and Use, Volume 1*.

For an interactive POE job, LoadLeveler does not start the POE process therefore LoadLeveler has no control over the process environment or resource limits.

You also may run interactive POE jobs under a reservation. For additional details about reservations and submitting jobs to run under them, see “Working with reservations” on page 197.

Interactive POE jobs cannot be submitted to a remote cluster.

Running MPICH, MVAPICH, and MPICH-GM jobs

LoadLeveler for AIX 5L and Linux support three open-source implementations of the Message-Passing Interface (MPI).

MPICH is an open-source, portable implementation of the MPI Standard developed by Argonne National Laboratory. It contains a complete implementation of version 1.2 of the MPI Standard and also significant parts of MPI-2, particularly in the area of parallel I/O. MPICH, MVAPICH, and MPICH-GM are the three MPI implementations supported by LoadLeveler for AIX 5L and Linux:

- Additional documentation for MPICH is available from the Argonne National Laboratory web site at <http://www-unix.mcs.anl.gov/mapi/mpich/index.htm#docs>.
- MVAPICH is a high performance implementation of MPI-1 over InfiniBand based on MPICH. Additional documentation for MVAPICH is available at the Ohio State University web site at <http://nowlab.cse.ohio-state.edu/projects/mapi-iba/>.
- MPICH-GM is a port of MPICH on top of GM (ch_gm). GM is a low-level message-passing system for Myrinet Networks. Additional documentation for MPICH-GM is available from the Myrinet web site at <http://www.myri.com/scs/>.

For either MPICH, MVAPICH, or MPICH-GM, LoadLeveler allocates the machines to run the parallel job and starts the implementation specific script as master task. Some of the options of implementation specific scripts might not be required or are not supported when used with LoadLeveler.

The following standard **mpirun** script options are not supported:

-map <list>

The **mpirun** script can either take a **machinefile** or a mapping of the machines in which to run the **mpirun** job. If both the **machinefile** and **map** are specified, then the map list overrides the **machinefile**. Because we want LoadLeveler to decide which nodes to run on, use the **machinefile** specified by the environment variable **LOADL_HOSTFILE**. Specifying a mapping of the host name is not supported.

-allcpus

This option is *only* supported when the **-machinefile** option is used. The **mpirun** script will run the job using all machines specified in the machine file, without the need to specify the **-np** option. Without specifying **machinefile**, the **mpirun** script will look in the default machines <arch> file to find the machines on which to run the job. The machines defined in the default file might not match what LoadLeveler has selected, which will cause the job to be removed.

-exclude <list>

This option is not supported because if you specified a machine in the exclude list that has already been scheduled by LoadLeveler to run the job, the job will be removed.

-dbg

This option might be used to select a debugger. This option is used to select a debugger to be used with the **mpirun** script. LoadLeveler currently does not support running interactive MPICH jobs, so starting **mpirun** jobs under a debugger is not supported.

-ksq

This option keeps the send queue. This is useful if you expect later to attach totalview to the running (or deadlocked) job, and want to see the send queues. This option is used for debugging purposes when attaching the **mpirun** job to totalview. Since we do not support running debuggers under LoadLeveler MPICH job management, this option is not supported.

-machinedir <directory>

This option looks for the machine files in the indicated directory. LoadLeveler will create a **machinefile** that contains the host name for each task in the **mpirun** job. The environment variable **LOADL_HOSTFILE** contains the full path to the **machinefile**. A different **machinefile** is created per job and stored in the LoadLeveler execute directory. Because there might be multiple jobs running at one time, we do not want the **mpirun** script to choose any file in the execute directory because it might not be the correct file that the central manager has assigned to the job step. This option is therefore not supported, use the **-machinefile** option instead.

- When using MPICH, the **mpirun** script is run on the first machine allocated to the job. The **mpirun** script starts the actual execution of the parallel tasks on the other nodes included in the LoadLeveler cluster using **lspawn.stdio** as **RSHCOMMAND**.

The following option of MPICHs **mpirun** script is not supported.

-nolocal

This option specifies not to run on the local machine. The default behavior of MPICH (p4) is that the first MPI process is always spawned on the machine which **mpirun** has invoked. The **-nolocal** option disables the default behavior and does not run the MPI process on the local node. Under

Working with parallel jobs

LoadLeveler's MPICH Job management, it is required that at least one task is run on the local node, so the **-nolocal** option should *not* be used.

- When using MVAPICH, the **mpirun_rsh** command is run on the first machine allocated to the job as master task. The **mpirun_rsh** command starts the actual execution of parallel tasks on the other nodes included in the LoadLeveler cluster using **llspawn** as **RSHCOMMAND**.

The following options of MVAPICHs **mpirun_rsh** command are not supported when used with LoadLeveler.

-rsh

Specifies to use **rsh** for connecting.

-ssh

Specifies to use **ssh** for connecting. The **-rsh** and **-ssh** options are supported, but the behavior has been changed to run **mpirun_rsh** jobs under LoadLeveler MPICH job manager. Replace the **-rsh** and **-ssh** commands with **llspawn** before compiling **mpirun_rsh**. Even if you select **-rsh** and **-ssh**, the **llspawn** command is actually used in place of **-rsh** and **-ssh** at runtime.

-xterm

Runs remote processes under **xterm**. This option starts an xterm window for each task in the **mpirun** job and runs the remote shell with the application inside the xterm window. This will not work under LoadLeveler because the **llspawn** command replaces the remote shell (**rsh** or **ssh**) and **llspawn** is not kept alive to the end of the application process.

-debug

Runs each process under the control of **gdb**. This option is used to select a debugger to be used with **mpirun** jobs. LoadLeveler currently does not support running interactive MPICH jobs so starting **mpirun** jobs under a debugger is not supported. This option also requires **xterm** to be working properly as it opens **gdb** under an xterm window. Since we do not support the **-xterm** option, the **-debug** option is also not supported.

h1 h2....

Specifies the names of hosts where processes should run. The **mpirun_rsh** script can either take a host file or read in the names of the hosts, **h1 h2** and so on, in which to run the **mpirun** job. If both host file and list of machines are specified in the **mpirun_rsh** arguments, **mpirun_rsh** will have an error parsing the arguments. Because we want LoadLeveler to decide which nodes to run on, you should use the host list specified by the environment variable **LOADL_HOSTFILE**. Specifying the names of the hosts is not supported.

- When using MPICH-GM, the **mpirun.ch_gm** script is run on the first machine allocated to the job as master task. The **mpirun.ch_gm** script starts the actual execution of the parallel tasks on the other nodes included in the LoadLeveler cluster using the **llspawn** command as **RSHCOMMAND**.

The following options of MPICH-GMs **mpirun** script are not supported when used with LoadLeveler.

--gm-kill <n>

This is an option that allows you to kill all remaining processes **<n>** seconds after the first one dies or exits. Do not specify this option when running the application under LoadLeveler, because LoadLeveler will handle the cleanup of the tasks.

--gm-tree-spawn

This is an option that uses a two-level spawn tree to launch the processes in an effort to reduce the load on any particular host. Because LoadLeveler

is providing its own scalable method for spawning the application tasks from the master host, using the `lspawn` command, spawning processes in a tree-like fashion is not supported.

-totalview

This option is used to select a totalview debugging session to be used with the `mpirun` script. LoadLeveler currently does not support running interactive MPICH jobs, so starting `mpirun` jobs under a debugger is not supported.

- r This is an optional option for MPICH-GM, which forces the removal of the shared memory files. Because this option is not required, it is not supported. If you specify this option, it will be ignored.

-ddt

This option is used to select a DDT debugging session to be used with the `mpirun` script. LoadLeveler currently does not support running interactive MPICH jobs, so starting `mpirun` jobs under a debugger is not supported.

Sample programs are available:

- See “MPICH sample job command file” on page 192 for a sample MPICH job command file.
- See “MPICH-GM sample job command file” on page 194 for a sample MPICH-GM job command file.
- See “MVAPICH” on page 195 for a sample MVAPICH job command file.
- The LoadLeveler samples directory also contains sample files:
 - On AIX 5L, use directory `/usr/lpp/LoadL/full/samples/l1mpich`
 - On Linux, use directory `/opt/ibm11/LoadL/full/samples/l1mpich`

These sample files include:

- `ivp.c`: A simple MPI application that you may run as an MPICH, MVAPICH, or MPICH-GM job.
- Job command files to run the `ivp.c` program as a batch job:
 - For MPICH: `mpich_ivp.cmd`
 - For MPICH-GM: `mpich_gm_ivp.cmd`

Examples: Building parallel job command files

This topic contains sample job command files for the following parallel environments:

- IBM AIX Parallel Operating Environment (POE)
- MPICH
- MPICH-GM
- MVAPICH

POE sample job command file

Figure 26 on page 192 is a sample job command file for POE.

Working with parallel jobs

```
#
# @ job_type = parallel
# @ environment = COPY_ALL
# @ output = poe.out
# @ error = poe.error
# @ node = 8,10
# @ tasks_per_node = 2
# @ network.LAPI = sn_all,US,,instances=1
# @ network.MPI = sn_all,US,,instances=1
# @ wall_clock_limit = 60
# @ executable = /usr/bin/poe
# @ arguments = /u/richc/My_POE_program -euilib "us"
# @ class = POE
# @ queue
```

Figure 26. POE job command file – multiple tasks per node

Figure 26 shows the following:

- The total number of nodes requested is a minimum of eight and a maximum of 10 (**node=8,10**). Two tasks run on each node (**tasks_per_node=2**). Thus the total number of tasks can range from 16 to 20.
- Each task of the job will run using the LAPI protocol in US mode with a switch adapter (**network.LAPI=sn_all,US,,instances=1**), and using the MPI protocol in US mode with a switch adapter (**network.MPI=sn_all,US,,instances=1**).
- The maximum run time allowed for the job is 60 seconds (**wall_clock_limit=60**).

Figure 27 is a second sample job command file for POE

```
#
# @ job_type = parallel
# @ input = poe.in.1
# @ output = poe.out.1
# @ error = poe.err
# @ node = 2,8
# @ network.MPI = sn_single,shared,IP
# @ wall_clock_limit = 60
# @ class = POE
# @ queue
/usr/bin/poe /u/richc/my_POE_setup_program -infolevel 2
/usr/bin/poe /u/richc/my_POE_main_program -infolevel 2
```

Figure 27. POE sample job command file – invoking POE twice

Figure 27 shows the following:

- POE is invoked twice, through **my_POE_setup_program** and **my_POE_main_program**.
- The job requests a minimum of two nodes and a maximum of eight nodes (**node=2,8**).
- The job by default runs one task per node.
- The job uses the MPI protocol with a switch adapter in IP mode (**network.MPI=sn_single,shared,IP**).
- The maximum run time allowed for the job is 60 seconds (**wall_clock_limit=60**).

MPICH sample job command file

Figure 28 on page 193 is a sample job command file for MPICH.

```

# ! /bin/ksh
# LoadLeveler JCF file for running an MPICH job
# @ job_type = MPICH
# @ node = 4
# @ tasks_per_node = 2
# @ output = mpich_test.$(cluster).$(process).out
# @ error = mpich_test.$(cluster).$(process).err
# @ queue
echo "-----"
echo LOADL_STEP_ID=$LOADL_STEP_ID
echo "-----"

/opt/mpich/bin/mpirun -np $LOADL_TOTAL_TASKS -machinefile \
  $LOADL_HOSTFILE /common/NFS/11_bin/mpich_test

```

Figure 28. MPICH job command file - sample 1

Note: You can also specify the **job_type=parallel** keyword and invoke the **mpirun** script to run an MPICH job. In that case, the **mpirun** script would use **rsh** or **ssh** and not the **lspawn** command.

Figure 28 shows that in the following job command file statement:

```

/opt/mpich/bin/mpirun -np $LOADL_TOTAL_TASKS -machinefile \
  $LOADL_HOSTFILE /common/NFS/11_bin/mpich_test

```

-np

Specifies the number of parallel processes.

LOADL_TOTAL_TASKS

Is the environment variable set by LoadLeveler with the number of parallel processes of the job step.

-machinefile

Specifies the machine list file.

LOADL_HOSTFILE

Is the environment variable set by LoadLeveler with the file name that contains host names assigned to the parallel job step.

The following is another example of a MPICH job command file:

```

# ! /bin/ksh
# LoadLeveler JCF file for running an MPICH job
# @ job_type = MPICH
# @ node = 4
# @ tasks_per_node = 2
# @ output = mpich_test.$(cluster).$(process).out
# @ error = mpich_test.$(cluster).$(process).err
# @ executable = /opt/mpich/bin/mpirun
# @ arguments = -np $LOADL_TOTAL_TASKS -machinefile \
  $LOADL_HOSTFILE /common/NFS/11_bin/mpich_test
# @ queue

```

Figure 29. MPICH job command file - sample 2

Figure 29 shows the following:

- The **mpirun** script is specified as a value of the executable job command file keyword.
- The following **mpirun** script arguments are specified with the arguments job command file keyword:

```

-np $LOADL_TOTAL_TASKS -machinefile $LOADL_HOSTFILE /common/NFS/11_bin/mpich_test

```

Working with parallel jobs

-np
Specifies the number of parallel processes.

LOADL_TOTAL_TASKS
Is the environment variable set by LoadLeveler with the number of parallel processes of the job step.

-machinefile
Specifies the machine list file.

LOADL_HOSTFILE
Is the environment variable set by LoadLeveler with file name, which contains host names assigned to the parallel job step.

MPICH-GM sample job command file

Figure 30 is a sample job command file for MPICH-GM.

```
#!/bin/ksh
# LoadLeveler JCF file for running an MPICH-GM job
# @ job_type = MPICH
# @ resources = gmports(1)
# @ node = 4
# @ tasks_per_node = 2
# @ output = mpich_gm_test.${cluster}.${process}.out
# @ error = mpich_gm_test.${cluster}.${process}.err
# @ queue
echo "-----"
echo LOADL_STEP_ID=$LOADL_STEP_ID
echo "-----"
/opt/mpich/bin/mpirun.ch_gm -np $LOADL_TOTAL_TASKS -machinefile \
$LOADL_HOSTFILE /common/NFS/ll_bin/mpich_gm_test
```

Figure 30. MPICH-GM job command file - sample 1

Figure 30 shows the following:

- The statement **# @ resources = gmports(1)** specifies that each task consumes one GM port. This is how LoadLeveler limits the number of GM ports simultaneously in use on any machine. This resource name is the name you specified in **schedule_by_resources** in the configuration file and each machine stanza in the administration file must define GM ports and specify the quantity of GM ports available on each machine. Use the **llstatus -R** command to confirm the names and values of the configured and available consumable resources.
- In the following job command file statement:

```
/opt/mpich/bin/mpirun.ch_gm -np $LOADL_TOTAL_TASKS \  
-machinefile $LOADL_HOSTFILE /common/NFS/ll_bin/mpich_gm_test
```

/opt/mpich/bin/mpirun.ch_gm

Specifies the location of the **mpirun.ch_gm** script shipped with the MPICH-GM implementation that runs the MPICH-GM application.

-np
Specifies the number of parallel processes.

-machinefile
Specifies the machine list file.

LOADL_HOSTFILE
Is the environment variable set by LoadLeveler with file name, which contains host names assigned to the the parallel job step.

Figure 31 is another sample job command file for MPICH-GM.

```
#!/bin/ksh
# LoadLeveler JCF file for running an MPICH-GM job
# @ job_type = MPICH
# @ resources = gports(1)
# @ node = 4
# @ tasks_per_node = 2
# @ output = mpich_gm_test.${cluster}.${process}.out
# @ error = mpich_gm_test.${cluster}.${process}.err
# @ executable = /opt/mpich/bin/mpirun.ch_gm
# @ arguments = -np $LOADL_TOTAL_TASKS -machinefile \
$LOADL_HOSTFILE /common/NFS/11_bin/mpich_gm_test
# @ queue
```

Figure 31. MPICH-GM job command file - sample 2

Figure 31 shows the following:

- The **mpirun_gm** script is specified as value of the executable job command file keyword.
- The following **mpirun_gm** script arguments are specified with the arguments job command file keyword:

```
-np $LOADL_TOTAL_TASKS -machinefile $LOADL_HOSTFILE /common/NFS/11_bin/mpich_test
```

-np

Specifies the number of parallel processes.

LOADL_TOTAL_TASKS

Is the environment variable set by LoadLeveler with the number of parallel processes of the job step.

-machinefile

Specifies the machine list file.

LOADL_HOSTFILE

Is the environment variable set by LoadLeveler with file name, which contains host names assigned to the parallel job step.

MVAPICH

Figure 32 is a sample job command file for MVAPICH:

```
#! /bin/ksh
# LoadLeveler JCF file for running an MVAPICH job
# @ job_type = MPICH
# @ node = 4
# @ tasks_per_node = 2
# @ output = mvapich_test.${cluster}.${process}.out
# @ error = mvapich_test.${cluster}.${process}.err
# @ queue
echo "-----"
echo LOADL_STEP_ID=$LOADL_STEP_ID
echo "-----"

/opt/mpich/bin/mpirun_rsh -np $LOADL_TOTAL_TASKS -machinefile \
$LOADL_HOSTFILE /common/NFS/11_bin/mpich_test
```

Figure 32. MVAPICH job command file - sample 1

Figure 32 shows that in the following job command file statement:

```
/opt/mpich/bin/mpirun_rsh -np $LOADL_TOTAL_TASKS -machinefile \
$LOADL_HOSTFILE /common/NFS/11_bin/mpich_test
```

Working with parallel jobs

-np
Specifies the number of parallel processes.

LOADL_TOTAL_TASKS
Is the environment variable set by LoadLeveler with the number of parallel processes of the job step.

-machinefile
Specifies the machine list file.

LOADL_HOSTFILE
Is the environment variable set by LoadLeveler with file name, which contains host names assigned to the parallel job step.

Figure 32 on page 195 is another sample job command file for MVAPICH:

```
# ! /bin/ksh
# LoadLeveler JCF file for running an MVAPICH job
# @ job_type = MPICH
# @ node = 4
# @ tasks_per_node = 2
# @ output = mvapich_test.${cluster}.${process}.out
# @ error = mvapich_test.${cluster}.${process}.err
# @ executable = /opt/mpich/bin/mpirun_rsh
# @ arguments = -np $LOADL_TOTAL_TASKS -machinefile \
  $LOADL_HOSTFILE /common/NFS/11_bin/mpich_test
# @ queue
```

Figure 33. MVAPICH job command file - sample 2

Figure 33 shows the following:

- The **mpirun_rsh** command is specified as value for the executable job command file keyword.
- The following **mpirun_rsh** command arguments are specified with the arguments job command file keyword:

```
-np $LOADL_TOTAL_TASKS -machinefile $LOADL_HOSTFILE /common/NFS/11_bin/mpich_test
```

-np
Specifies the number of parallel processes.

LOADL_TOTAL_TASKS
Is the environment variable set by LoadLeveler with the number of parallel processes of the job step.

-machinefile
Specifies the machine list file.

LOADL_HOSTFILE
Is the environment variable set by LoadLeveler with file name, which contains host names assigned to the parallel job step.

Obtaining status of parallel jobs

Both end users and LoadLeveler administrators can obtain status of parallel jobs in the same way as they obtain status of serial jobs – either by using the **llq** command or by viewing the Jobs window on the graphical user interface (GUI). By issuing **llq -l**, or by using the Job Actions → Details selection in xloadl, users get a list of machines allocated to the parallel job. If you also need to see task instance information use the **-x** option in addition to the **-l** option (**llq -l -x**). See “llq -

Query job status” on page 449 for samples of output using the `-x` and `-l` options with the `llq` command. As an alternative, you can also use the GUI and select: Job Actions → Extended Details.

Obtaining allocated host names

`llq -l` output includes information on allocated host names. Another way to obtain the allocated host names is with the `LOADL_PROCESSOR_LIST` environment variable, which you can use from a shell script in your job command file as shown in Figure 34.

This example uses `LOADL_PROCESSOR_LIST` to perform a remote copy of a local file to all of the nodes, and then invokes `POE`. Note that the processor list contains an entry for each task running on a node. If two tasks are running on a node, `LOADL_PROCESSOR_LIST` will contain two instances of the host name where the tasks are running. The example in Figure 34 removes any duplicate entries.

Note that `LOADL_PROCESSOR_LIST` is set by `LoadLeveler`, not by the user. This environment variable is limited to 128 hostnames. If the value is greater than the 128 limit, the environment variable is not set.

```
#!/bin/ksh
# @ output      = my_POE_program.$(cluster).$(process).out
# @ error       = my_POE_program.$(cluster).$(process).err
# @ class       = POE
# @ job_type    = parallel
# @ node        = 8,12
# @ network.MPI = sn_single,shared,US
# @ queue

tmp_file="/tmp/node_list"
rm -f $tmp_file

# Copy each entry in the list to a new line in a file so
# that duplicate entries can be removed.
for node in $LOADL_PROCESSOR_LIST
do
    echo $node >> $tmp_file
done

# Sort the file removing duplicate entries and save list in variable
nodelist= sort -u /tmp/node_list

for node in $nodelist
do
    rcp localfile $node:/home/userid
done

rm -f $tmp_file

/usr/bin/poe /home/userid/my_POE_program
```

Figure 34. Using `LOADL_PROCESSOR_LIST` in a shell script

Working with reservations

Under the `BACKFILL` scheduler only, `LoadLeveler` allows authorized users to make reservations, which specify a time period during which specific node resources are reserved for use by particular users or groups.

Working with reservations

Use Table 49 to find information about working with reservations.

Table 49. Roadmap of tasks for reservation owners and users

Subtask	Associated instructions (see . . .)
Learn how reservations work in the LoadLeveler environment	<ul style="list-style-type: none">• “Overview of reservations” on page 23• “Understanding the reservation life cycle”
Creating new reservations	“Creating new reservations” on page 200
Managing jobs that run under a reservation	<ul style="list-style-type: none">• “Submitting jobs to run under a reservation” on page 202• “Removing bound jobs from the reservation” on page 203
Managing existing reservations	<ul style="list-style-type: none">• “Querying existing reservations” on page 204• “Modifying existing reservations” on page 204• “Canceling existing reservations” on page 205
Using the LoadLeveler interfaces for reservations	<ul style="list-style-type: none">• Chapter 16, “Commands,” on page 385• “Reservation API” on page 600

Understanding the reservation life cycle

From the time at which LoadLeveler creates a reservation through the time the reservation ends or is canceled, a reservation goes through various states, which are indicated in command listings and other displays or output. Understanding these states is important because the current state of a reservation dictates what actions you can take; for example, if you want to modify the start time for a reservation, you may do so only while the reservation is in Waiting state. Table 50 lists the possible reservation states, their abbreviations, and usage notes.

Table 50. Reservation states, abbreviations, and usage notes

Reservation state	Abbreviation in displays / output	Usage notes
Waiting	W	<p>When LoadLeveler first creates a reservation, the reservation is in Waiting state. While the reservation is in this state:</p> <ul style="list-style-type: none">• Only administrators and reservation owners may modify, cancel, and add users or groups to the reservation.• Administrators, reservation owners, and users or groups that are allowed to use the reservation may query it, and submit jobs to run during the reservation period.

Table 50. Reservation states, abbreviations, and usage notes (continued)

Reservation state	Abbreviation in displays / output	Usage notes
Setup	S	<p>LoadLeveler changes the state of a reservation from Waiting to Setup just before the start time of the reservation. The actual time at which LoadLeveler places the reservation in Setup state depends on the value set for the RESERVATION_SETUP_TIME keyword in the configuration file.</p> <p>While the reservation is in Setup state:</p> <ul style="list-style-type: none"> • Only administrators and reservation owners may modify, cancel, and add users or groups to the reservation. • Administrators, reservation owners, and users or groups that are allowed to use the reservation may query it, and submit jobs to run during the reservation period. <p>During this setup period, LoadLeveler:</p> <ul style="list-style-type: none"> • Stops scheduling unbound job steps to reserved nodes. • Preempts any jobs that are still running on the nodes that are reserved through this reservation. To preempt the running jobs, LoadLeveler uses the preemption method specified through the DEFAULT_PREEMPT_METHOD keyword in the configuration file. <p>Note: The default value for DEFAULT_PREEMPT_METHOD is SU (suspend), which is not supported in all environments, and the default value for PREEMPTION_SUPPORT is NONE. If you want preemption to take place at the start of the reservation, make sure the cluster is configured for preemption (see “Steps for configuring a scheduler to preempt jobs” on page 120 for more information).</p>
Active	A	<p>At the reservation start time, LoadLeveler changes the reservation state from Setup to Active. It also dispatches only job steps that are bound to the reservation, until the reservation completes or is canceled.</p> <p>LoadLeveler does not dispatch bound job steps that:</p> <ul style="list-style-type: none"> • Require certain resources, such as floating consumable resources, that are not available during the reservation period. • Have expected end times that exceed the end time of the reservation. By default, LoadLeveler allows such jobs to run, but their completion is subject to resource availability. (An administrator may configure LoadLeveler to prevent such jobs from running.) <p>These bound job steps remain idle unless the required resources become available.</p> <p>While the reservation is in Active state:</p> <ul style="list-style-type: none"> • Only administrators and reservation owners may modify, cancel, and add users or groups to the reservation. • Administrators, reservation owners, and users or groups that are allowed to use the reservation may query it, and submit jobs to run during the reservation period.

Working with reservations

Table 50. Reservation states, abbreviations, and usage notes (continued)

Reservation state	Abbreviation in displays / output	Usage notes
Active_Shared	AS	<p>At the reservation start time, LoadLeveler changes the reservation state from Setup to Active. It also dispatches only job steps that are bound to the reservation, unless the reservation was created with the SHARED mode. In this case, if reserved resources are still available after LoadLeveler dispatches any bound job steps that are eligible to run, LoadLeveler changes the reservation state to Active_Shared, and begins dispatching job steps that are not bound to the reservation. Once the reservation state changes to Active_Shared, it remains in that state until the reservation completes or is canceled. During this time, LoadLeveler dispatches both bound and unbound job steps, pending resource availability; bound job steps are considered before unbound job steps.</p> <p>The conditions under which LoadLeveler will not dispatch bound job steps are the same as those listed in the notes for the Active state.</p> <p>The actions that administrators, reservation owners, and users may perform are the same as those listed in the notes for the Active state.</p>
Canceled	CA	<p>When a reservation owner, administrator, or LoadLeveler issues a request to cancel the reservation, LoadLeveler changes the state of a reservation to Canceled and unbinds any job steps bound to this reservation. When the reservation is in this state, no one can modify or submit jobs to this reservation.</p>
Complete	C	<p>When a reservation end time is reached, LoadLeveler changes the state of a reservation to Complete. When the reservation is in this state, no one can modify or submit jobs to this reservation.</p>

Creating new reservations

You must be an authorized user or member of an authorized group to successfully create a reservation. LoadLeveler administrators define authorized users by adding the **max_reservations** keyword to the user or group stanza in the administration file. The **max_reservations** keyword setting also defines how many reservations you are allowed to own. Ask your administrator whether you are authorized to create reservations.

To be authorized to create reservations, LoadLeveler administrators also must have the **max_reservations** keyword set in their user or group stanza.

To create a reservation, use the **llmkres** command. When you create a reservation, you must:

- Specify the start time and duration of the reservation. Use the **-t** and **-d** command options, respectively.
- Explicitly specify nodes through one of the following methods, which are mutually exclusive. You must use only one method when you request LoadLeveler to create a reservation.

- The **-n** option on the **llmkres** command instructs LoadLeveler to reserve a number of nodes. LoadLeveler may select any unreserved node to satisfy a reservation. This command option is perhaps the easiest to use, because you need to know only how many nodes you want, not specific node characteristics.

The minimum number of nodes a reservation must have is 1.

- The **-h** option on the **llmkres** command instructs LoadLeveler to reserve specific nodes.
 - The **-f** option on the **llmkres** command instructs LoadLeveler to submit the specified job command file, and reserve appropriate nodes for the first job step in the job command file. Through this action, all job steps for the job are bound to the reservation. If the reservation request fails, LoadLeveler changes the state for all job steps for this job to NotQueued, and will not schedule any of those job steps to run.
 - The **-j** option on the **llmkres** command instructs LoadLeveler to reserve appropriate nodes for that job step. Through this action, the job step is bound to the reservation. If the reservation request fails, the job step remains in the same state as it was before.
- The **-c** option on the **llmkres** command instructs LoadLeveler to reserve a number of Blue Gene compute nodes (C-nodes). The **-j** and **-f** option also reserve Blue Gene resources if the job type is bluegene.

You also may define other reservation attributes, including:

- Whether additional users or groups are allowed to use the reservation. Use the **-U** or **-G** command options, respectively.
- Whether the reservation will be in one or both of these optional modes:
 - **SHARED** mode: When you use the **-s** command option, LoadLeveler allows reserved resources to be shared by job steps that are not associated with a reservation. This mode enables the efficient use of reserved resources; if the bound job steps do not use all of the reserved resources, LoadLeveler can schedule unbound job steps as well so the resources do not remain idle. Unless you specify this mode, however, only job steps bound to the reservation may use the reserved resources.
 - **REMOVE_ON_IDLE** mode: When you use the **-i** command option, LoadLeveler automatically cancels the reservation when all bound job steps that can run finish running. Using this mode is efficient because it prevents LoadLeveler from wasting reserved resources when no jobs are available to use them. Selecting this mode is especially useful for workloads that will run unattended.

Additional rules apply to the use of these options; see “llmkres - Make a reservation” on page 431 for details.

Alternatives:

- Use the **ll_make_reservation** and the **ll_init_reservation_param** subroutines in a program.
- Use the LoadLeveler GUI by selecting **File ► Reservations ► Create a reservation**.

Tips:

- If your user ID is not authorized to create reservations but you are member of a group with authority to create reservations, you must use the **-g** option to specify the name of the authorized group on the **llmkres** command.

Working with reservations

- Only reservations in waiting and in use are counted toward the limit of allowed reservations set through the **max_reservations** keyword. LoadLeveler does not count reservations that already have ended or are in the process of being canceled.
- Although you may create more than one reservation for a particular node or set of nodes, only one of those reservations may be active at a time. If LoadLeveler determines that the reservation you are requesting will overlap with another reservation, LoadLeveler fails the create request. No reservation periods for the same set of machines can overlap.

If the create request is successful, LoadLeveler assigns and returns to the owner a unique reservation identifier, in the form *host.rid.r*, where:

- host** The name of the machine which assigned the reservation identifier.
- rid** A number assigned to the reservation by LoadLeveler.
- r** The letter **r** is used to distinguish a reservation identifier from a job step identifier.

The following are examples of reservation identifiers:

```
c94n16.80.r  
c94n06.1.r
```

For details about the LoadLeveler interfaces for creating reservations, see:

- “llmkres - Make a reservation” on page 431.
- “ll_make_reservation subroutine” on page 609 and “ll_init_reservation_param subroutine” on page 608.

Submitting jobs to run under a reservation

LoadLeveler administrators, reservation owners, and authorized users may submit jobs to run under a reservation. You may bind both batch and interactive POE job steps to a reservation, both before a reservation starts or while it is active.

Before you begin:

- If you are a reservation owner and used the **-f** or **-j** options on the **llmkres** command when you created the reservation, you do not have to perform the steps listed in Table 51 on page 203. Those command options automatically bind the job steps to the reservation. To find out whether a particular job step is bound to a reservation, use the command **llq -l** and check the listing for a reservation ID.
- To find out which reservation IDs you may use, check with your LoadLeveler administrator, or enter the command **llqres -l** and check the names in the Users or Groups fields (under the Modification time field) in the output listing. If your user name or a group name to which you belong appears in these output fields, you are authorized to use the reservation.
- LoadLeveler cannot guarantee that certain resources will be available during a reservation period. If you submit job steps that require these resources, LoadLeveler will bind the job steps to the reservation, but will not dispatch them unless the resources become available during the reservation. These resources include:
 - Specific nodes that were not reserved under this reservation.
 - Floating consumable resources for a cluster.
 - Resources that are not released through preemption, such as virtual memory and adapters.

Also, your job step will be bound to the reservation but will remain idle when the job step requires more nodes than the number of reserved nodes.

- Whether bound job steps are successfully dispatched depends not only on resource availability, but also on administration file keywords that set maximum numbers, including:
 - **max_jobs_scheduled**
 - **maxidle**
 - **maxjobs**
 - **maxqueued**

If LoadLeveler determines that scheduling a bound job will exceed one or more of these configured limits, your job will remain idle unless conditions permit scheduling at a later time during the reservation period.

Table 51. Instructions for submitting a job to run under a reservation

To bind this type of job:	Use these instructions:
Already submitted jobs	<p>Use the llbind command</p> <p>Alternatives:</p> <ul style="list-style-type: none"> • Use the ll_bind_reservation subroutine in a program. • From the Jobs window, use the LoadLeveler GUI by selecting one or more jobs, then selecting Actions ► Bind to reservation, and selecting the ID of the reservation to which you want to bind these jobs. <p>Result: LoadLeveler either sets the reservation ID for each job step that can be bound to the reservation, or sends a failure notification for the bind request.</p>
A new, unsubmitted job	<ol style="list-style-type: none"> 1. Specify the reservation ID through the LL_RES_ID environment variable. Tip: You may examine but cannot modify this environment variable using a job command filter. 2. Use the llsubmit command to submit the job. <p>Result: If the job can be bound to the requested reservation, LoadLeveler sets the reservation ID for each job step that can be bound to the reservation. Otherwise, if the job step cannot be bound to the reservation, LoadLeveler changes the job state to NotQueued. To change the job step's state to Idle, issue the llbind -r command.</p>

Use the **llqres** command or **llq** command with the **-I** option to check the success or failure of the binding request for each job step.

For details about the LoadLeveler interfaces for submitting jobs under reservations, see:

- “llbind - Bind job steps to a reservation” on page 389.
- “ll_bind subroutine” on page 601.
- “llsubmit - Submit a job” on page 494.

Removing bound jobs from the reservation

LoadLeveler administrators, reservation owners, and authorized users may use the **llbind** command to unbind one or more existing jobs from a reservation.

Alternatives:

- Use the **ll_bind_reservation** subroutine in a program.

Working with reservations

- From the Jobs window, use the LoadLeveler GUI by selecting one or more jobs, then selecting **Actions ► Unbind from reservation**, and selecting the ID of the reservation from which you want to unbind these jobs.

Result: LoadLeveler either unbinds the jobs from the reservation, or sends a failure notification for the unbind request. Use the **llqres** or **llq** command to check the success or failure of the remove request.

For details about the LoadLeveler interfaces for removing bound jobs from the reservation, see:

- “llbind - Bind job steps to a reservation” on page 389.
- “ll_bind subroutine” on page 601.

Querying existing reservations

Any LoadLeveler administrator or user can issue the **llqres** and **llq** commands to request specific information about reservations:

- Various options are available to filter reservations to be displayed.
- To show details of specific reservations, use the **llqres** command with the **-l** option.
- To show job steps that are bound to specific reservations, use the **llq** command with the **-R** option.

Alternative: Use the LoadLeveler GUI by selecting **File ► Reservations**.

For details about:

- Reservation attributes and **llqres** command syntax, see “llqres - Query a reservation” on page 468.
- **llq** command syntax, see “llq - Query job status” on page 449.

Modifying existing reservations

Only administrators and reservation owners may use the **llchres** command to modify one or more attributes of a reservation. Certain attributes cannot be changed after a reservation has become active. Typical uses for the **llchres** command include the following:

- Using the command **llchres -U +newuser1 newuser2** to allow additional users to submit jobs to the reservation.
- If a reservation was made through the command **llmkres -h free** but LoadLeveler cannot include a particular node because it is down, you can use the command **llchres -h +node** to add the node to the reserved node list when that node becomes available again.
- If a reserved node is down after the reservation becomes active, a LoadLeveler administrator can use:
 - The command **llchres -h -node** to remove that node from the reservation.
 - The command **llchres -h +1** to add another node to the reservation.

Alternatives:

- Use the **ll_change_reservation** subroutine in a program.
- Use the LoadLeveler GUI by selecting **File ► Reservations ► Modify a reservation**.

For details about the LoadLeveler interfaces for modifying reservations, see:

- “llchres - Change attributes of a reservation” on page 395.
- “ll_change_reservation subroutine” on page 604.

Canceling existing reservations

Only administrators and reservation owners may use the `llrmres` command to cancel one or more reservations.

Alternatives:

- Use the `ll_remove_reservation` subroutine in a program.
- Use the LoadLeveler GUI by selecting **File ▶ Reservations ▶ Cancel a reservation**.

Result: If the cancel request can be granted, LoadLeveler:

1. Unbinds all jobs associated with the reservation to be removed.
2. Removes the reservation.

Use the `llqres` command to check the success or failure of the remove request.

For details about the LoadLeveler interfaces for canceling reservations, see:

- “`llrmres` - Cancel a reservation” on page 474.
- “`ll_remove_reservation` subroutine” on page 612.

Submitting jobs requesting scheduling affinity

A user can request that a job use scheduling affinity by setting the `RSET` job command file keyword. Specify `RSET` with a value of:

- `RSET_MCM_AFFINITY` to have LoadLeveler schedule the job to machines where `RSET_SUPPORT` is enabled with a value of `RSET_MCM_AFFINITY`.
- `RSET_CONSUMABLE_CPUS` to have LoadLeveler schedule the job to machines where `RSET_SUPPORT` is enabled with a value of `RSET_CONSUMABLE_CPUS`.
- `user_defined_rset` to have LoadLeveler schedule the job to machines where `RSET_SUPPORT` is enabled with a value of `RSET_USER_DEFINED`; `user_defined_rset` is the name of a valid user-defined RSet.

Specifying the `RSET` job command file keyword defaults to requesting memory affinity as a requirement and adapter affinity as a preference. Scheduling affinity options can be customized by using the job command file keyword `MCM_AFFINITY_OPTIONS`. For more information on these keywords, see “Job command file keyword descriptions” on page 336.

Note: If a job specifies memory or adapter affinity scheduling as a requirement, LoadLeveler will only consider machines where `RSET_SUPPORT` is set to `RSET_MCM_AFFINITY`. If there are not enough machines satisfying the memory affinity requirements, the job will stay in the idle state.

Submitting and monitoring jobs in a LoadLeveler multicluster

Table 52 shows the subtasks and associated instructions for submitting and monitoring jobs in a LoadLeveler multicluster:

Table 52. Submitting and monitoring jobs in a LoadLeveler multicluster

Subtask	Associated instructions (see . . .)
Prepare and submit a job in the LoadLeveler multicluster	“Steps for submitting jobs in a LoadLeveler multicluster environment” on page 206

Submitting jobs requesting scheduling affinity

Table 52. Submitting and monitoring jobs in a LoadLeveler multicluster (continued)

Subtask	Associated instructions (see . . .)
Display information about a job in the LoadLeveler multicluster environment	<ul style="list-style-type: none">• Use the llq -X cluster_name command to display information about jobs on remote clusters.• Use llq -x -d to display the user's job command file keyword statements.• Use llq -X cluster_name -l to obtain multicluster specific information.
Transfer an idle job from one cluster to another cluster	Use the llmovejob command, which is described in "llmovejob - Move a single idle job from the local cluster to another cluster" on page 440.

Steps for submitting jobs in a LoadLeveler multicluster environment

In a multicluster environment, you may specify either of the following:

- That a job is to run on a particular cluster.
- That LoadLeveler is to decide which cluster is best from the list of clusters, based on an administrator-defined metric. If **any** is specified, the job is submitted to the best cluster, based on an administrator-defined metric.

The following procedure explains how to prepare your job to be submitted in the multicluster environment.

Before you begin: You need to know that:

- Only batch jobs are supported in the LoadLeveler multicluster environment. LoadLeveler will fail any interactive jobs that you attempt to submit in a multicluster environment.
- LoadLeveler assigns all steps of a multistep job to the same cluster.
- Job identifiers are assigned by the local cluster and are retained by the job regardless of what cluster the job executes in.
- Remote jobs are subjected to the same configuration checks as locally submitted jobs. Examples include account validation, class limits, include lists, and exclude lists.

Perform the following steps to submit jobs in a LoadLeveler multicluster environment.

1. If files used by your job need to be copied between clusters, you must specify the job files to be copied from the local to the remote cluster in the job command file. Use the **cluster_input_file** and **cluster_output_file** keywords to specify these files.

Rules:

- Any local file specified for copy must be accessible from the local gateway Schedd machines. Input files must be readable. Directories and permissions must be in place to write output files.
- Any remote file specified for copy must be accessible from the remote gateway Schedd machines. Directories and permissions must be in place to write input files. Output files must be readable when the job terminates.
- To copy more than one file, these keywords can be specified multiple times.

Alternative: Use the LoadLeveler GUI by selecting **File ► Build a Job** and editing the **Cluster Input File** and **Cluster Output File** fields.

Tip: Each instance of these keywords allows you to specify a single local file and a single remote file. If your job requires copying multiple files (for example, all files in a directory), you may want to use a procedure to consolidate the multiple files into a single file rather than specify multiple `cluster_file` statements in the job command file. The following is an example of how you could consolidate input files:

- a. Use the `tar` command to produce a single tar file from multiple files.
 - b. On the `cluster_input_file` keyword, specify the file that resulted from the `tar` command processing.
 - c. Modify your job command file such that it uses the `tar` command to restore the multiple files from the tar file prior to invoking your application.
2. In the job command file, specify the clusters to which LoadLeveler may submit the job. The `cluster_list` keyword is a blank-delimited list of cluster names or the reserved word `any` where:
 - A single cluster name indicates that the job is to be submitted to that cluster.
 - A list of multiple cluster names indicates that the job is to be submitted to one of the clusters as determined by the installation exit `CLUSTER_METRIC`.
 - The reserved word `any` indicates that the job is to be submitted to any cluster defined by the installation exit `CLUSTER_METRIC`.

Alternatives:

- a. From the Jobs window, use the LoadLeveler GUI by selecting **Tools ▶ Set Cluster**.
 - b. You can specify the clusters to which LoadLeveler can submit your job on the `llsubmit` command using the `-X` option. To do the same using the LoadLeveler GUI, select **File ▶ Submit a Job**, then select **Change Cluster** and **Submit**.
3. Use the `llsubmit` command or the LoadLeveler GUI to submit the job.

Note: Using Set Cluster when submitting from the Build a Job window, `cluster_list` is set in the generated job command file, while when using the Submit a Job window, the `-X` option is set on the submit.

Tip: You may use the `-X` option on the `llsubmit` command to specify:
`-X {cluster_list | any}`

Is a blank-delimited list of cluster names or the reserved word `any` where:

- A single cluster name indicates that the job is to be submitted to that cluster.
- A list of multiple cluster names indicates that the job is to be submitted to one of the clusters as determined by the installation exit `CLUSTER_METRIC`.
- The reserved word `any` indicates that the job is to be submitted to any cluster defined by the installation exit `CLUSTER_METRIC`.

The `llsubmit` command displays the assigned local outbound Schedd, the assigned remote inbound Schedd, the scheduling cluster and the job identifier when the remote job has been successfully submitted. Use the `-q` flag to stop these additional messages from being displayed.

Note: If a remote job is submitted with a list of clusters or the reserved word `any` and the installation exit `CLUSTER_METRIC` is not specified, the remote job is not submitted.

Submitting jobs requesting scheduling affinity

When you are done, you can use commands to display information about the submitted job; for example:

- Use `llq -l -X cluster_name -j job_id` where `cluster_name` and `job_id` were displayed by the `llsubmit` command to display information about the remote job.
- Use `llq -l -X cluster_list` to display the long listing about jobs, including scheduling cluster, submitting cluster, user-requested cluster, cluster input and output files.
- Use `llq -X all` to display information about all jobs in all configured clusters.

You can also use the LoadLeveler GUI to display information about your job. When you invoke `xloadl` in a multicluster environment, you will be prompted to open a window for other configured clusters. The title for remote clusters is in the form `local_cluster→remote_cluster`, where `local_cluster` is the cluster of the machine where you started `xloadl`, and `remote_cluster` is a remote cluster you selected. The jobs window contains the jobs running on that remote cluster.

Submitting and monitoring Blue Gene jobs

The following procedure explains how to prepare your job to be submitted to the Blue Gene system. The submission of Blue Gene jobs is similar to the submission of other job types.

Before you begin: You need to know that checkpointing Blue Gene jobs is not currently supported.

Tip: Use the `llstatus` command to check if Blue Gene support is enabled and whether Blue Gene is currently present. The `llstatus` command will display:

```
The BACKFILL scheduler with Blue Gene support is in use
```

```
Blue Gene is present
```

```
when Blue Gene is support is enabled and Blue Gene is currently present
```

Perform the following steps to submit Blue Gene jobs:

1. In the job command file, set the job type to Blue Gene by specifying:

```
#@job_type = bluegene
```
2. Specify the size or shape of the Blue Gene job or the Blue Gene partition in which the job will run.
 - The size of the Blue Gene job can be specified by using the job command file keyword `bg_size` to specify the size of the job. For more information, see the detailed description of the `bg_size` keyword .
 - The shape of the Blue Gene job can be specified by using the job command file keyword `bg_shape` to specify the shape of the job. If you require the specific shape you specified, you may wish to specify the `bg_rotate` keyword to false. For more information on these keywords, see the detailed descriptions of the `bg_shape` keyword and `bg_rotate` keyword.
 - The partition in which the Blue Gene job is run can be specified using the `bg_partition` job command file keyword. For more information, see the detailed description of the `bg_partition` keyword.
3. Specify any other job command file keywords you require, including the `bg_connection` and `bg_requirements` Blue Gene job command file keywords. See “Job command file keyword descriptions” on page 336 for more information on job command file keywords.
4. Upon completing your job command file, submit the job using the `llsubmit` command.

Submitting jobs requesting scheduling affinity

Alternative: Use the Build a Job window in the LoadLeveler GUI and select the radio button for Blue Gene under Job Type. Then the Blue Gene button can be clicked to request Blue Gene job request attributes. The generated job command file can be saved or submitted.

When you are done, you can use the `llq -b` command to display information about Blue Gene jobs in short form. For more information see “llq - Query job status” on page 449.

You can also use the LoadLeveler GUI to display the status of a Blue Gene job. From the Jobs window, after selecting one or more jobs, select **Actions ► Blue Gene Job Status**.

Example:

The following is a sample job command file for a Blue Gene job:

```
# @ job_name           = bgsample
# @ job_type           = bluegene
# @ comment            = "BGL Job by Size"
# @ error              = $(job_name).err
# @ output             = $(job_name).out
# @ environment        = COPY_ALL;
# @ wall_clock_limit   = 200:00,200:00
# @ notification       = always
# @ notify_user        = sam
# @ bg_size            = 1024
# @ bg_connection      = torus
# @ class              = 2bp
# @ queue
/usr/bin/mpirun -exe /bgscratch/sam/com -verbose 2 -args "-o 100 -b 64 -r"
```

Submitting jobs requesting scheduling affinity

Chapter 9. Managing submitted jobs

Table 53 lists the tasks and sources of additional information for managing LoadLeveler jobs.

Table 53. Roadmap of user tasks for managing submitted jobs

To learn about:	Read the following:
Displaying information about a submitted job or its environment	<ul style="list-style-type: none">• “Querying the status of a job”• “Working with machines”• “Displaying currently available resources” on page 212• “llclass - Query class information” on page 403• “llq - Query job status” on page 449• “llstatus - Query machine status” on page 477• “llsummary - Return job resource information for accounting” on page 496
Changing the priority of a submitted job	<ul style="list-style-type: none">• “Setting and changing the priority of a job” on page 212• “llmodify - Change attributes of a submitted job step” on page 435
Changing the state of a submitted job	<ul style="list-style-type: none">• “Placing and releasing a hold on a job” on page 213• “Canceling a job” on page 214• “llhold - Hold or release a submitted job” on page 426• “llcancel - Cancel a submitted job” on page 392
Checkpointing a submitted job	<ul style="list-style-type: none">• “Checkpointing a job” on page 214• “llckpt - Checkpoint a running job step” on page 400

Querying the status of a job

Once you submit a job, you can query the status of the job to determine, for example, if it is still in the queue or if it is running. You also receive other job status related information such as the job ID and the submitting user ID. You can query the status of a LoadLeveler job either by using the GUI or the **llq** command. For an example of querying the status of a job, see Chapter 10, “Example: Using commands to build, submit, and manage jobs,” on page 215.

Querying the status of a job using a submit-only machine: In addition to allowing you to submit and cancel jobs, a submit-only machine allows you to query the status of jobs. You can query a job using either the submit-only version of the GUI or by using the **llq** command. For information on **llq**, see “llq - Query job status” on page 449.

Working with machines

You can perform the following types of tasks related to machines:

- **Display machine status**

When you submit a job to a machine, the status of the machine automatically appears in the Machines window on the GUI. This window displays machine related information such as the names of the machines running jobs, as well as

the machine's architecture and operating system. For detailed information on one or more machines in the cluster, you can use the Details option on the Actions pull-down menu. This will provide you with a detailed report that includes information such as the machine's state and amount of installed memory.

For an example of displaying machine status, see Chapter 10, "Example: Using commands to build, submit, and manage jobs," on page 215.

- **Display central manager**

The LoadLeveler administrator designates one of the machines in the LoadLeveler cluster as the central manager. When jobs are submitted to any machine, the central manager is notified and decides where to schedule the jobs. In addition, it keeps track of the status of machines in the cluster and jobs in the system by communicating with each machine. LoadLeveler uses this information to make the scheduling decisions and to respond to queries.

Usually, the system administrator is more concerned about the location of the central manager than the typical end user but you may also want to determine its location. One reason why you might want to locate the central manager is if you want to browse some configuration files that are stored on the same machine as the central manager.

- **Display public scheduling machines**

Public scheduling machines are machines that participate in the scheduling of LoadLeveler jobs on behalf of users at submit-only machines and users at other workstations that are not running the Schedd daemon. You can find out the names of all these machines in the cluster.

Submit-only machines allow machines that are not part of the LoadLeveler cluster to submit jobs to the cluster for processing.

Displaying currently available resources

The LoadLeveler user can get information about currently available resources by using the **llstatus** command with either the **-F**, or **-R** options. The **-F** option displays a list of all of the floating resources associated with the LoadLeveler cluster. The **-R** option lists all of the consumable resources associated with all of the machines in the LoadLeveler cluster. The user can specify a hostlist with the **llstatus** command to display only the consumable resources associated with specific hosts.

Setting and changing the priority of a job

LoadLeveler uses the priority of a job to determine its position among a list of all jobs waiting to be dispatched. LoadLeveler schedules jobs based on the *adjusted system priority*, which takes in account both system priority and user priority:

User priority

Every job has a user priority associated with it. A job with a higher priority runs before a job with a lower priority (when both jobs are owned by the same user). You can set this priority through the **user_priority** keyword in the job command file, and modify it through the **llprio** command. See "llprio - Change the user priority of submitted job steps" on page 447 for more information.

System priority

Every job has a system priority associated with it. Administrators can set this priority in the configuration file using the **SYSPRIO** keyword

expression. The **SYSPRIO** expression can contain class, group, and user priorities, as shown in the following example:

```
SYSPRIO : (ClassSysprio * 100) + (UserSysprio * 10) + (GroupSysprio * 1) - (QDate)
```

The **SYSPRIO** expression is evaluated by LoadLeveler to determine the overall system priority of a job. To determine which jobs to run first, LoadLeveler does the following:

1. Assigns a system priority value when the negotiator adds the new job to the queue of jobs eligible for dispatch.
2. Orders jobs first by system priority.
3. Assigns jobs belonging to the same user and the same class an adjusted system priority, which takes all the system priorities and orders them by user priority. Jobs with a higher adjusted system priority are scheduled ahead of jobs with a lower adjusted system priority.

Only administrators may modify the system priority through the **llmodify** command with the **-s** option. See “llmodify - Change attributes of a submitted job step” on page 435 for more information.

Example: How does a job’s priority affect dispatching order?

To understand how a job’s priority affects dispatching order, consider the sample jobs in Table 54, which lists the priorities assigned to jobs submitted by two users, Rich and Joe. Two of the jobs belong to Joe, and three belong to Rich. User Joe has two jobs (Joe1 and Joe2) in Class A with SYSPRIOs of 9 and 8 respectively. Since Joe2 has the higher user priority (20), and because both of Joe’s jobs are in the same class, Joe2’s priority is swapped with that of Joe1 when the adjusted system priority is calculated. This results in Joe2 getting an adjusted system priority of 9, and Joe1 getting an adjusted system priority of 8. Similarly, the Class A jobs belonging to Rich (Rich1 and Rich3) also have their priorities swapped. The priority of the job Rich2 does not change, since this job is in a different class (Class B).

Table 54. How LoadLeveler handles job priorities

Job	User Priority	System Priority (SYSPRIO)	Class	Adjusted System Priority
Rich1	50	10	A	6
Joe1	10	9	A	8
Joe2	20	8	A	9
Rich2	100	7	B	7
Rich3	90	6	A	10

Placing and releasing a hold on a job

You may place a hold on a job and thereby cause the job to remain in the queue until you release it.

There are two types of holds: a user hold and a system hold. Both you and your LoadLeveler administrator can place and release a user hold on a job. Only a LoadLeveler administrator, however, can place and release a system hold on a job.

You can place a hold on a job or release the hold either by using the GUI or the **llhold** command. For examples of holding and releasing jobs, see Chapter 10, “Example: Using commands to build, submit, and manage jobs,” on page 215.

As a user or an administrator, you can also use the **startdate** keyword to place a hold on a job. This keyword allows you to specify when you want to run a job.

Canceling a job

You can cancel one of your jobs that is either running or waiting to run by using either the GUI or the **llcancel** command. You can use **llcancel** to cancel LoadLeveler jobs, including jobs from a submit-only machine.

For more information about the **llcancel** command, see “**llcancel** - Cancel a submitted job” on page 392.

Checkpointing a job

Checkpointing is a method of periodically saving the state of a job so that, if for some reason, the job does not complete, it can be restarted from the saved state. Checkpoints can be taken either under the control of the user application or external to the application.

On AIX only, the LoadLeveler API **ll_init_ckpt** is used to initiate a serial checkpoint from the user application. For initiating checkpoints from within a parallel application, the API **mpc_init_ckpt** should be used. These APIs allow the writer of the application to determine at what points in the application it would be appropriate to save the state of the job. To enable parallel applications to initiate checkpointing, you must use the APIs provided with the Parallel Environment (PE) program. For information on parallel checkpointing, see *IBM Parallel Environment for AIX: Operation and Use, Volume 1*.

It is also possible to checkpoint a program running under LoadLeveler outside the control of the application. There are several ways to do this:

- Use the **llckpt** command to initiate checkpoint for a specific job step. See “**llckpt** - Checkpoint a running job step” on page 400 for more information.
- Checkpoint from a program which invokes the **ll_ckpt** API to initiate checkpoint of a specific job step. See “**ll_ckpt** subroutine” on page 512 for more information.
- Have LoadLeveler automatically checkpoint all running jobs that have been enabled for checkpoint. To enable this automatic checkpoint, specify **checkpoint = interval** in the job command file.
- As the result of an **llctl flush** command.

Note: For interactive parallel jobs, the environment variable **CHECKPOINT** must be set to **yes** in the environment prior to starting the parallel application or the job will not be enabled for checkpoint. For more information see *IBM Parallel Environment for AIX: MPI Programming Guide*.

Chapter 10. Example: Using commands to build, submit, and manage jobs

The following procedure presents a series of simple tasks that a user might perform using commands. For additional information about individual commands noted in the procedure, see Chapter 16, “Commands,” on page 385.

1. Build your job command file by using a text editor to create a script file. Into the file enter the name of the executable, other keywords designating such things as output locations for messages, and the necessary LoadLeveler statements, as shown in Figure 35:

```
# This job command file is called longjob.cmd. The
# executable is called longjob, the input file is longjob.in,
# the output file is longjob.out, and the error file is
# longjob.err.
#
# @ executable = longjob
# @ input      = longjob.in
# @ output     = longjob.out
# @ error      = longjob.err

# @ queue
```

Figure 35. Building a job command file

2. You can optionally edit the job command file you created in step 1.
3. To submit the job command file that you created in step 1, use the **llsubmit** command:

```
llsubmit longjob.cmd
```

LoadLeveler responds by issuing a message similar to:

```
submit: The job "wizard.22" has been submitted.
```

Where *wizard* is the name of the machine to which the job was submitted and *22* is the job identifier (ID). You may want to record the identifier for future use (although you can obtain this information later if necessary).

4. To display the status of the job you just submitted, use the **llq** command. This command returns information about all jobs in the LoadLeveler queue:

```
llq wizard.22
```

Where *wizard* is the machine name to which you submitted the job, and *22* is the job ID. You can also query this job using the command **llq wizard.22.0**, where *0* is the step ID.

5. To change the priority of a job, use the **llprio** command. To increase the priority of the job you submitted by a value of 10, enter:

```
llprio +10 wizard.22.0
```

You can change the user priority of a job that is in the queue or one that is running. This only affects jobs belonging to the same user and the same class. If you change the priority of a job in the queue, the job’s priority increases or decreases in relation to your other jobs in the queue. If you change the priority of a job that is running, it does not affect the job while it is running. It only affects the job if the job re-enters the queue to be dispatched again. For more information, see “Setting and changing the priority of a job” on page 212.

6. To place a temporary hold on a job in a queue, use the **llhold** command. This command only takes effect if jobs are in the Idle or NotQueued state. To place a hold on *wizard.22.0*, enter:

```
llhold wizard.22.0
```

7. To release the hold you placed in step 6, use the **llhold** command:

```
llhold -r wizard.22.0
```

8. To display the status of the machine to which you submitted a job, use the **llstatus** command:

```
llstatus -l wizard
```

9. To cancel *wizard.22.0*, use the **llcancel** command:

```
llcancel wizard.22.0
```

Chapter 11. Using LoadLeveler's GUI to build, submit, and manage jobs

This topic describes tasks a user may need to accomplish through the graphical user interface (GUI). You do not have to perform the tasks in the order listed. You may perform certain tasks before others without any difficulty; however, some tasks must be performed prior to others for succeeding tasks to work. For example, you cannot submit a job if you do not have a job command file that you built using either the GUI or an editor.

The tasks included in this topic are listed in Table 55.

Table 55. User tasks available through the GUI

Subtask	Associated information (see...)
Building and submitting jobs	<ul style="list-style-type: none">• "Building jobs"• "Editing the job command file" on page 229• "Submitting a job command file" on page 230
Obtaining job status	<ul style="list-style-type: none">• "Displaying and refreshing job status" on page 231• "Specifying which jobs appear in the Jobs window" on page 237• "Sorting the Jobs window" on page 232
Managing a submitted job	<ul style="list-style-type: none">• "Changing the priority of your jobs" on page 232• "Placing a job on hold" on page 233• "Releasing the hold on a job" on page 233• "Canceling a job" on page 233
Working with machines	<ul style="list-style-type: none">• "Displaying and refreshing machine status" on page 235• "Specifying which machines appear in Machines window" on page 238• "Sorting the Machines window" on page 236• "Finding the location of the central manager" on page 237• "Finding the location of the public scheduling machines" on page 237
Saving LoadLeveler messages in a file	"Saving LoadLeveler messages in a file" on page 238

Building jobs

From the Jobs window:

SELECT **File → Build a Job**

▲ The dialog box shown in Figure 36 on page 218 appears:

Using the GUI

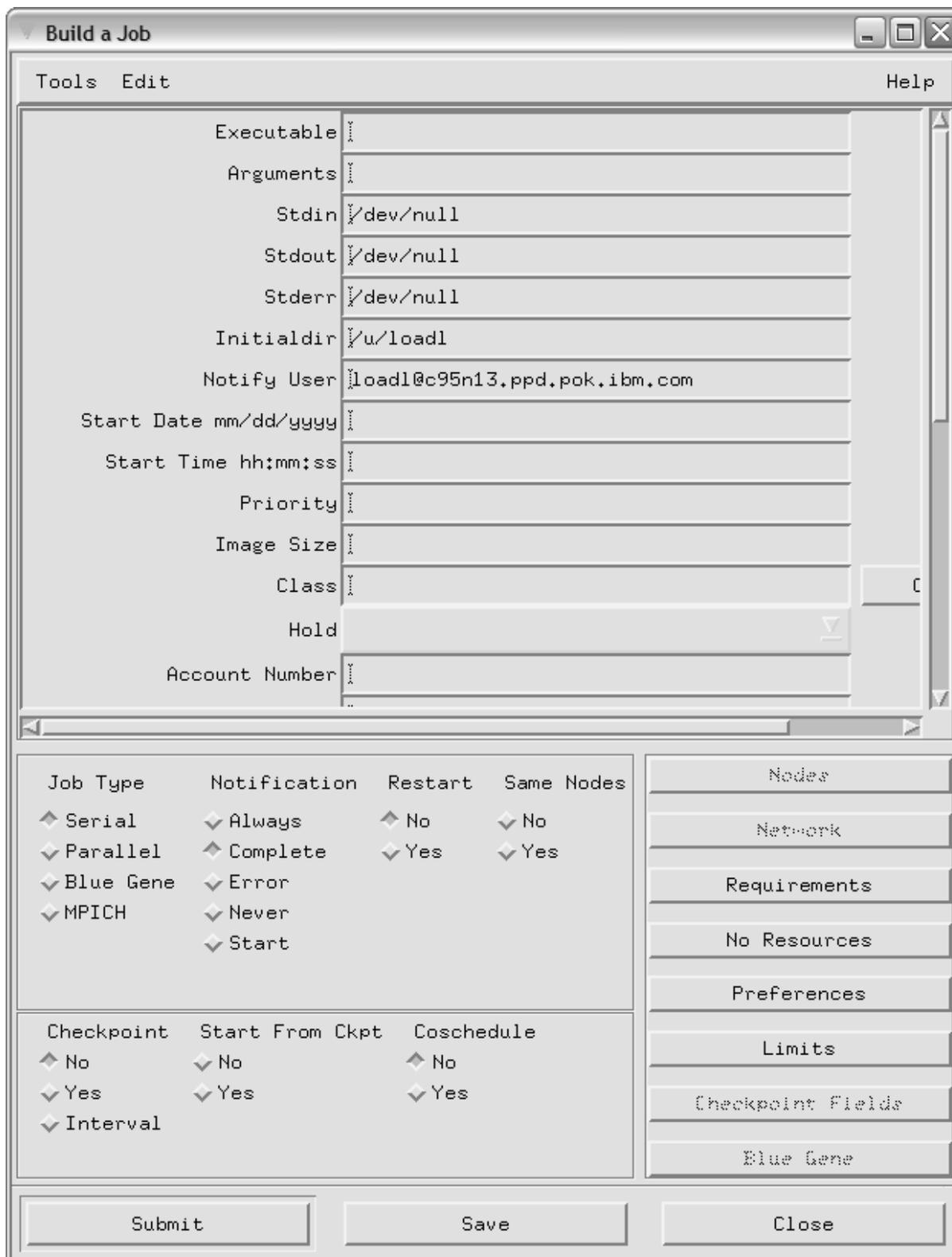


Figure 36. LoadLeveler build a job window

Complete those fields for which you want to override what is currently specified in your **skel.cmd** defaults file. Sample **skel.cmd** and **mcluster_skel.cmd** files are found in the samples subdirectory

of the release directory. You can update this file to define defaults for your site, and then update the `*skelfile` resource in `Xloadl` to point to your new `skel.cmd` file. If you want a personal defaults file, copy `skel.cmd` to one of your directories, edit the file, and update the `*skelfile` resource in `.Xdefaults`. Table 56 shows the fields displayed in the Build a Job window:

Table 56. GUI fields and input

Field	Input
Executable	Name of the program to run. It must be an executable file. Optional. If omitted, the command file is executed as if it were a shell script.
Arguments	Parameters to pass to the program. Required only if the executable requires them.
Stdin	Filename to use as standard input (stdin) by the program. Optional. The default is <code>/dev/null</code> .
Stdout	Filename to use as standard output (stdout) by the program. Optional. The default is <code>/dev/null</code> .
Stderr	Filename to use as standard error (stderr) by the program. Optional. The default is <code>/dev/null</code> .
Cluster Input File	A comma delimited local and remote pathname pair, representing the local file to copy to the remote location. If you have more than one pair to enter, the More button will display a Cluster Input Files input window. Optional. The default is no files are copied.
Cluster Output File	A comma delimited local and remote pathname pair, representing the local file destination to copy to the remote file into. If you have more than one pair to enter, the More button will display a Cluster Output Files input window. Optional. The default is no files are copied.
Initialdir	Initial directory. LoadLeveler changes to this directory before running the job. Optional. The default is your current working directory.
Notify User	User id of person to notify regarding status of submitted job. Optional. The default is your userid.
StartDate	Month, day, and year in the format <code>mm/dd/yyyy</code> . The job will not start before this date. Optional. The default is to run the job as soon as possible.
StartTime	Hour, minute, second in the format <code>hh:mm:ss</code> . The job will not start before this time. Optional. The default is to run the job as soon as possible. If you specify <code>StartTime</code> but not <code>StartDate</code> , the default <code>StartDate</code> is the current day. If you specify <code>StartDate</code> but not <code>StartTime</code> , the default <code>StartTime</code> is <code>00:00:00</code> . This means that the job will start as soon as possible on the specified date.

Table 56. GUI fields and input (continued)

Field	Input						
Priority	<p>Number between 0 and 100, inclusive.</p> <p>Optional. The default is 50.</p> <p>This is the user priority. For more information on this priority, refer to “Setting and changing the priority of a job” on page 212.</p>						
Image size	<p>Number in kilobytes that reflects the maximum size you expect your program to grow to as it runs.</p> <p>Optional.</p>						
Class	<p>Class name. The job will only run on machines that support the specified class name. Your system administrator defines the class names.</p> <p>Optional:</p> <ul style="list-style-type: none"> • Press the Choices button to get a list of available classes. • Press the Details button under the class list to obtain long listing information about classes. 						
Hold	<p>Hold status of the submitted job. Permitted values are:</p> <table> <tr> <td>user</td> <td>User hold</td> </tr> <tr> <td>system</td> <td>System hold (only valid for LoadLeveler administrators)</td> </tr> <tr> <td>usersys</td> <td>User and system hold (only valid for LoadLeveler administrators)</td> </tr> </table> <p>Note: The default is a no-hold state.</p>	user	User hold	system	System hold (only valid for LoadLeveler administrators)	usersys	User and system hold (only valid for LoadLeveler administrators)
user	User hold						
system	System hold (only valid for LoadLeveler administrators)						
usersys	User and system hold (only valid for LoadLeveler administrators)						
Account Number	<p>Number associated with the job. For use with the llacctmrg and llsummary commands for acquiring job accounting data.</p> <p>Optional. Required only if the ACCT keyword is set to A_VALIDATE in the configuration file.</p>						
Environment	<p>Your initial environment variables when your job starts. Separate environment specifications with semicolons.</p> <p>Optional.</p>						
Copy Environment	<p>All or Master, to indicate whether the environment variables specified in the keyword Environment are copied to all nodes or just to the master node of a parallel job.</p> <p>Optional.</p>						
Shell	<p>The name of the shell to use for the job.</p> <p>Optional. If not specified, the shell used in the owner’s password file entry is used. If none is specified, /bin/sh is used.</p>						
Group	<p>The LoadLeveler group name to which the job belongs.</p> <p>Optional.</p>						
Step Name	<p>The name of this job step.</p> <p>Optional.</p>						

Table 56. GUI fields and input (continued)

Field	Input
Node Usage	How the node is used. Permitted values are: shared The node can be shared with other tasks of other job steps. This is the default. not shared The node cannot be shared. slice not shared Has the same meaning as not shared . It is provided for compatibility.
Dependency	A Boolean expression defining the relationship between the job steps. Optional.
Large Page	Whether or not the job step requires Large Page memory. yes Use Large Page memory if available, otherwise use regular memory. mandatory Use of Large Page memory is mandatory. no Do not use Large Page memory.
Bulk Transfer	Indicates to the communication subsystem whether it should use the bulk transfer mechanism to communicate between tasks. yes Use bulk transfer. no Do not use bulk transfer. Optional.
Rset	What type of RSet support is requested. Permitted values are: rset_mcm_affinity Requests scheduling affinity. Use the MCM options button to specify task allocation method, memory affinity preference or requirement, and adapter affinity preference or requirement. rset_consumable_cpus Requests nodes where rset_support is set to rset_consumable_cpus . <i>rset_name</i> Requests a user defined RSet and nodes with rset_support set to rset_user_defined . Optional.
Comments	Comments associated with the job. These comments help to distinguish one job from another job. Optional.
SMT	Indicates whether a job requires dynamic simultaneous multithreading (SMT) function. yes The job requires SMT function. no The job does not require SMT function.

Note: The fields that appear in this table are what you see when viewing the Build a Job window. The text in these fields does not necessarily correspond with the keywords listed in “Job command file keyword descriptions” on page 336.

See “Job command file keyword descriptions” on page 336 for information on the defaults associated with these keywords.

SELECT

A Job Type if you want to change the job type.

Your choices are:

- Serial** Specifies a serial job. This is the default.
- Parallel** Specifies a parallel job.
- Blue Gene** Specifies a bluegene job.

	MPICH	Specifies a MPICH job.
		Note that the job type you select affects the choices that are active on the Build A Job window.
	SELECT	a Notification option. Your choices are: Always Notify you when the job starts, completes, and if it incurs errors. Complete Notify you when the job completes. This is the default option as initially defined in the skel.cmd file. Error Notify you if the job cannot run because of an error. Never Do not notify you. Start Notify you when the job starts.
	SELECT	a Restart option. Your choices are: No This job is not restartable. This is the default. Yes Restart the job.
	SELECT	To restart the job on the same nodes from which it was vacated. Your choices are: No Restart the job on any available nodes. Yes Restart the job on the same nodes it ran on previously. This option is valid after a job has been vacated. Note that there is no default for the selection.
	SELECT	a Checkpoint option. Your choices are: No Do not checkpoint the job. This is the default. Yes Yes, checkpoint the job at intervals you determine. See the checkpoint keyword for more information. Interval Yes, checkpoint the job at intervals determined by LoadLeveler. See the checkpoint keyword for more information.
	SELECT	To start from a checkpoint file Your choices are: No Do not start the job from a checkpoint file (start job from beginning). Yes Yes, restart the job from an existing checkpoint file when you submit the job. The file name must be specified by the job command file. The directory name may be specified by the job command file, configuration file, or default location.
	SELECT	Coschedule if you want steps within a job to be scheduled and dispatched at the same time. Your choices are: No Disables coscheduling for your job step. Yes Allows coscheduling to occur for your job step.

Notes:

1. This keyword is not inherited by other job steps.
2. The default is No.
3. The coscheduling function is only available with the BACKFILL scheduler.

SELECT Nodes (available when the job type is parallel)

▲ The Nodes dialog box appears.

Complete the necessary fields to specify node information for a parallel job (see Table 57). Depending upon which model you choose, different fields will be available; any unavailable fields will be desensitized. LoadLeveler will assign defaults for any fields that you leave blank. For more information, see the appropriate job command file keyword (listed in parentheses) in “Job command file keyword descriptions” on page 336.

Table 57. Nodes dialog box

Field	Available in:	Input
Min # of Nodes	Tasks Per Node Model and Tasks with Uniform Blocking Model	Minimum number of nodes required for running the parallel job (node keyword). Optional. The default is one.
Max # of Nodes	Tasks Per Node Model	Maximum number of nodes required for running the parallel job (node keyword). Optional. The default is the minimum number of nodes.
Tasks per Node	Tasks Per Node Model	The number of tasks of the parallel job you want to run per node (tasks_per_node keyword). Optional.
Total Tasks	Tasks with Uniform Blocking Model, and Custom Blocking Model	The total number of tasks of the parallel job you want to run on all available nodes (total_tasks keyword). Optional for Uniform, required for Custom Blocking. The default is one.
Blocking	Custom Blocking Model	The number of tasks assigned (as a block) to each consecutive node until all of a job's tasks have been assigned (blocking keyword)
Task Geometry	Custom Geometry Model	The task ids of each task that you want to run on each node. You can use the “Set Geometry” button for step-by-step directions (task_geometry keyword).

SELECT Close to return to the Build a Job dialog box.

SELECT Network (available when the job type is parallel)

▲ The Network dialog box appears.

The Network dialog box consists of two parts: The top half of the panel is for MPI, and the bottom half is for LAPI. Click on the check box to the left of **MPI** or **LAPI** to activate the part of the panel for which you want to specify network information. If you want to use MPI with LAPI, click on both:

- The MPI check box.
- The check box for **Share windows between MPI and LAPI**.

Complete those fields for which you want to specify network information (see Table 58). For more information, see the **network** keyword description in “Job command file keyword descriptions” on page 336.

Table 58. Network dialog box fields

Field	Input
MPI (MPI/LAPI)	<p>Select:</p> <ul style="list-style-type: none"> • Only the MPI check box to use the Message Passing Interface (MPI) protocol only. • Both the MPI check box and the Share windows between MPI and LAPI check box to use both MPI and the Low-level Application Programming Interface (LAPI) protocols. This selection corresponds to setting the network keyword in the job command file to MPI_LAPI. <p>Optional.</p>
LAPI	<p>Select the LAPI check box to use Low-level Application Programming Interface (LAPI) protocol only.</p> <p>Optional.</p>
Adapter/Network	<p>Select an adapter name or a network type from the list.</p> <p>Required for each protocol you select.</p>
Adapter Usage	<p>Specifies that the adapter is either shared or not shared.</p> <p>Optional. The default is shared.</p>
Communication Mode	<p>Specifies the communication subsystem mode used by the communication protocol that you specify and can be either IP (Internet Protocol) or US (User Space).</p> <p>Optional. The default is IP.</p>
Communication Level	<p>Implies the amount of memory to be allocated to each window for User Space mode. Allocation can be Low, Average, or High. It is ignored by <code>Switch_Network_Interface_For_HPS</code> adapters.</p>
Instances	<p>Specifies the number of windows or IP addresses the communication subsystem should allocate to this protocol.</p> <p>Optional. The default is 1 unless <code>sn_all</code> is specified for network and then the default is max.</p>

Table 58. Network dialog box fields (continued)

Field	Input
rCxt Blocks	The number of user rCxt blocks requested for each window used by the associated protocol. It is recognized only by Switch_Network_Interface_For_HPS adapters. Optional.

SELECT Close to return to the Build a Job dialog box.

SELECT Requirements

▲ The Requirements dialog box appears.

Complete those fields for which you want to specify requirements (see Table 59). Defaults are used for those fields that you leave blank. LoadLeveler dispatches your job only to one of those machines with resources that matches the requirements you specify.

Table 59. Build a job dialog box fields

Field	Input
Architecture (see note 2)	Machine type. The job will not run on any other machine type. Optional. The default is the architecture of your current machine.
Operating System (see note 2)	Operating system. The job will not run on any other operating system. Optional. The default is the operating system of your current machine.
Disk	Amount of disk space in the execute directory. The job will only run on a machine with at least this much disk space. Optional. The default is defined in your local configuration file.
Memory	Amount of memory. The job will only run on a machine with at least this much memory. Optional. The default is defined in your local configuration file.
Large Page Memory	Amount of Large Page Memory, in megabytes. The job step requires at least this much Large Page Memory to run. Optional.
Total Memory	Amount of total (regular and Large Page Memory) in megabytes needed to run the job step. Optional.
Machines	Machine names. The job will only run on the specified machines. Optional.

Table 59. Build a job dialog box fields (continued)

Field	Input
Features	Features. The job will only run on machines with specified features. Optional.
Pool	Specifies the number associated with the pool you want to use. All available pools listed in the administration file appear as choices. The default is to select nodes from any pool.
LoadLeveler Version	Specifies the version of LoadLeveler, in dotted decimal format, on the machine where you want the job to run. For example: 3.3.0.0 specifies that your job will run on a machine running LoadLeveler Version 3.3.0.0 or higher. Optional.
Connectivity	A number from 0.0 through 1.0, representing the average connectedness of the node's managed adapters.
Requirement	Requirements. The job will only run if these requirements are met.

Notes:

1. If you enter a resource that is not available, you will NOT receive a message. LoadLeveler holds your job in the Idle state until the resource becomes available. Therefore, make certain that the spelling of your entry is correct. You can issue `llq -s jobID` to find out if you have a job for which requirements were not met.
2. If you do not specify an architecture or operating system, LoadLeveler assumes that your job can run only on your machine's architecture and operating system. If your job is not a shell script that can be run successfully on any platform, you should specify a required architecture and operating system.

SELECT Close to return to the Build a Job dialog box.

SELECT Resources

▲ The Resources dialog box appears.

This dialog box allows you to set the amount of defined consumable resources required for a job step. Resources with an "*" appended to their names are not in the SCHEDULE_BY_RESOURCES list. For more information, see the resources keyword.

SELECT Close to return to the Build a Job dialog box.

SELECT Preferences

▲ The Preferences dialog box appears.

This dialog box is similar to the Requirements dialog box, with the exception of the Adapter choice, which is not supported as a Preference. Complete the fields for those parameters that you want to specify. These parameters are not binding. For any preferences that you specify, LoadLeveler attempts to find a machine that matches these preferences along with your requirements. If it cannot find the machine, LoadLeveler chooses the first machine that matches the requirements.

SELECT Close to return to the Build a Job dialog box.

SELECT Limits

▲ The Limits dialog box appears.

Complete the fields for those limits that you want to impose upon your job (see Table 60). If you type *copy* in any field except **wall_clock_limit** or **job_cpu_limit**, the limits in effect on the submit machine are used. If you leave any field blank, the default limits in effect for your userid on the machine that runs the job are used. For more information, see “Using limit keywords” on page 83.

Table 60. Limits dialog box fields

Field	Input
CPU Limit	Maximum amount of CPU time that the submitted job can use. Express the amount as: [[hours:]minutes:]seconds[.fraction] For example, 12:56:21 is 12 hours, 56 minutes, and 21 seconds. Optional
Data Limit	Maximum amount of the data segment that the submitted job can use. Express the amount as: integer[.fraction] [units] Optional
Core Limit	Maximum size of a core file. Optional
RSS Limit	Maximum size of the resident set size. It is the largest amount of physical memory a user's process can allocate. Optional
File Limit	Maximum size of a file that is created. Optional
Stack Limit	Maximum size of the stack. Optional
Job CPU Limit	Maximum total CPU time to be used by all processes of a serial job step or if a parallel job, then this is the total CPU time for each LoadL_starter process and its descendants for each job step of a parallel job. Optional
Wall Clock Limit	Maximum amount of elapsed time for which a job can run. Optional

SELECT Close to return to the Build a Job dialog box.

SELECT Checkpointing to specify checkpoint options (available when the checkpoint option is set to Yes or Interval)

▲ The checkpointing dialog box appears.

Complete those fields for which you want to specify checkpoint information (see Table 61). For detailed information on specific keywords, see “Job command file keyword descriptions” on page 336.

Table 61. Checkpointing dialog box fields

Field	Input
Ckpt File	Specifies a checkpoint file. The serial default is : \$(job_name).\$(host).\$(domain).\$(jobid).\$(stepid).ckpt
Ckpt Directory	Specifies a checkpoint directory name.
Ckpt Execute Directory	Specifies a directory to use for staging the checkpoint executable file.
Ckpt Time Limits	Sets the limits for the elapsed time a job can take checkpointing.

SELECT

Close to return to the Build a Job dialog box.

SELECT

Blue Gene (available when the job type is bluegene)

▲ The Blue Gene window appears.

Complete the necessary fields to specify information for a Blue Gene job (see Table 62). Depending upon which request type you choose, different fields will be available; any unavailable fields will be desensitized. For more information, see the appropriate job command file keyword (listed in parentheses) in “Job command file keyword descriptions” on page 336.

Table 62. Blue Gene job fields

Field	Available when requesting by:	Input
# of Compute Nodes	Size	The requested size in number of compute nodes that describes the size of the partition for this Blue Gene job. (bg_size)
Shape	Shape	The requested shape of the requested Blue Gene job. The units of each dimension of the shape are in number of base partitions, $X \times Y \times Z$, where X, Y, and Z are the number of base partitions in the X-direction, Y-direction, and Z-direction. (bg_shape)
Partition Name	Partition	The name of an existing partition in the Blue Gene system where the requested job should run. (bg_partition)
Connection Type	Size and Shape	The kinds of Blue Gene partitions that can be selected for this job. You can select Torus, Mesh, or Prefer Torus. (bg_connection) Optional. The default is Mesh.

Table 62. Blue Gene job fields (continued)

Field	Available when requesting by:	Input
Rotate Dimensions	Shape	Whether to consider all possible rotations of the specified shape (True) or only the specified shape (False) when assigning a partition for the Blue Gene job. (bg_rotate) Optional. The default is True.
Memory	Megabytes	A number (in megabytes) that represents the minimum available virtual memory that is needed to run the job. LoadLeveler generates a Blue Gene requirement that specifies memory that is greater than or equal to the amount you specify. Optional. If you leave this field blank, this parameter is not used when searching for machines to run your job.
Requirements	Expression	An expression that specifies the Blue Gene requirements that a machine must meet in order to run the job. Memory is the supported keyword.

SELECT Close to return to the Build a Job dialog box.

Editing the job command file

There are several ways that you can edit the job command file that you just built:

- Using the Jobs window:

SELECT **File → Submit a Job**

▲ The Submit a Job dialog box appears.

SELECT The job file you want to edit from the file column.

SELECT **Edit**

▲ Your job command file appears in a window. You can use any editor to edit the job command file. The default editor is specified in your .Xdefaults file.

If you have an icon manager, an icon may appear. An icon manager is a program that creates a graphic symbol, displayed on a screen, that you can point to with a device such as a mouse in order to select a particular function or application. Select this icon to view your job command file.

- Using the **Tools Edit** pull-down menus on the Build a Job window:

Using the Edit pull-down menu, you can modify the job command file. Your choices appear in the Table 63 on page 230:

Table 63. Modifying the job command file with the Edit pull-down menu

To	Select
Add a step to the job command file	Add a Step or Add a First Step
Delete a step from the job command file	Delete a Step
Clear the fields in the Build a Job window	Clear Fields
Select defaults to use in the fields	Set Field Defaults

Note: Other options include Go to Next Step, Go to Previous Step, and Go to Last Step that allow you to edit various steps in the job command file.

Using the **Tools** pull-down menu, you can modify the job command file. Your choices appear in Table 64:

Table 64. Modifying the job command file with the Tools pull-down menu

To	Select
Name the job	Set Job Name
Specify a cluster, cluster list, or any cluster, if a multicluster environment is configured.	Set Cluster
Open a window where you can enter a script file	Append Script
Fill in the fields using another file	Restore from File
View the job command file in a window	View Entire Job
Determine which step you are viewing	What is step #
Start a new job command file	Start a new job

You can save and submit the information you entered by selecting the choices shown in Table 65:

Table 65. Saving and submitting information

To	Do This
Save the information you entered into a file which you can submit later	<p>SELECT Save</p> <p>▲ A window appears prompting you to enter a job filename.</p> <p>ENTER a job filename in the text entry field.</p> <p>SELECT OK</p> <p>▲ The window closes and the information you entered is saved in the file you specified.</p>
Submit the program immediately and discard the information you entered	<p>SELECT Submit</p>

Submitting a job command file

After building a job command file, you can submit it to one or more machines for processing.

To submit a job, from the Jobs window:

SELECT **File → Submit a Job**

▲ The Submit a Job dialog box appears.

SELECT The job file that you want to submit from the file column.

You can also use the filter field and the directories column to select the file or you can type in the file name in the text entry field.

SELECT **Submit**

▲ The job is submitted for processing.

You can now submit another job or you can press Close to exit the window.

Displaying and refreshing job status

When you submit a job, the status of the job is automatically displayed in the Jobs window. You can update or refresh this status using the Jobs window and selecting one of the following:

- **Refresh → Refresh Jobs**
- **Refresh → Refresh All.**

To change how often the amount of time should pass before the jobs window is automatically refreshed, use the Jobs window.

SELECT **Refresh → Set Auto Refresh**

▲ A window appears.

TYPE IN a value for the number of seconds to pass before the Jobs window is updated.

Automatic refresh can be expensive in terms of network usage and CPU cycles. You should specify a refresh interval of 120 seconds or more for normal use.

SELECT **OK**

▲ The window closes and the value you specified takes effect.

To receive detailed information on a job:

SELECT **Actions → Extended Status** to receive additional information on the job. Selecting this option is the same as typing **llq -x** command.

You can also get information in the following way:

SELECT **Actions → Extended Details**

Selecting this option is the same as typing **llq -x -l** command. You can also double click on the job in the Jobs window to get details on the job.

Note: Obtaining extended status or details on multiple jobs can be expensive in terms of network usage and CPU cycles.

SELECT **Actions → Job Status**

You can also use the **llq -s** command to determine why a submitted job remains in the Idle or Deferred state.

SELECT **Actions → Resource Use**

Allows you to display resource use for running jobs. Selecting this option is the same as entering the **llq -w** command.

Using the GUI

SELECT **Actions → Blue Gene Job Status**

Allows you to display Blue Gene job information for jobs. Selecting this option is the same as entering the **llq -b** command.

For more information on requests for job information, see “llq - Query job status” on page 449.

Sorting the Jobs window

You can specify up to two sorting options for the Jobs window. The options you specify determine the order in which the jobs appear in the Jobs window.

From the Jobs window:

Select **Sort → Set Sort Parameters**

▲ A window appears

Select A primary and secondary sort

Table 66 lists the sorting options:

Table 66. Sorting the jobs window

To:	Select Sort
Sort jobs by the machine from which they were submitted	Sort by Submitting Machine
Sort by owner	Sort by Owner
Sort by the time the jobs were submitted	Sort by Submission Time
Sort by the state of the job	Sort by State
Sort jobs by their user priority (last job listed runs first)	Sort by Priority
Sort by the class of the job	Sort by Class
Sort by the group associated with the job	Sort by Group
Sort by the machine running the job	Sort by Running Machine
Sort by dispatch order	Sort by Dispatch Order
Not specify a sort	No Sort

You can select a sort type as either a Primary or Secondary sorting option. For example, suppose you select Sort by Owner as the primary sorting option and Sort by Class as the secondary sorting option. The Jobs window is sorted by owner and, within each owner, by class.

Changing the priority of your jobs

If your job has not yet begun to run and is still in the queue, you can change the priority of the job in relation to your other jobs in the queue that belong to the same class. This only affects the user priority of the job. For more information on this priority, refer to “Setting and changing the priority of a job” on page 212. Only the owner of a job or the LoadLeveler administrator can change the priority of a job.

From the Jobs window:

SELECT a job by clicking on it with the mouse

- SELECT** **Actions → Priority**
 ▲ A window appears.
- TYPE IN** a number between 0 and 100, inclusive, to indicate a new priority.
- SELECT** **OK**
 ▲ The window closes and the priority of your job changes.

Placing a job on hold

Only the owner of a job or the LoadLeveler administrator can place a hold on a job.

From the Jobs window:

- SELECT** The job you want to hold by clicking on it with the mouse
- SELECT** **Actions → Hold**
 ▲ The job is put on hold and its status changes in the Jobs window.

Releasing the hold on a job

Only the owner of a job or the LoadLeveler administrator can release a hold on a job.

From the Jobs window:

- SELECT** The job you want to release by clicking on it with the mouse
- SELECT** **Actions → Release from Hold**
 ▲ The job is released from hold and its status is updated in the Jobs window.

Canceling a job

Only the owner of a job or the LoadLeveler administrator can cancel a job.

From the Jobs window:

- SELECT** The job you want to cancel by clicking on it with the mouse
- SELECT** **Actions → Cancel**
 ▲ LoadLeveler cancels the job and the job information disappears from the Jobs window.

Modifying consumable resources and other job attributes

Modifies the consumable CPUs or memory requirements of a nonrunning job.

- SELECT**
- Modify → Consumable CPUs**
 or
 Modify → Consumable Memory
 or
 Modify → Class
 or
 Modify → Account number

Using the GUI

or

Modify → Blue Gene → Connection

or

Modify → Blue Gene → Partition

or

Modify → Blue Gene → Rotate

or

Modify → Blue Gene → Shape

or

Modify → Blue Gene → Size

or

Modify → Blue Gene → Requirement

▲ A dialog box appears prompting you to enter a new value for the selected job attribute. Blue Gene attributes are available when Blue Gene is enabled.

TYPE IN

The new value

SELECT

OK

▲ The dialog box closes and the value you specified takes effect.

Taking a checkpoint

Checkpoints the selected job.

SELECT

One of the following actions to take when checkpoint has completed:

- Continue the step
- Terminate the step
- Hold the step

▲ A checkpoint monitor for this step appears.

Adding a job to a reservation

Binds selected job steps to a reservation so that they will only be scheduled to run on the nodes reserved for the reservation.

SELECT

The job you want to bind by clicking on it with the mouse.

SELECT

Actions → Bind to Reservation

▲ A window appears.

SELECT

A reservation from the list.

SELECT

OK

▲ The window closes and the job is bound to that reservation.

Removing a job from a reservation

Unbinds selected job steps from reservations to which they currently belong.

SELECT

The job you want to unbind by clicking on it with the mouse.

SELECT

Actions → Unbind from Reservation

If the job is bound to a reservation, it is removed from the reservation.

Displaying and refreshing machine status

The status of the machines is automatically displayed in the Machines window. You can update or refresh this status using the Machines window and selecting one of the following:

- **Refresh → Refresh Machines**
- **Refresh → Refresh All.**

To specify an amount of time to pass before the Machines window is automatically refreshed, from the Machines window:

SELECT Refresh → Set Auto Refresh

▲ A window appears.

TYPE IN

a value for the number of seconds to pass before the Machines window is updated.

Automatic refresh can be expensive in terms of network usage and CPU cycles. You should specify a refresh interval of 120 seconds or more for normal use.

SELECT OK

▲ The window closes and the value you specified takes effect.

To receive detailed information on a machine:

SELECT

Actions → Details

This displays status information about the selected machines. Selecting this option has the same effect as typing the **llstatus -l** command

SELECT

Actions → Adapter Details

This displays virtual and physical adapter information for each selected machine. Selecting this option has the same effect as typing the **llstatus -a** command

SELECT

Actions → Floating Resources

This displays consumable resources for the LoadLeveler cluster. Selecting this option has the same effect as typing the **llstatus -R** command

SELECT

Actions → Machine Resources

This displays consumable resources defined for the selected machines or all machines. Selecting this option has the same effect as typing the **llstatus -R** command

SELECT

Actions → Cluster Status

This displays status of machines in the defined cluster or clusters. It appears only when a multicluster environment is configured and is equivalent to the `llstatus -X all` command.

SELECT

Actions → Cluster Config

This displays cluster information from the `LoadL_admin` file. Only fields with data specified or which have defaults when not specified are displayed. It appears only when a multicluster environment is configured and is equivalent to the `llstatus -C` command.

SELECT

Actions → Blue Gene ...

This displays information about the Blue Gene system. You can select the option for **Status** for a short listing, **Details** for a long listing, **Base Partitions** for Blue Gene base partition status, or **Partitions** for existing Blue Gene partition status. It is available only when Blue Gene support is enabled in LoadLeveler. This is equivalent to the `llstatus` command with the options `-b`, `-b -l`, `-B`, or `-P`.

Sorting the Machines window

You can specify up to two sorting options for the Machines window. The options you specify determine the order in which machines appear in the window.

From the Machines window:

Select **Sort** → **Set Sort Parameters**

▲ A window appears

Select A **primary and secondary sort**

Table 67 lists sorting options for the Machines window:

Table 67. Sorting the machines window

To:	Select Sort →
Sort by machine name	Sort by Name
Sort by Schedd state	Sort by Schedd
Sort by total number of jobs scheduled	Sort by InQ
Sort by number of running jobs scheduled by this machine	Sort by Act
Sort by startd state	Sort by Startd
Sort by the number of jobs running on this machine	Sort by Run
Sort by load average	Sort by LdAvg
Sort by keyboard idle time	Sort by Idle
Sort by hardware architecture	Sort by Arch
Sort by operating system type	Sort by OpSys
Not specify a sort	No Sort

You can select a sort type as either a Primary or Secondary sorting option. For example, suppose you select Sort by Arch as the primary sorting option and Sort by Name as the secondary sorting option. The Machines window is sorted by hardware architecture, and within each architecture type, by machine name.

Finding the location of the central manager

The LoadLeveler administrator designates one of the nodes in the LoadLeveler cluster as the central manager. When jobs are submitted at any node, the central manager is notified and decides where to schedule the jobs. In addition, it keeps track of the status of machines in the cluster and the jobs in the system by communicating with each node. LoadLeveler uses this information to make the scheduling decisions and to respond to queries.

To find the location of the central manager, from the Machines window:

SELECT **Actions → Find Central Manager**

▲ A message appears in the message window declaring on which machine the central manager is located.

Finding the location of the public scheduling machines

Public scheduling machines are those machines that participate in the scheduling of LoadLeveler jobs on behalf of the submit-only machines.

To get a list of these machines in your cluster, use the Machines window:

SELECT **Actions → Find Public Scheduler**

▲ A message appears displaying the names of these machines.

Finding the type of scheduler in use

The LoadLeveler administrator defines the scheduler used by the cluster. To determine which scheduler is currently in use:

SELECT **Actions → Find Scheduler Type**

- ▲ A message appears displaying the type:
- ll_default
 - BACKFILL
 - External (API)

Specifying which jobs appear in the Jobs window

Normally, only your jobs appear in the Jobs window. You can, however, specify which jobs you want to appear by using the Select pull-down menu on the Jobs window (see Table 68).

Table 68. Specifying which jobs appear in the Jobs window

To Display	Select Select →
All jobs in the queue	All
All jobs belonging to a specific user (or users)	By User ▲ A window appears prompting you to enter the user IDs whose jobs you want to view.

Table 68. Specifying which jobs appear in the Jobs window (continued)

To Display	Select Select →
All jobs submitted to a specific machine (or machines)	By Machine ▲ A window appears prompting you to enter the machine names on which the jobs you want to view are running.
All jobs belonging to a specific group (or groups)	By Group ▲ A window appears prompting you to enter the LoadLeveler group names to which the jobs you want to view belong.
All jobs having a particular ID	By Job Id A dialog box prompts you to enter the id of the job you want to appear. This ID appears in the left column of the Jobs window. Type in the ID and press OK.

Note: When you choose By User, By Machines, or By Group, you can use a UNIX regular expression enclosed in parenthesis. For example, you can enter (^k10) to display all machines beginning with the characters “k10”.

SELECT **Select → Show Selection** to show the selection parameters.

Specifying which machines appear in Machines window

You can specify which machines will appear in the Machines window (see Table 69). The default is to view all of the machines in the LoadLeveler pool.

From the Machines window:

Table 69. Specifying which machines appear in Machines window

To	Select Select →
View all of the machines	All
View machines by operating system	by OpSys ▲ A window appears prompting you to enter the operating system of those machines you want to view.
View machines by hardware architecture	by Arch ▲ A window appears prompting you to enter the hardware architecture of those machines you want to view.
View machines by state	by State ▲ A cascading pull-down menu appears prompting you to select the state of the machines that you want to view.

SELECT **Select → Show Selection** to show the selection parameters.

Saving LoadLeveler messages in a file

Normally, all the messages that LoadLeveler generates appear in the Messages window. If you would also like to have these messages written to a file, use the Messages window.

SELECT **Actions → Start logging to a file**

▲ A window appears prompting you to enter a filename in which to log the messages.

TYPE IN

The filename in the text entry field.

SELECT

OK

▲ The window closes.

Using the GUI

Part 4. TWS LoadLeveler interfaces reference

The topics in Table 70 provide the details you need to know to correctly use the IBM Tivoli Workload Scheduler (TWS) LoadLeveler interfaces.

Table 70. Learning about LoadLeveler interfaces

To learn about:	Read the following:
Correctly specifying keywords in the TWS LoadLeveler control files	<ul style="list-style-type: none">• Chapter 12, “Configuration file reference,” on page 243• Chapter 13, “Administration file reference,” on page 301• Chapter 14, “Job command file reference,” on page 333
Starting and customizing the TWS LoadLeveler GUI	Chapter 15, “Graphical user interface (GUI) reference,” on page 377
Correctly coding the TWS LoadLeveler commands and APIs	<ul style="list-style-type: none">• Chapter 16, “Commands,” on page 385• Chapter 17, “Application programming interfaces (APIs),” on page 503

Chapter 12. Configuration file reference

The configuration file contains many parameters that you can set or modify to control how LoadLeveler operates. You may control LoadLeveler's operation either:

- Across the cluster, by modifying the global configuration file, **LoadL_config**, or
- Locally, by modifying the **LoadL_config.local** file on individual machines.

Table 71 shows the the configuration subtasks:

Table 71. Configuration subtasks

Subtask	Associated information (see . . .)
To find out what administrator tasks you can accomplish by using the configuration file	Part 2, "Configuring and managing the TWS LoadLeveler environment," on page 37
To learn how to correctly specify the contents of a configuration file	<ul style="list-style-type: none">• "Configuration file syntax"• "Configuration file keyword descriptions" on page 244• "User-defined keywords" on page 293• "LoadLeveler variables" on page 294

Configuration file syntax

The information in both the **LoadL_config** and the **LoadL_config.local** files is in the form of a statement. These statements are made up of *keywords* and *values*.

There are three types of configuration file keywords:

- Keywords, described in "Configuration file keyword descriptions" on page 244.
- User-defined variables, described in "User-defined keywords" on page 293.
- LoadLeveler variables, described in "LoadLeveler variables" on page 294.

Configuration file statements take one of the following formats:

keyword=value
keyword:value

Statements in the form *keyword=value* are used primarily to customize an environment. Statements in the form *keyword:value* are used by LoadLeveler to characterize the machine and are known as part of the machine description. Every machine in LoadLeveler has its own machine description which is read by the central manager when LoadLeveler is started.

Keywords are *not* case sensitive. This means you can enter them in lower case, upper case, or mixed case.

Note: For the *keyword=value* form, if the keyword is of a boolean type and only **true** and **false** are valid input, a value string starting with **t** or **T** is taken as **true**; all other values are taken as **false**.

To continue configuration file statements, use the back-slash character (\).

In the configuration file, comments must be on a separate line from keyword statements.

Configuration file reference

You can use the following types of constants and operators in the configuration file.

Numerical and alphabetical constants

Constants may be represented as:

- Boolean expressions
- Signed integers
- Floating point values
- Strings enclosed in double quotes (" ").

Mathematical operators

You can use the following C operators. The operators are listed in order of precedence. All of these operators are evaluated from left to right:

- !
- * /
- - +
- < <= > >=
- == !=
- &&
- ||

64-bit support for configuration file keywords and expressions

Administrators can assign 64-bit integer values to selected keywords in the configuration file.

floating_resources

Consumable resources associated with the **floating_resources** keyword may be assigned 64-bit integer values. Fractional and unit specifications are not allowed. The predefined `ConsumableCpus`, `ConsumableMemory`, and `ConsumableVirtualMemory` may not be specified as floating resources.

Example:

```
floating_resources = spice2g6(9876543210123) db2_license(1234567890)
```

MACHPRIO expression

The `LoadLeveler` variables `Memory`, `VirtualMemory`, `FreeRealMemory`, `Disk`, `ConsumableMemory`, `ConsumableVirtualMemory`, `ConsumableCpus`, `PagesScanned`, `PagesFreed` may be used in a `MACHPRIO` expression. They are 64-bit integers and 64-bit arithmetic is used to evaluate them.

Example:

```
MACHPRIO: (Memory + FreeRealMemory) - (LoadAvg*1000 + PagesScanned)
```

Configuration file keyword descriptions

This topic provides an alphabetical list of the keywords you can use in a `LoadLeveler` configuration file. It also provides examples of statements that use these keywords.

ACCT Turns the accounting function on or off.

Syntax:

```
ACCT = flag ...
```

The available flags are:

A_DETAIL Enables extended accounting. Using this flag causes

LoadLeveler to record detail resource consumption by machine and by events for each job step. This flag also enables the `-x` flag of the `llq` command, permitting users to view resource consumption for active jobs.

A_RES	Turns reservation data recording on.
A_OFF	Turns accounting data recording off.
A_ON	Turns accounting data recording on. If specified without the A_DETAIL flag, the following is recorded: <ul style="list-style-type: none"> • The total amount of CPU time consumed by the entire job • The maximum memory consumption of all tasks (or nodes).
A_VALIDATE	Turns account validation on.

Default value: A_OFF

Example: This example specifies that accounting should be turned on and that extended accounting data should be collected and that the `-x` flag of the `llq` command be enabled.

```
ACCT = A_ON A_DETAIL
```

ACCT_VALIDATION

Identifies the executable called to perform account validation.

Syntax:

```
ACCT_VALIDATION = program
```

Where *program* is a validation program.

Default value: `$(BIN)/llacctval` (the accounting validation program shipped with LoadLeveler.

ACTION_ON_MAX_REJECT

Specifies the state in which jobs are placed when their rejection count has reached the value of the **MAX_JOB_REJECT** keyword. **HOLD** specifies that jobs are placed in User Hold status; **SYSHOLD** specifies that jobs are placed in System Hold status; **CANCEL** specifies that jobs are canceled. When a job is rejected, LoadLeveler sends a mail message stating why the job was rejected.

Syntax:

```
ACTION_ON_MAX_REJECT = HOLD | SYSHOLD | CANCEL
```

Default value: **HOLD**

ACTION_ON_SWITCH_TABLE_ERROR

Points to an administrator supplied program that will be run when **DRAIN_ON_SWITCH_TABLE_ERROR** is set to **true** and a switch table unload error occurs.

Syntax:

```
ACTION_ON_SWITCH_TABLE_ERROR = program
```

Default value: The default is to not run a program.

Configuration file reference

ADMIN_FILE

Points to the administration file containing user, class, group, machine, and adapter stanzas.

Syntax:

ADMIN_FILE = *directory*

Default value: \$(tilde)/admin_file

AFS_GETNEWTOKEN

Specifies a filter that, for example, can be used to refresh an AFS token.

Syntax:

AFS_GETNEWTOKEN = *full_path_to_executable*

Where *full_path_to_executable* is an administrator-supplied program that receives the AFS authentication information on standard input and writes the new information to standard output. The filter is run when the job is scheduled to run and can be used to refresh a token which expired when the job was queued.

Default value: The default is to not run a program.

AGGREGATE_ADAPTERS

Allows an external scheduler to specify per-window adapter usages.

Syntax:

AGGREGATE_ADAPTERS = YES | NO

When this keyword is set to **YES**, the resources from multiple switch adapters on the same switch network are treated as one aggregate pool available to each job. When this keyword is set to **NO**, the switch adapters are treated individually and a job cannot use resources from multiple adapters on the same network.

Set this keyword to **NO** when you are using an external scheduler; otherwise, set to **YES** (or accept the default).

Default value: YES

ALLOC_EXCLUSIVE_CPU_PER_JOB

Specifies the way CPU affinity is enforced on Linux platforms. When this keyword is not specified or when an unrecognized value is assigned to it, LoadLeveler will not attempt to set CPU affinity for any application processes spawned by it.

Note: This keyword is ignored by LoadLeveler on platforms on which the LoadLeveler for Linux CPU affinity feature is not available.

The **ALLOC_EXCLUSIVE_CPU_PER_JOB** keyword can be specified in the global or local configuration files. It can also be specified in both configuration files, in which case the setting in the local configuration file will override that of the global configuration file. The keyword cannot be turned off in a local configuration file if it has been set to any value in the global configuration file.

Changes to **ALLOC_EXCLUSIVE_CPU_PER_JOB** will not take effect at reconfiguration. The administrator must stop and restart or recycle LoadLeveler when changing **ALLOC_EXCLUSIVE_CPU_PER_JOB**.

Syntax:

ALLOC_EXCLUSIVE_CPU_PER_JOB = LOGICAL|PHYSICAL

Default value: By default, when this keyword is not specified, CPU affinity is not set.

Example: When the value of this keyword is set to **LOGICAL**, only one LoadLeveler job step will run on each of the processors available on the machine:

ALLOC_EXCLUSIVE_CPU_PER_JOB = LOGICAL

Example: When the value of this keyword is set to **PHYSICAL**, all logical processors (or physical cores) configured in one physical CPU package will be allocated to one and only one LoadLeveler job step.

ALLOC_EXCLUSIVE_CPU_PER_JOB = PHYSICAL

For more information related to this keyword, see “Linux CPU affinity support” on page 137.

ARCH

Indicates the standard architecture of the system. The architecture you specify here must be specified in the same format in the **requirements** and **preferences** statements in job command files. The administrator defines the character string for each architecture.

Syntax:

ARCH = *string*

Default value: Use the command `llstatus -l` to view the default.

Example: To define a machine as an RS/6000®, the keyword would look like:

ARCH = R6000

BG_ALLOW_LL_JOBS_ONLY

Specifies if only jobs submitted through LoadLeveler will be accepted by the Blue Gene job launcher program.

Syntax:

BG_ALLOW_LL_JOBS_ONLY = true | false

Default value: false

BG_CACHE_PARTITIONS

Specifies whether allocated partitions are to be reused for Blue Gene jobs whenever possible.

Syntax:

BG_CACHE_PARTITIONS = true | false

Default value: true

BG_ENABLED

Specifies whether Blue Gene support is enabled.

Syntax:

BG_ENABLED = true | false

Configuration file reference

If the value of this keyword is **true**, the Central Manager will load the Blue Gene control system libraries and query the state of the Blue Gene system so that jobs of type **bluegene** can be scheduled.

Default value: false

BG_MIN_PARTITION_SIZE

Specifies the smallest number of compute nodes in a partition.

Syntax:

BG_MIN_PARTITION_SIZE = 32 | 128 | 512

The value for this keyword must not be smaller than the minimum partition size supported by the physical Blue Gene hardware. If the number of compute nodes requested in a job is less than the minimum partition size, then LoadLeveler will increase the requested size to the minimum partition size.

Default value: 32

BIN Defines the directory where LoadLeveler binaries are kept.

Syntax:

BIN = **\$(RELEASEDIR)/bin**

Default value: \$(tilde)/bin

CENTRAL_MANAGER_HEARTBEAT_INTERVAL

Specifies the amount of time, in seconds, that defines how frequently primary and alternate central manager communicate with each other.

Syntax:

CENTRAL_MANAGER_HEARTBEAT_INTERVAL = *number*

Default value: The default is 300 seconds or 5 minutes.

CENTRAL_MANAGER_TIMEOUT

Specifies the number of heartbeat intervals that an alternate central manager will wait before declaring that the primary central manager is not operating.

Syntax:

CENTRAL_MANAGER_TIMEOUT = *number*

Default value: The default is 6.

CKPT_CLEANUP_INTERVAL

Specifies the interval, in seconds, at which the **Schedd** daemon will run the program specified by the **CKPT_CLEANUP_PROGRAM** keyword.

Syntax:

CKPT_CLEANUP_INTERVAL = *number*

number must be a positive integer.

Default value: -1

CKPT_CLEANUP_PROGRAM

Identifies an administrator-provided program which is to be run at the interval specified by the **ckpt_cleanup_interval** keyword. The intent of this

program is to delete old checkpoint files created by jobs running under LoadLeveler during the checkpoint process.

Syntax:

`CKPT_CLEANUP_PROGRAM = program`

Where *program* is the fully qualified name of the program to be run. The program must be accessible and executable by LoadLeveler.

A sample program to remove checkpoint files is provided in the `/usr/lpp/LoadL/full/samples/l1ckpt/rmckptfiles.c` file.

Default value: No default value is set.

CKPT_EXECUTE_DIR

Specifies the directory where the job step's executable will be saved for checkpointable jobs. You may specify this keyword in either the configuration file or the job command file; different file permissions are required depending on where this keyword is set. For additional information, see "Planning considerations for checkpointing jobs" on page 130.

Syntax:

`CKPT_EXECUTE_DIR = directory`

This directory cannot be the same as the current location of the executable file, or LoadLeveler will not stage the executable. In this case, the user must have execute permission for the current executable file.

Default value: By default, the executable of a checkpointable job step is not staged.

CLASS

Determines whether a machine will accept jobs of a certain job class. For parallel jobs, you must define a class instance for each task you want to run on a node using one of two formats:

- The format, **CLASS = class_name (count)**, defines the **CLASS** names using a statement that names the classes and sets the number of tasks for each class in parenthesis.

With this format, the following rules apply:

- Each class can have only one entry
- If a class has more than one entry or there is a syntax error, the entire **CLASS** statement will be ignored
- If the **CLASS** statement has a blank value or is not specified, it will be defaulted to **No_Class (1)**
- The number of instances for a class specified inside the parenthesis () must be an unsigned integer. If the number specified is 0, it is correct syntactically, but the class will not be defined in LoadLeveler
- If the number of instances for all classes in the **CLASS** statement are 0, the default **No_Class(1)** will be used
- The format, **CLASS = { "class1" "class2" "class2" "class2" }**, defines the **CLASS** names using a statement that names each class and sets the number of tasks for each class based on the number of times that the class name is used inside the {} operands.

Note: With both formats, the class names list is blank delimited.

Configuration file reference

For a LoadLeveler job to run on a machine, the machine must have a vacancy for the class of that job. If the machine is configured for only one **No_Class** job and a LoadLeveler job is already running there, then no further LoadLeveler jobs are started on that machine until the current job completes.

You can have a maximum of 1024 characters in the class statement. You cannot use **allclasses** as a class name, since this is a reserved LoadLeveler keyword.

You can assign multiple classes to the same machine by specifying the classes in the LoadLeveler configuration file (called **LoadL_config**) or in the local configuration file (called **LoadL_config.local**). The classes, themselves, should be defined in the administration file. See "Setting up a single machine to have multiple job classes" on page 669 and "Defining classes" on page 83 for more information on classes.

Syntax:

```
CLASS = { "class_name" ... } | {"No_Class"} | class_name (count) ...
```

Default value: {"No_Class"}

CLIENT_TIMEOUT

Specifies the maximum time, in seconds, that a daemon waits for a response over TCP/IP from a process. If the waiting time exceeds the specified amount, the daemon tries again to communicate with the process. In general, you should use the default setting unless you are experiencing delays due to an excessively loaded network. If so, you should try increasing this value.

Syntax:

```
CLIENT_TIMEOUT = number
```

Default value: The default is 30 seconds.

CLUSTER_METRIC

Indicates the installation exit to be run by the Schedd to determine where a remote job is distributed. If a remote job is submitted with a list of clusters or the reserved word **any** and the installation exit is not specified, the remote job is not submitted.

Syntax:

```
CLUSTER_METRIC = full_pathname_to_executable
```

The installation exit is run with the following parameters passed as input. All parameters are character strings.

- The job ID of the job to be distributed
- The number of clusters in the list of clusters
- A blank-delimited list of clusters to be considered

If the user specifies the reserved word **any** as the **cluster_list** during job submission, the job is sent to the first outbound Schedd defined for the first configured remote cluster. The **CLUSTER_METRIC** is executed on this machine to determine where the job will be distributed. If this machine is not the **outbound_hosts** Schedd for the assigned cluster, the job will be forwarded to the correct **outbound_hosts** Schedd. If the user specifies a list of clusters as the **cluster_list** during job submission, the job is sent to the first outbound Schedd defined for the first specified remote cluster. The **CLUSTER_METRIC** is executed on this machine to determine where the

job will be distributed. If this machine is not the **outbound_hosts** Schedd for the assigned cluster, the job will be forwarded to the correct **outbound_hosts** Schedd.

Note: The list of clusters may contain a single entry of the reserved word **any**, which indicates that the **CLUSTER_METRIC** installation exit must determine its own list of clusters to select from. This can be all of the clusters available using the data access API or a predetermined list set by the administrator. If **any** is specified in place of a cluster list, the metric will receive a count of 1 followed by the keyword **any**.

The installation exit must write the remote cluster name to which the job is submitted as standard output and exit with a value of 0. An exit value of -1 indicates an error in determining the cluster for distribution and the job is not submitted. Returned cluster names that are not valid also cause the job to be not submitted. STDERR from the exit is written to the Schedd log.

LoadLeveler provides a set of sample exits for use in distributing jobs by the following metrics:

- The number of jobs in the idle queue
- The number of jobs in the specified class
- The number of free nodes in the cluster

The installation exit samples are available in the **`\${RELEASEDIR}/samples/llcluster** directory.

CLUSTER_REMOTE_JOB_FILTER

Indicates the installation exit to be run by the inbound Schedd for each remote job request to filter the user's job command file statements during submission or move job. If the keyword is not specified, no job filtering is done.

Syntax:

`CLUSTER_REMOTE_JOB_FILTER = full_pathname_to_executable`

The installation exit is run with the submitting user's ID. All parameters are character strings.

This installation exit is executed on the **inbound_hosts** of the local cluster when receiving a job submission or move job request.

The executable specified is called with the submitting user's unfiltered job command file statements as the standard input. The standard output is submitted to LoadLeveler. If the exit returns with a nonzero exit code, the remote job submission or job move will fail. A submit filter can only make changes to LoadLeveler job command file statements.

The data access API can be used by the remote job filter to query the Schedd for the job object received from the sending cluster.

If the local submission filter on the submitting cluster has added or deleted steps from the original user's job command file, the remote job filter must add or delete the same number of steps. The job command file statements returned by the remote job filter must contain the same number of steps as the job object received from the sending cluster.

Configuration file reference

Changes to the following job command file keyword statements are ignored:

- **executable**
- **environment**
- **image_size**
- **cluster_input_file**
- **cluster_output_file**
- **cluster_list**

The following job command file keyword will have different behavior:

- **initialdir** – If not set by the remote job filter or the submitting user's unfiltered job command file, the default value will remain the current working directory at the time the job was submitted. Access to the **initialdir** will be verified on the cluster selected to run the job. If access to **initialdir** fails, the submission or move job will fail.

To maintain compatibility between the **SUBMIT_FILTER** and **CLUSTER_REMOTE_JOB_FILTER** programs, the following environment variables are set when either exit is invoked:

- **LOADL_ACTIVE** – the LoadLeveler version.
- **LOADL_STEP_COMMAND** – the location of the job command file passed as input to the program. This job command file only contains LoadLeveler keywords.
- **LOADL_STEP_ID** – The job identifier, generated by the submitting LoadLeveler cluster.

Note: The environment variable name is **LOADL_STEP_ID** although the value it contains is a "job" identifier. This name is used to be compatible with the local job filter interface.

- **LOADL_STEP_OWNER** – The owner (UNIX user name) of the job.

CLUSTER_USER_MAPPER

Indicates the installation exit to be run by the inbound Schedd for each remote job request to determine the user mapping of the cluster. This keyword implies that user mapping is performed. If the keyword is not specified, no user mapping is done.

Syntax:

`CLUSTER_USER_MAPPER = full_pathname_to_executable`

The installation exit is run with the following parameters passed as input. All parameters are character strings.

- The user name to be mapped
- The cluster name where the user originated from

This installation exit is executed on the **inbound_hosts** of the local cluster when receiving a job submission, move job request or remote command.

The installation exit must write the new user name as standard output and exit with a value of 0. An exit value of -1 indicates an error and the job is not submitted. STDERR from the exit is written to the Schedd log. An exit value of 1 indicates that the user name returned for this job was **not** mapped.

COLLECTOR_DGRAM_PORT

Specifies the port number used when connecting to a daemon.

Syntax:

`CM_COLLECTOR_PORT = port number`

Default value: The default is 9612.

COMM

Specifies a local directory where LoadLeveler keeps special files used for UNIX domain sockets for communicating among LoadLeveler daemons running on the same machine. This keyword allows the administrator to choose a different file system other than /tmp for these files. If you change the COMM option you must stop and then restart LoadLeveler using the **llctl** command.

Syntax:

`COMM = local directory`

Default value: The default location for the files is /tmp.

CONTINUE

Determines whether suspended jobs should continue execution.

Syntax:

`CONTINUE: expression that evaluates to T or F (true or false)`

When T, suspended LoadLeveler jobs resume execution on the machine.

Default value: No default value is set.

For information about time-related variables that you may use for this keyword, see “Variables to use for setting times” on page 299.

CUSTOM_METRIC

Specifies a machine’s relative priority to run jobs.

Syntax:

`CUSTOM_METRIC = number`

This is an arbitrary number which you can use in the MACHPRIO expression. Negative values are not allowed.

Default value: If you specify neither **CUSTOM_METRIC** nor **CUSTOM_METRIC_COMMAND**, **CUSTOM_METRIC = 1** is assumed. For more information, see “Setting negotiator characteristics and policies” on page 43.

For more information related to using this keyword, see “Defining a LoadLeveler cluster” on page 41.

CUSTOM_METRIC_COMMAND

Specifies an executable and any required arguments. The exit code of this command is assigned to **CUSTOM_METRIC**. If this command does not exit normally, **CUSTOM_METRIC** is assigned a value of 1. This command is forked every (**POLLING_FREQUENCY** * **POLLS_PER_UPDATE**) period.

Syntax:

`CUSTOM_METRIC_COMMAND = command`

Configuration file reference

Default value: No default is set; LoadLeveler does not run any command to determine CUSTOM_METRIC.

DCE_AUTHENTICATION_PAIR

Specifies a pair of installation supplied programs that are used to authenticate DCE security credentials.

Restriction: DCE security is not supported by LoadLeveler for Linux.

Syntax:

DCE_AUTHENTICATION_PAIR = *program1*, *program2*

Where *program1* and *program2* are LoadLeveler- or installation-supplied programs that are used to authenticate DCE security credentials. *program1* obtains a handle (an opaque credentials object), at the time the job is submitted, which is used to authenticate to DCE. *program2* uses the handle obtained by *program1* to authenticate to DCE before starting the job on the executing machines.

Default value: See “Handling DCE security credentials” on page 68 for information about defaults.

DEFAULT_PREEMPT_METHOD

Specifies the default preemption method for LoadLeveler to use when a preempt method is not specified in a PREEMPT_CLASS statement or in the **llpreempt** command. LoadLeveler also uses this default preemption method to preempt job steps that are running on reserved machines when a reservation period begins.

Restrictions:

- This keyword is valid only for the BACKFILL scheduler.
- LoadLeveler for Linux does not support the suspend method of preemption, which is the default method. If you want to preempt jobs running on LoadLeveler for Linux, you must use this keyword to specify a method other than suspend.

Syntax:

DEFAULT_PREEMPT_METHOD = rm | sh | su | vc | uh

Valid values are:

- | | |
|-----------|--|
| rm | LoadLeveler preempts the jobs and removes them from the job queue. To rerun the job, the user must resubmit the job to LoadLeveler. |
| sh | LoadLeveler ends the jobs and puts them into System Hold state. They remain in that state on the job queue until an administrator releases them. After being released, the jobs go into Idle state and will be rescheduled to run as soon as resources for the job are available. |
| su | LoadLeveler suspends the jobs and puts them in Preempted state. They remain in that state on the job queue until the preempting job has terminated, and resources are available to resume the preempted job on the same set of nodes. To use this value, process tracking must be enabled. |
| vc | LoadLeveler ends the jobs and puts them in Vacate state. They remain in that state on the job queue and will be rescheduled to run as soon as resources for the job are available. |
| uh | LoadLeveler ends the jobs and puts them into User Hold state. They remain in that state on the job queue until an administrator |

releases them. After being released, the jobs go into Idle state and will be rescheduled to run as soon as resources for the job are available.

Default value: su (suspend method)

For more information related to using this keyword, see “Steps for configuring a scheduler to preempt jobs” on page 120.

DRAIN_ON_SWITCH_TABLE_ERROR

Specifies whether the **startd** should be drained when the switch table fails to unload. This will flag the administrator that intervention may be required to unload the switch table. When **DRAIN_ON_SWITCH_TABLE_ERROR** is set to true, the **startd** will be drained when the switch table fails to unload.

Syntax:

DRAIN_ON_SWITCH_TABLE_ERROR = true | false

Default value: false

ENFORCE_RESOURCE_MEMORY

Specifies whether the AIX Workload Manager is configured to limit, as precisely as possible, the real memory usage of a WLM class. For this keyword to be valid, ConsumableMemory must be set through the **ENFORCE_RESOURCE_USAGE** keyword.

Syntax:

ENFORCE_RESOURCE_MEMORY = true | false

Default value: false

ENFORCE_RESOURCE_POLICY

Specifies what type of resource entitlements will be assigned to the AIX Workload Manager classes. If the value specified is **shares**, it means a share value is assigned to the class based on the job step’s requested resources (one unit of resource equals one share). This is the default policy. If the value specified is **soft**, it means a percentage value is assigned to the class based on the job step’s requested resources and the total machine resources. This percentage can be exceeded if there is no contention for the resource. If the value specified is **hard**, it means a percentage value is assigned to the class based on the job step’s requested resources and the total machine resources. This percentage cannot be exceeded regardless of the contention for the resource. If desired, this keyword can be used in the **LoadL_config.local** file to set up a different policy for each machine. The **ENFORCE_RESOURCE_USAGE** keyword must be set for this keyword to be valid.

Syntax:

ENFORCE_RESOURCE_POLICY = hard |soft | shares

Default value: shares

ENFORCE_RESOURCE_SUBMISSION = true | false

Indicates whether jobs submitted should be checked for the **resources** keyword. If the value specified is **true**, LoadLeveler will check all jobs at submission time for the **resources** keyword. The job command file

Configuration file reference

resources keyword needs to have at least the resources specified as the **ENFORCE_RESOURCE_USAGE** keyword for the job to be submitted successfully.

If the value specified is **false**, no checking will be done and jobs submitted without the **resources** keyword will not have resources enforced. In this instance, those jobs may interfere with other jobs whose resources are enforced.

Syntax:

```
ENFORCE_RESOURCE_SUBMISSION = true | false
```

Default value: false

ENFORCE_RESOURCE_USAGE

Specifies that the AIX Workload Manager should be used to enforce CPU or real memory resources. This keyword accepts the predefined resources **ConsumableCpus** and **ConsumableMemory**. Either memory or CPUs or both can be enforced but the resources must also be specified on the **SCHEDULE_BY_RESOURCES** keyword. If **deactivate** is specified, LoadLeveler will deactivate AIX Workload Manager on all the nodes in the LoadLeveler cluster.

Restriction: WLM enforcement is ignored by LoadLeveler for Linux.

Syntax:

```
ENFORCE_RESOURCE_USAGE = ConsumableCpus ConsumableMemory | deactivate
```

EXECUTE

Specifies the local directory to store the executables of jobs submitted by other machines.

Syntax:

```
EXECUTE = local directory/execute
```

Default value: \$(tilde)/execute

FAIR_SHARE_INTERVAL

Specifies, in units of hours, the time interval it takes for resource usage in fair share scheduling to decay to 5% of its initial value. Historic fair share data collected before the most recent time interval of this length will have little impact on fair share scheduling.

Syntax:

```
FAIR_SHARE_INTERVAL = hours
```

Default value: The default value is 168 hours (one week). If a negative value or 0 is specified, the default value is used.

FAIR_SHARE_TOTAL_SHARES

Specifies the total number of shares that the cluster CPU or Blue Gene resources are divided into. If this value is less than or equal to 0, fair share scheduling is turned off.

Syntax:

```
FAIR_SHARE_TOTAL_SHARES = shares
```

Default value: The default value is 0.

FEATURE

Specifies an optional characteristic to use to match jobs with machines. You

can specify unique characteristics for any machine using this keyword. When evaluating job submissions, LoadLeveler compares any required features specified in the job command file to those specified using this keyword. You can have a maximum of 1024 characters in the feature statement.

Syntax:

```
Feature = {"string" ...}
```

Default value: No default value is set.

Example: If a machine has licenses for installed products ABC and XYZ, in the local configuration file you can enter the following:

```
Feature = {"abc" "xyz"}
```

When submitting a job that requires both of these products, you should enter the following in your job command file:

```
requirements = (Feature == "abc") && (Feature == "xyz")
```

Note: You must define a feature on all machines that will be able to run dynamic simultaneous multithreading (SMT). SMT is only supported on IBM System p5 machines.

Example: When submitting a job that requires the SMT function, first specify **smt = yes** in job command file (or select a class which had **smt = yes** defined). Next, specify **node_usage = not_shared** and last, enter the following in the job command file:

```
requirements = (Feature == "smt")
```

FLOATING_RESOURCES

Specifies which consumable resources are available collectively on all of the machines in the LoadLeveler cluster. The count for each resource must be an integer greater than or equal to zero, and each resource can only be specified once in the list. Any resource specified for this keyword that is not already listed in the **SCHEDULE_BY_RESOURCES** keyword will not affect job scheduling. If any resource is specified incorrectly with the **FLOATING_RESOURCES** keyword, then all floating resources will be ignored. **ConsumableCpus**, **ConsumableMemory**, and **ConsumableVirtualMemory** may not be specified as floating resources.

Syntax:

```
FLOATING_RESOURCES = name(count) name(count) ... name(count)
```

Default value: No default value is set.

FS_INTERVAL

Defines the number of minutes used as the interval for checking free file system space or inodes. If your file system receives many log messages or copies large executables to the LoadLeveler spool, the file system will fill up quicker and you should perform file size checking more frequently by setting the interval to a smaller value. LoadLeveler will not check the file system if the value of **FS_INTERVAL** is:

- Set to zero
- Set to a negative integer

Syntax:

```
FS_INTERVAL = minutes
```

Configuration file reference

Default value: If FS_INTERVAL is not specified but any of the other file-system keywords (FS_NOTIFY, FS_SUSPEND, FS_TERMINATE, INODE_NOTIFY, INODE_SUSPEND, INODE_TERMINATE) are specified, the FS_INTERVAL value will default to 5 and the file system will be checked. If no file-system or inode keywords are set, LoadLeveler does not monitor file systems at all.

For more information related to using this keyword, see “Setting up file system monitoring” on page 50.

FS_NOTIFY

Defines the lower and upper amounts, in bytes, of free file-system space at which LoadLeveler is to notify the administrator:

- If the amount of free space becomes less than the lower threshold value, LoadLeveler sends a mail message to the administrator indicating that logging problems may occur.
- When the amount of free space becomes greater than the upper threshold value, LoadLeveler sends a mail message to the administrator indicating that problem has been resolved.

Syntax:

FS_NOTIFY = *lower threshold, upper threshold*

Specify space in bytes with the unit B. A metric prefix such as K, M or G may precede the B. The valid range for both the lower and upper thresholds are -1B and all positive integers. If the value is set to -1, the transition across the threshold is not checked.

Default value: In bytes: 1KB, -1B

For more information related to using this keyword, see “Setting up file system monitoring” on page 50.

FS_SUSPEND

Defines the lower and upper amounts, in bytes, of free file system space at which LoadLeveler drains and resumes the Schedd and startd daemons running on a node.

- If the amount of free space becomes less than the lower threshold value, then LoadLeveler drains the Schedd and the startd daemons if they are running on a node. When this happens, logging is turned off and mail notification is sent to the administrator.
- When the amount of free space becomes greater than the upper threshold value, LoadLeveler signals the Schedd and the startd daemons to resume. When this happens, logging is turned on and mail notification is sent to the administrator.

Syntax:

FS_SUSPEND = *lower threshold, upper threshold*

Specify space in bytes with the unit B. A metric prefix such as K, M or G may precede the B. The valid range for both the lower and upper thresholds are -1B and all positive integers. If the value is set to -1, the transition across the threshold is not checked.

Default value: In bytes: -1B, -1B

For more information related to using this keyword, see “Setting up file system monitoring” on page 50.

FS_TERMINATE

Defines the lower and upper amounts, in bytes, of free file system space at which LoadLeveler is terminated. This keyword sends the SIGTERM signal to the Master daemon which then terminates all LoadLeveler daemons running on the node.

- If the amount of free space becomes less than the lower threshold value, all LoadLeveler daemons are terminated.
- An upper threshold value is required for this keyword. However, since LoadLeveler has been terminated at the lower threshold, no action occurs.

Syntax:

`FS_TERMINATE = lower threshold, upper threshold`

Specify space in bytes with the unit B. A metric prefix such as K, M or G may precede the B. The valid range for the lower threshold is -1B and all positive integers. If the value is set to -1, the transition across the threshold is not checked.

Default value: In bytes: -1B, -1B

For more information related to using this keyword, see “Setting up file system monitoring” on page 50.

GLOBAL_HISTORY

Identifies the directory that will contain the global history files produced by `llacmtrg` command when no directory is specified as a command argument.

Syntax:

`GLOBAL_HISTORY = directory`

Default value: The default value is `$(SPOOL)` (the local spool directory).

For more information related to using this keyword, see “Collecting the accounting information and storing it into files” on page 60.

GSMONITOR

Location of the `gsmonitor` executable (`LoadL_GSmonitor`).

Restriction: This keyword is ignored by LoadLeveler for Linux.

Syntax:

`GSMONITOR = directory`

Default value: `$(BIN)/LoadL_GSmonitor`

GSMONITOR_COREDUMP_DIR

Local directory for storing `LoadL_GSmonitor` core dump files.

Restriction: This keyword is ignored by LoadLeveler for Linux.

Syntax:

`GSMONITOR_COREDUMP_DIR = directory`

Default value: The `/tmp` directory.

For more information related to using this keyword, see “Specifying file and directory locations” on page 45.

Configuration file reference

GSMONITOR_DOMAIN

Specifies the peer domain, on which the GSMONITOR daemon will execute.

Restriction: This keyword is ignored by LoadLeveler for Linux.

Syntax:

GSMONITOR_DOMAIN = PEER

Default value: No default value is set.

For more information related to using this keyword, see “The gsmonitor daemon” on page 13.

GSMONITOR_RUNS_HERE

Specifies whether the gsmonitor daemon will run on the host.

Restriction: This keyword is ignored by LoadLeveler for Linux.

Syntax:

GSMONITOR_RUNS_HERE = TRUE | FALSE

Default value: FALSE

For more information related to using this keyword, see “The gsmonitor daemon” on page 13.

HISTORY

Defines the path name where a file containing the history of local LoadLeveler jobs is kept.

Syntax:

HISTORY = *directory*

Default value: \$(SPOOL)/history

For more information related to using this keyword, see “Collecting the accounting information and storing it into files” on page 60.

HISTORY_PERMISSION

Specifies the owner, group, and world permissions of the history file associated with a **LoadL_schedd** daemon.

Syntax:

HISTORY_PERMISSION = *permissions* | rw-rw----

permissions must be a string with a length of nine characters and consisting of the characters, **r**, **w**, **x**, or **-**.

Default value: The default settings are 660 (**rw-rw----**). **LoadL_schedd** will use the default setting if the specified permission are less than **rw-----**.

Example: A specification such as HISTORY_PERMISSION = rw-rw-r-- will result in permission settings of 664.

INODE_NOTIFY

Defines the lower and upper amounts, in inodes, of free file-system inodes at which LoadLeveler is to notify the administrator:

- If the number of free inodes becomes less than the lower threshold value, LoadLeveler sends a mail message to the administrator indicating that logging problems may occur.
- When the number of free inodes becomes greater than the upper threshold value, LoadLeveler sends a mail message to the administrator indicating that problem has been resolved.

Syntax:

`INODE_NOTIFY = lower threshold, upper threshold`

The valid range for both the lower and upper thresholds are -1 and all positive integers. If the value is set to -1, the transition across the threshold is not checked.

Default value: In inodes: 1000, -1

For more information related to using this keyword, see “Setting up file system monitoring” on page 50.

INODE_SUSPEND

Defines the lower and upper amounts, in inodes, of free file system inodes at which LoadLeveler drains and resumes the Schedd and startd daemons running on a node.

- If the number of free inodes becomes less than the lower threshold value, then LoadLeveler drains the Schedd and the startd daemons if they are running on a node. When this happens, logging is turned off and mail notification is sent to the administrator.
- When the number of free inodes becomes greater than the upper threshold value, LoadLeveler signals the Schedd and the startd daemons to resume. When this happens, logging is turned on and mail notification is sent to the administrator.

Syntax:

`INODE_SUSPEND = lower threshold, upper threshold`

The valid range for both the lower and upper thresholds are -1 and all positive integers. If the value is set to -1, the transition across the threshold is not checked.

Default value: In inodes: -1, -1

For more information related to using this keyword, see “Setting up file system monitoring” on page 50.

INODE_TERMINATE

Defines the lower and upper amounts, in inodes, of free file system inodes at which LoadLeveler is terminated. This keyword sends the SIGTERM signal to the Master daemon which then terminates all LoadLeveler daemons running on the node.

- If the number of free inodes becomes less than the lower threshold value, all LoadLeveler daemons are terminated.
- An upper threshold value is required for this keyword. However, since LoadLeveler has been terminated at the lower threshold, no action occurs.

Syntax:

`INODE_TERMINATE = lower threshold, upper threshold`

Configuration file reference

The valid range for the lower threshold is -1 and all positive integers. If the value is set to -1, the transition across the threshold is not checked.

Default value: In inodes: -1, -1

For more information related to using this keyword, see “Setting up file system monitoring” on page 50.

JOB_ACCT_Q_POLICY

Specifies the amount of time, in seconds, that determines how often the `startd` daemon updates the `Schedd` daemon with accounting data of running jobs. This controls the accuracy of the `llq -x` command.

Syntax:

`JOB_ACCT_Q_POLICY = number`

Default value: 300 seconds

For more information related to using this keyword, see “Gathering job accounting data” on page 57.

JOB_EPILOG

Pathname of the epilog program.

Syntax:

`JOB_EPILOG = program name`

Default value: No default value is set.

For more information related to using this keyword, see “Writing prolog and epilog programs” on page 70.

JOB_LIMIT_POLICY

Specifies the amount of time, in seconds, that `LoadLeveler` checks to see if `job_cpu_limit` has been exceeded. The smaller of `JOB_LIMIT_POLICY` and `JOB_ACCT_Q_POLICY` is used to control how often the `startd` daemon collects resource consumption data on running jobs, and how often the `job_cpu_limit` is checked.

Syntax:

`JOB_LIMIT_POLICY = number`

Default value: The default for `JOB_LIMIT_POLICY` is `POLLING_FREQUENCY` multiplied by `POLLS_PER_UPDATE`.

JOB_PROLOG

Pathname of the prolog program.

Syntax:

`JOB_PROLOG = program name`

Default value: No default value is set.

For more information related to using this keyword, see “Writing prolog and epilog programs” on page 70.

JOB_USER_EPILOG

Pathname of the user epilog program.

Syntax:

`JOB_USER_EPILOG = program name`

Default value: No default value is set.

For more information related to using this keyword, see “Writing prolog and epilog programs” on page 70.

JOB_USER_PROLOG

Pathname of the user prolog program.

Syntax:

`JOB_USER_PROLOG = program name`

Default value: No default value is set.

For more information related to using this keyword, see “Writing prolog and epilog programs” on page 70.

KBDD

Location of kbdd executable (**LoadL_kbdd**).

Syntax:

`KBDD = directory`

Default value: `$(BIN)/LoadL_kbdd`

KBDD_COREDUMP_DIR

Local directory for storing **LoadL_kbdd** daemon core dump files.

Syntax:

`KBDD_COREDUMP_DIR = directory`

Default value: The `/tmp` directory.

For more information related to using this keyword, see “Specifying file and directory locations” on page 45.

KILL Determines whether or not vacated jobs should be sent the SIGKILL signal and replaced in the queue. It is used to remove a job that is taking too long to vacate.

Syntax:

`KILL: expression that evaluates to T or F (true or false)`

When **T**, vacated LoadLeveler jobs are removed from the machine with no attempt to take checkpoints.

For information about time-related variables that you may use for this keyword, see “Variables to use for setting times” on page 299.

LIB Defines the directory where LoadLeveler libraries are kept.

Syntax:

`LIB = directory`

Default value: `$(RELEASEDIR)/lib`

LL_RSH_COMMAND

Specifies an administrator provided executable to be used by **llctl start** when starting LoadLeveler on remote machines in the administration file. The **LL_RSH_COMMAND** keyword is any executable that can be used as

Configuration file reference

a substitute for `/usr/bin/rsh`. The `llctl start` command passes arguments to the executable specified by `LL_RSH_COMMAND` in the following format:
`LL_RSH_COMMAND hostname -n llctl start options`

Syntax:

`LL_RSH_COMMAND = full_path_to_executable`

Default value: `/usr/bin/rsh`. This keyword must specify the full path name to the executable provided. If no value is specified, LoadLeveler will use `/usr/bin/rsh` as the default when issuing a start. If an error occurred while locating the executable specified, an error message will be displayed.

Example: This example shows that using the secure shell (`/usr/bin/ssh`) is the preferred method for the `llctl start` command to communicate with remote nodes. Specify the following in the configuration file:

```
LL_RSH_COMMAND=/usr/bin/ssh
```

LOADL_ADMIN

Specifies a list of LoadLeveler administrators.

Syntax:

`LOADL_ADMIN = list of user names`

Where *list of user names* is a blank-delimited list of those individuals who will have administrative authority. These users are able to invoke the administrator-only commands such as `llctl`, `llfavorjob`, and `llfavoruser`. These administrators can also invoke the administrator-only GUI functions. For more information, see Chapter 7, “Using LoadLeveler’s GUI to perform administrator tasks,” on page 155.

Default value: No default value is set, which means no one has administrator authority until this keyword is defined with one or more user names.

Example: To grant administrative authority to users bob and mary, enter the following in the configuration file:

```
LOADL_ADMIN = bob mary
```

For more information related to using this keyword, see “Defining LoadLeveler administrators” on page 41.

LOCAL_CONFIG

Specifies the path name of the optional local configuration file containing information specific to a node in the LoadLeveler network.

Syntax:

`LOCAL_CONFIG = directory`

Default value: No default value is set.

Examples:

- If you are using a distributed file system like NFS, some examples are:
`LOCAL_CONFIG = $(tilde)/$(host).LoadL_config.local`
`LOCAL_CONFIG = $(tilde)/LoadL_config.$(host).$(domain)`
`LOCAL_CONFIG = $(tilde)/LoadL_config.local.$(hostname)`

See “LoadLeveler variables” on page 294 for information about the **tilde**, **host**, and **domain** variables.

- If you are using a local file system, an example is:

```
LOCAL_CONFIG = /var/LoadL/LoadL_config.local
```

LOG Defines the local directory to store log files. It is not necessary to keep all the log files created by the various LoadLeveler daemons and programs in one directory, but you will probably find it convenient to do so.

Syntax:

```
LOG = local directory/log
```

Default value: \$(tilde)/log

MACHINE_AUTHENTICATE

Specifies whether machine validation is performed. When set to **true**, LoadLeveler only accepts connections from machines specified in the administration file. When set to **false**, LoadLeveler accepts connections from any machine.

When set to **true**, every communication between LoadLeveler processes will verify that the sending process is running on a machine which is identified via a machine stanza in the administration file. The validation is done by capturing the address of the sending machine when the **accept** function call is issued to accept a connection. The **gethostbyaddr** function is called to translate the address to a name, and the name is matched with the list derived from the administration file.

Syntax:

```
MACHINE_AUTHENTICATE = true | false
```

Default value: false

For more information related to using this keyword, see “Defining a LoadLeveler cluster” on page 41.

MACHINE_UPDATE_INTERVAL

Specifies the time, in seconds, during which machines must report to the central manager.

Syntax:

```
MACHINE_UPDATE_INTERVAL = number
```

Where *number* specifies the time period, in seconds, during which machines must report to the central manager. Machines that do not report in this number of seconds are considered *down*. *number* must be a numerical value and cannot be an arithmetic expression.

Default value: The default is 300 seconds.

For more information related to using this keyword, see “Setting negotiator characteristics and policies” on page 43.

MACHPRIO

Machine priority expression.

Syntax:

```
MACHPRIO = expression
```

Configuration file reference

You can use the following LoadLeveler variables in the **MACHPRIO** expression:

- **LoadAvg**
- **Connectivity**
- **Cpus**
- **Speed**
- **Memory**
- **VirtualMemory**
- **Disk**
- **CustomMetric**
- **MasterMachPriority**
- **ConsumableCpus**
- **ConsumableMemory**
- **ConsumableVirtualMemory**
- **PagesFreed**
- **PagesScanned**
- **FreeRealMemory**

For detailed descriptions of these variables, see “LoadLeveler variables” on page 294.

Default value: (0 - LoadAvg)

Examples:

- Example 1

This example orders machines by the Berkeley one-minute load average.

```
MACHPRIO : 0 - (LoadAvg)
```

Therefore, if **LoadAvg** equals .7, this example would read:

```
MACHPRIO : 0 - (.7)
```

The **MACHPRIO** would evaluate to -.7.

- Example 2

This example orders machines by the Berkeley one-minute load average normalized for machine speed:

```
MACHPRIO : 0 - (1000 * (LoadAvg / (Cpus * Speed)))
```

Therefore, if **LoadAvg** equals .7, **Cpus** equals 1, and **Speed** equals 2, this example would read:

```
MACHPRIO : 0 - (1000 * (.7 / (1 * 2)))
```

This example further evaluates to:

```
MACHPRIO : 0 - (350)
```

The **MACHPRIO** would evaluate to -350.

Notice that if the speed of the machine were increased to 3, the equation would read:

```
MACHPRIO : 0 - (1000 * (.7 / (1 * 3)))
```

The **MACHPRIO** would evaluate to approximately -233. Therefore, as the speed of the machine increases, the **MACHPRIO** also increases.

- Example 3

This example orders machines accounting for real memory and available swap space (remembering that Memory is in Mbytes and VirtualMemory is in Kbytes):

```
MACHPRIO : 0 - (10000 * (LoadAvg / (Cpus * Speed))) +
(10 * Memory) + (VirtualMemory / 1000)
```

- Example 4

This example sets a relative machine priority based on the value of the **CUSTOM_METRIC** keyword.

```
MACHPRIO : CustomMetric
```

To do this, you must specify a value for the **CUSTOM_METRIC** keyword or the **CUSTOM_METRIC_COMMAND** keyword in either the **LoadL_config.local** file of a machine or in the global **LoadL_config** file.

To assign the same relative priority to all machines, specify the **CUSTOM_METRIC** keyword in the global configuration file. For example:

```
CUSTOM_METRIC = 5
```

You can override this value for an individual machine by specifying a different value in that machine's **LoadL_config.local** file.

- Example 5

This example gives master nodes the highest priority:

```
MACHPRIO : (MasterMachPriority * 10000)
```

- Example 6

This example gives nodes the with highest percentage of switch adapters with connectivity the highest priority:

```
MACHPRIO : Connectivity
```

For more information related to using this keyword, see "Setting negotiator characteristics and policies" on page 43.

MAIL Name of a local mail program used to override default mail notification.

Syntax:

```
MAIL = program name
```

Default value: No default value is set.

For more information related to using this keyword, see "Using your own mail program" on page 75.

MASTER

Location of the master executable (**LoadL_master**).

Syntax:

```
MASTER = directory
```

Default value: \$(BIN)/LoadL_master

For more information related to using this keyword, see "How LoadLeveler daemons process jobs" on page 7.

MASTER_COREDUMP_DIR

Local directory for storing **LoadL_master** core dump files.

Syntax:

```
MASTER_COREDUMP_DIR = directory
```

Configuration file reference

Default value: The `/tmp` directory.

For more information related to using this keyword, see “Specifying file and directory locations” on page 45.

MASTER_DGRAM_PORT

The port number used when connecting to the daemon.

Syntax:

`MASTER_DGRAM_PORT = port number`

Default value: The default is 9617.

For more information related to using this keyword, see “Defining network characteristics” on page 45.

MASTER_STREAM_PORT

Specifies the port number to be used when connecting to the daemon.

Syntax:

`MASTER_STREAM_PORT = port number`

Default value: The default is 9616.

For more information related to using this keyword, see “Defining network characteristics” on page 45.

MAX_CKPT_INTERVAL

The maximum number of seconds between checkpoints for running jobs.

Syntax:

`MAX_CKPT_INTERVAL = number`

Default value: 7200 (2 hours)

For more information related to using this keyword, see “LoadLeveler support for checkpointing jobs” on page 129.

MAX_JOB_REJECT

Determines the number of times a job is rejected before it is canceled or put in User Hold or System Hold status.

Syntax:

`MAX_JOB_REJECT = number`

number must be a numerical value and cannot be an arithmetic expression.

MAX_JOB_REJECT may be set to unlimited rejects by specifying a value of `-1`.

Default value: The default value is 0, which indicates a rejected job will immediately be canceled or placed on hold.

For related information, see the `NEGOTIATOR_REJECT_DEFER` keyword.

MAX_RESERVATIONS

Specifies the maximum number of reservations that this LoadLeveler cluster can have. Only reservations in waiting and in use are counted toward this limit; LoadLeveler does not count reservations that have already ended or are in the process of being canceled.

Note: Having too many reservations in a LoadLeveler cluster can have performance impacts. Administrators should select a suitable value for this keyword.

Syntax:

MAX_RESERVATIONS = *number*

The value for this keyword can be 0 or a positive integer.

Default value: The default is 10.

MAX_STARTERS

Specifies the maximum number of tasks that can run simultaneously on a machine. In this case, a task can be a serial job step or a parallel task.

MAX_STARTERS defines the number of initiators on the machine (the number of tasks that can be initiated from a **startd**).

Syntax:

MAX_STARTERS = *number*

Default value: If this keyword is not specified, the default is the number of elements in the **Class** statement.

For more information related to using this keyword, see “Specifying how many jobs a machine can run” on page 52.

MAX_TOP_DOGS

Specifies the maximum total number of top dogs that the central manager daemon will allocate. When scheduling jobs, after **MAX_TOP_DOGS** total top dogs have been allocated, no more will be considered.

Syntax:

MAX_TOP_DOGS = *k* | 1

where: *k* is a non-negative integer specifying the global maximum top dogs limit.

Default value: The default value is 1.

For more information related to using this keyword, see “Using the BACKFILL scheduler” on page 101.

MIN_CKPT_INTERVAL

The minimum number of seconds between checkpoints for running jobs.

Syntax:

MIN_CKPT_INTERVAL = *number*

Default value: 900 (15 minutes)

For more information related to using this keyword, see “LoadLeveler support for checkpointing jobs” on page 129.

NEGOTIATOR

Location of the negotiator executable (**LoadL_negotiator**).

Syntax:

NEGOTIATOR = *directory*

Default value: \$(BIN)/LoadL_negotiator

Configuration file reference

For more information related to using this keyword, see “How LoadLeveler daemons process jobs” on page 7.

NEGOTIATOR_COREDUMP_DIR

Local directory for storing **LoadL_negotiator** core dump files.

Syntax:

NEGOTIATOR_COREDUMP_DIR = *directory*

Default value: The **/tmp** directory.

For more information related to using this keyword, see “Specifying file and directory locations” on page 45.

NEGOTIATOR_CYCLE_DELAY

Specifies the minimum time, in seconds, the negotiator delays between periods when it attempts to schedule jobs. This time is used by the negotiator daemon to respond to queries, reorder job queues, collect information about changes in the states of jobs, and so on. Delaying the scheduling of jobs might improve the overall performance of the negotiator by preventing it from spending excessive time attempting to schedule jobs.

Syntax:

NEGOTIATOR_CYCLE_DELAY = *number*

number must be a numerical value and cannot be an arithmetic expression.

Default value: The default is 0 seconds

NEGOTIATOR_CYCLE_TIME_LIMIT

Specifies the maximum amount of time, in seconds, that LoadLeveler will allow the negotiator to spend in one cycle trying to schedule jobs. The negotiator cycle will end, after the specified number of seconds, even if there are additional jobs waiting for dispatch. Jobs waiting for dispatch will be considered at the next negotiator cycle. The **NEGOTIATOR_CYCLE_TIME_LIMIT** keyword applies only to the BACKFILL scheduler.

Syntax:

NEGOTIATOR_CYCLE_TIME_LIMIT = *number*

Where *number* must be a positive integer or zero and cannot be an arithmetic expression.

Default value: If the keyword value is not specified or a value of zero is used, the negotiator cycle will be unlimited.

NEGOTIATOR_INTERVAL

The time interval, in seconds, at which the negotiator daemon updates the status of jobs in the LoadLeveler cluster and negotiates with machines that are available to run jobs.

Syntax:

NEGOTIATOR_INTERVAL = *number*

Where *number* specifies the interval, in seconds, at which the negotiator daemon performs a “negotiation loop” during which it attempts to assign available machines to waiting jobs. A negotiation loop also occurs

whenever job states or machine states change. *number* must be a numerical value and cannot be an arithmetic expression.

When this keyword is set to zero, the central manager's automatic scheduling activity is been disabled, and LoadLeveler will not attempt to schedule any jobs unless instructed to do so through the `llrunscheduler` command or `ll_run_scheduler` subroutine.

Default value: The default is 30 seconds.

For more information related to using this keyword, see "Controlling the central manager scheduling cycle" on page 67.

NEGOTIATOR_LOADAVG_INCREMENT

Specifies the value the negotiator adds to the startd machine's load average whenever a job in the Pending state is queued on that machine. This value is used to compensate for the increased load caused by starting another job.

Syntax:

`NEGOTIATOR_LOADAVG_INCREMENT = number`

number must be a numerical value and cannot be an arithmetic expression.

Default value: The default value is .5

NEGOTIATOR_PARALLEL_DEFER

Specifies the amount of time, in seconds, that defines how long a job stays out of the queue after it fails to get the correct number of processors. This keyword applies only to the default LoadLeveler scheduler. This keyword must be greater than the `NEGOTIATOR_INTERVAL` value; if it is not, the default is used.

Syntax:

`NEGOTIATOR_PARALLEL_DEFER = number`

number must be a numerical value and cannot be an arithmetic expression.

Default value: The default is `NEGOTIATOR_INTERVAL` multiplied by 5.

NEGOTIATOR_PARALLEL_HOLD

Specifies the amount of time, in seconds, that defines how long a job is given to accumulate processors. This keyword applies only to the default LoadLeveler scheduler. This keyword must be greater than the `NEGOTIATOR_INTERVAL` value; if it is not, the default is used.

Syntax:

`NEGOTIATOR_PARALLEL_HOLD = number`

number must be a numerical value and cannot be an arithmetic expression.

Default value: The default is `NEGOTIATOR_INTERVAL` multiplied by 5.

NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL

Specifies the amount of time, in seconds, between calculation of the `SYSPRIO` values for waiting jobs. Recalculating the priority can be CPU-intensive; specifying low values for the `NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL` keyword may

Configuration file reference

lead to a heavy CPU load on the **negotiator** if a large number of jobs are running or waiting for resources. A value of 0 means the **SYSPRIO** values are not recalculated.

You can use this keyword to base the order in which jobs are run on the current number of running, queued, or total jobs for a user or a group.

Syntax:

`NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL = number`

number must be a numerical value and cannot be an arithmetic expression.

Default value: The default is 120 seconds.

NEGOTIATOR_REJECT_DEFER

Specifies the amount of time in seconds the negotiator waits before it considers scheduling a job to a machine that recently rejected the job.

Syntax:

`NEGOTIATOR_REJECT_DEFER = number`

number must be a numerical value and cannot be an arithmetic expression.

Default value: The default is 120 seconds.

For related information, see the **MAX_JOB_REJECT** keyword.

NEGOTIATOR_REMOVE_COMPLETED

Specifies the amount of time, in seconds, that you want the negotiator to keep information regarding completed and removed jobs so that you can query this information using the **llq** command.

Syntax:

`NEGOTIATOR_REMOVE_COMPLETED = number`

number must be a numerical value and cannot be an arithmetic expression.

Default value: The default is 0 seconds.

NEGOTIATOR_RESCAN_QUEUE

specifies the amount of time in seconds that defines how long the negotiator waits to rescan the job queue for machines which have bypassed jobs which could not run due to conditions which may change over time. This keyword must be greater than the **NEGOTIATOR_INTERVAL** value; if it is not, the default is used.

Syntax:

`NEGOTIATOR_RESCAN_QUEUE = number`

number must be a numerical value and cannot be an arithmetic expression.

Default value: The default is 900 seconds.

NEGOTIATOR_STREAM_PORT

Specifies the port number used when connecting to the daemon.

Syntax:

`NEGOTIATOR_STREAM_PORT = port number`

Default value: The default is 9614.

For more information related to using this keyword, see “Defining network characteristics” on page 45.

OBITUARY_LOG_LENGTH

Specifies the number of lines from the end of the file that are appended to the mail message. The master daemon mails this log to the LoadLeveler administrators when one of the daemons dies.

Syntax:

OBITUARY_LOG_LENGTH = *number*

number must be a numerical value and cannot be an arithmetic expression.

Default value: The default is 25.

POLLING_FREQUENCY

Specifies the interval, in seconds, with which the startd daemon evaluates the load on the local machine and decides whether to suspend, resume, or abort jobs. This time is also the minimum interval at which the kbdd daemon reports keyboard or mouse activity to the startd daemon.

Syntax:

POLLING_FREQUENCY = *number*

number must be a numerical value and cannot be an arithmetic expression.

Default value: The default is 5.

POLLS_PER_UPDATE

Specifies how often, in **POLLING_FREQUENCY** intervals, startd daemon updates the central manager. Due to the communication overhead, it is impractical to do this with the frequency defined by the **POLLING_FREQUENCY** keyword. Therefore, the startd daemon only updates the central manager every *n*th (where *n* is the number specified for **POLLS_PER_UPDATE**) local update. Change **POLLS_PER_UPDATE** when changing the **POLLING_FREQUENCY**.

Syntax:

POLLS_PER_UPDATE = *number*

number must be a numerical value and cannot be an arithmetic expression.

Default value: The default is 24.

PRESTARTED_STARTERS

Specifies how many prestarted starter processes LoadLeveler will maintain on an execution node to manage jobs when they arrive. The startd daemon starts the number of starter processes specified by this keyword. You may specify this keyword in either the global or local configuration file.

Syntax:

PRESTARTED_STARTERS = *number*

number must be less than or equal to the value specified through the **MAX_STARTERS** keyword. If the value of **PRESTARTED_STARTERS** specified is greater than **MAX_STARTERS**, LoadLeveler records a warning message in the startd log and assigns **PRESTARTED_STARTERS** the same value as **MAX_STARTERS**.

Configuration file reference

If the value `PRESTARTED_STARTERS` is zero, no starter processes will be started before jobs arrive on the execution node.

Default value: The default is 1.

PREEMPT_CLASS

Defines the preemption rule for a job class.

Syntax: The following forms illustrate correct syntax.

```
PREEMPT_CLASS[incoming_class] = ALL[:preempt_method] {  
outgoing_class1 [outgoing_class2 ...] }
```

Using this form, `ALL` indicates that job steps of *incoming_class* have priority and will not share nodes with job steps of *outgoing_class1*, *outgoing_class2*, or other outgoing classes. If a job step of the *incoming_class* is to be started on a set of nodes, all job steps of *outgoing_class1*, *outgoing_class2*, or other outgoing classes running on those nodes will be preempted.

Note: The `ALL` preemption rule does not apply to Blue Gene jobs.

```
PREEMPT_CLASS[incoming_class] = ENOUGH[:preempt_method] {  
outgoing_class1 [outgoing_class2 ...] }
```

Using this form, `ENOUGH` indicates that job steps of *incoming_class* will share nodes with job steps of *outgoing_class1*, *outgoing_class2*, or other outgoing classes if there are sufficient resources. If a job step of the *incoming_class* is to be started on a set of nodes, one or more job steps of *outgoing_class1*, *outgoing_class2*, or other outgoing classes running on those nodes may be preempted to get needed resources.

Combinations of these forms are also allowed.

Notes:

1. The optional specification *preempt_method* indicates which method LoadLeveler is to use to preempt the jobs; this specification is valid only for the `BACKFILL` scheduler. Valid values for this specification in keyword syntax are the highlighted abbreviations in parentheses:
 - Remove (**rm**)
 - System hold (**sh**)
 - Suspend (**su**)
 - Vacate (**vc**)
 - User hold (**uh**)

LoadLeveler for Linux does not support the suspend method of preemption. For more information about preemption methods, see “Steps for configuring a scheduler to preempt jobs” on page 120.

2. Using the “`ALL`” value in the `PREEMPT_CLASS` keyword places implied restrictions on when a job can start. See “Planning to preempt jobs” on page 117 for more information.
3. The incoming class is designated inside [] brackets.
4. Outgoing classes are designated inside { } curly braces.
5. The job classes on the right hand (outgoing) side of the statement must be different from incoming class, or it may be **allclasses**. If the outgoing side is defined as **allclasses** then all job classes are preemptable with the exception of the incoming class specified within brackets.

6. A class name or **allclasses** should not be in both the ALL list and the ENOUGH list. If you do so, the entire statement will be ignored. An example of this is:
PREEMPT_CLASS[Class_A]=ALL{allclasses} ENOUGH {allclasses}
7. If you use **allclasses** as an outgoing (preemptable) class, then no other class names should be listed at the right hand side as the entire statement will be ignored. An example of this is:
PREEMPT_CLASS[Class_A]=ALL{Class_B} ENOUGH {allclasses}
8. More than one ALL statement and more than one ENOUGH statement may appear at the right hand side. Multiple statements have a cumulative effect.
9. Each ALL or ENOUGH statement can have multiple class names inside the curly braces. However, a blank space delimiter is required between each class name.
10. Both the ALL and ENOUGH statements can include an optional specification indicating the method LoadLeveler will use to preempt the jobs. Valid values for this specification are listed in the description of the DEFAULT_PREEMPT_METHOD keyword. If a value is specified on the PREEMPT_CLASS ALL or ENOUGH statement, that value overrides the value set on the DEFAULT_PREEMPT_METHOD keyword, if any.
11. ALL and ENOUGH may be in mixed cases.
12. Spaces are allowed around the brackets and curly braces.
13. PREEMPT_CLASS [allclasses] will be ignored.

Default value: No default value is set.

Examples:

PREEMPT_CLASS[Class_B]=ALL{Class_E Class_D} ENOUGH {Class_C}

This indicates that all Class_E jobs and all Class_D jobs and enough Class_C jobs will be preempted to enable an incoming Class_B job to run.

PREEMPT_CLASS[Class_D]=ENOUGH:VC {Class_E}

This indicates that zero, one, or more Class_E jobs will be preempted using the vacate method to enable an incoming Class_D job to run.

PREEMPTION_SUPPORT

For the BACKFILL or API schedulers only, specifies the level of preemption support for a cluster.

Syntax:

PREEMPTION_SUPPORT= full | no_adapter | none

- When set to **full**, preemption is fully supported.
- When set to **no_adapter**, preemption is supported but the adapter resources are not released by preemption.
- When set to **none**, preemption is not supported, and preemption requests will be rejected.

Notes:

1. If the value of this keyword is set to any value other than **none** for the default scheduler, LoadLeveler will not start.

Configuration file reference

2. For the BACKFILL or API scheduler, when this keyword is set to **full** or **no_adapter** and preemption by the suspend method is required, the configuration keyword **PROCESS_TRACKING** must be set to **true**.

Default value: The default value for all schedulers is **none**; if you want to enable preemption under these schedulers, you must set a value for this keyword.

PROCESS_TRACKING

Specifies whether or not LoadLeveler will cancel any processes (throughout the entire cluster), left behind when a job terminates.

Restriction: Process tracking is ignored by LoadLeveler for Linux.

Syntax:

PROCESS_TRACKING = TRUE | FALSE

When **TRUE** ensures that when a job is terminated, no processes created by the job will continue running.

Note: It is necessary to set this keyword to **true** to do preemption by the suspend method with the BACKFILL or API scheduler.

Default value: FALSE

PROCESS_TRACKING_EXTENSION

Specifies the directory containing the kernel extension binary **LoadL_pt_ke**.

Restriction: Process tracking is ignored by LoadLeveler for Linux.

Syntax:

PROCESS_TRACKING_EXTENSION = *directory*

Default value: The directory **\$HOME/bin**

For more information related to using this keyword, see “Tracking job processes” on page 64.

PUBLISH_OBITUARIES

Specifies whether or not the master daemon sends mail to the administrator when any daemon it manages ends abnormally. When set to **true**, this keyword specifies that the master daemon sends mail to the administrators identified by **LOADL_ADMIN** keyword.

Syntax:

PUBLISH_OBITUARIES = true | false

Default value: **true**

REJECT_ON_RESTRICTED_LOGIN

Specifies whether the user’s account status will be checked on every node where the job will be run by calling the AIX **loginrestrictions** function with the **S_DIST_CLNT** flag.

Restriction: Login restriction checking is ignored by LoadLeveler for Linux.

Login restriction checking includes:

- Does the account still exist?
- Is the account locked?
- Has the account expired?
- Do failed login attempts exceed the limit for this account?

- Is login disabled via `/etc/nologin`?

If the AIX `loginrestrictions` function indicates a failure then the user's job will be rejected and will be processed according to the LoadLeveler configuration parameters `MAX_JOB_REJECT` and `ACTION_ON_MAX_REJECT`.

Syntax:

`REJECT_ON_RESTRICTED_LOGIN = true | false`

Default value: `false`

RELEASEDIR

Defines the directory where all the LoadLeveler software resides.

Syntax:

`RELEASEDIR = release directory`

Default value: `$(RELEASEDIR)`

RESERVATION_CAN_BE_EXCEEDED

Specifies whether LoadLeveler will schedule job steps that are bound to a reservation when their end times (based on hard wall-clock limits) exceed the reservation end time.

Syntax:

`RESERVATION_CAN_BE_EXCEEDED = true | false`

When this keyword is set to `false`, LoadLeveler schedules only those job steps that will complete before the reservation ends. When set to `true`, LoadLeveler schedules job steps to run under a reservation even if their end times are expected to exceed the reservation end time. When the reservation ends, however, the reserved nodes no longer belong to the reservation, and so these nodes might not be available for the jobs to continue running. In this case, LoadLeveler might preempt the running jobs.

Note that this keyword setting does not change the actual end time of the reservation. It only affects how LoadLeveler manages job steps whose end times exceed the end time of the reservation.

Default value: `true`

RESERVATION_HISTORY

Defines the name of a file that is to contain the local history of reservations.

Syntax:

`RESERVATION_HISTORY = file name`

LoadLeveler appends a single line to the reservation history file for each reservation. For an example, see "Collecting accounting data for reservations" on page 126.

Default value: `$(SPOOL)/reservation_history`

RESERVATION_MIN_ADVANCE_TIME

Specifies the minimum time, in minutes, between the time at which a reservation is created and the time at which the reservation is to start.

Configuration file reference

Syntax:

RESERVATION_MIN_ADVANCE_TIME = *number of minutes*

By default, the earliest time at which a reservation may start is the current time plus the value set for the **RESERVATION_SETUP_TIME** keyword.

Default value: 0 (zero)

RESERVATION_PRIORITY

Specifies whether LoadLeveler administrators may reserve nodes on which running jobs are expected to end after the reservation start time. This keyword value applies only for LoadLeveler administrators; other reservation owners do not have this capability.

Syntax:

RESERVATION_PRIORITY = NONE | HIGH

When you set this keyword to HIGH, before activating the reservation, LoadLeveler preempts the job steps running on the reserved nodes (Blue Gene job steps are handled the same way). The only exceptions are non-preemptable jobs; LoadLeveler will not preempt those jobs because of any reservations.

Default value: NONE

RESERVATION_SETUP_TIME

Specifies how much time, in seconds, that LoadLeveler may use to prepare for a reservation before it is to start. The tasks that LoadLeveler performs during this time include checking and reporting node conditions, and preempting job steps still running on the reserved nodes.

For a given reservation, LoadLeveler uses the **RESERVATION_SETUP_TIME** keyword value that is set at the time that the reservation is created, not whatever value might be set when the reservation starts. If the start time of the reservation is modified, however, LoadLeveler uses the **RESERVATION_SETUP_TIME** keyword value that is set at the time of the modification.

Syntax:

RESERVATION_SETUP_TIME = *number of seconds*

Default value: 60

RESTARTS_PER_HOUR

Specifies how many times the master daemon attempts to restart a daemon that dies abnormally. Because one or more of the daemons may be unable to run due to a permanent error, the master only attempts **\$(RESTARTS_PER_HOUR)** restarts within a 60 minute period. Failing that, it sends mail to the administrators identified by the **LOADL_ADMIN** keyword and exits.

Syntax:

RESTARTS_PER_HOUR = *number*

number must be a numerical value and cannot be an arithmetic expression.

Default value: The default is 12.

RESUME_ON_SWITCH_TABLE_ERROR_CLEAR

Specifies whether or not the **startd** that was drained when the switch table failed to unload will automatically resume once the unload errors are cleared. The unload error is considered cleared after LoadLeveler can successfully unload the switch table. For this keyword to work, the **DRAIN_ON_SWITCH_TABLE_ERROR** option in the configuration file must be turned on and not disabled. Flushing, suspending, or draining of a **startd** manually or automatically will disable this option until the **startd** is manually resumed.

Syntax:

RESUME_ON_SWITCH_TABLE_ERROR_CLEAR = true | **false**

Default value: false

RSET_SUPPORT

Indicates the level of RSet support present on a machine.

Restriction: RSET support is not available on Linux platforms.

Syntax:

RSET_SUPPORT = *option*

The available options are:

RSET_CONSUMABLE_CPUS

Indicates that the jobs scheduled to the machine will be attached to RSets with the number of CPUs specified by the consumableCPUs variable.

RSET_MCM_AFFINITY

Indicates the machine can run jobs requesting memory and adapter affinity.

RSET_NONE Indicates LoadLeveler RSet support is not available on the machine.

RSET_USER_DEFINED

Indicates the machine can be used for jobs with a user-created RSet in their job command file.

Default value: RSET_NONE

SAVELOGS

Specifies the directory in which log files are archived.

Syntax:

SAVELOGS = *directory*

Where *directory* is the directory in which log files will be archived.

Default value: No default value is set.

For more information related to using this keyword, see “Configuring recording activity and log files” on page 46.

SCHEDD

Location of the Schedd executable (**LoadL_schedd**).

Syntax:

SCHEDD = *directory*

Configuration file reference

Default value: \$(BIN)/LoadL_schedd

For more information related to using this keyword, see “How LoadLeveler daemons process jobs” on page 7.

SCHEDD_COREDUMP_DIR

Specifies the local directory for storing **LoadL_schedd** core dump files.

Syntax:

SCHEDD_COREDUMP_DIR = *directory*

Default value: The **/tmp** directory.

For more information related to using this keyword, see “Specifying file and directory locations” on page 45.

SCHEDD_INTERVAL

Specifies the interval, in seconds, at which the Schedd daemon checks the local job queue and updates the negotiator daemon.

Syntax:

SCHEDD_INTERVAL = *number*

number must be a numerical value and cannot be an arithmetic expression.

Default value: The default is 60 seconds.

SCHEDD_RUNS_HERE

Specifies whether the Schedd daemon runs on the host. If you do not want to run the Schedd daemon, specify **false**.

This keyword does not designate a machine as a public scheduling machine. Unless configured as a public scheduling machine, a machine configured to run the Schedd daemon will only accept job submissions from the same machine running the Schedd daemon. A public scheduling machine accepts job submissions from other machines in the LoadLeveler cluster. To configure a machine as a public scheduling machine, see the **schedd_host** keyword description in “Administration file keyword descriptions” on page 306.

Syntax:

SCHEDD_RUNS_HERE = **true** | **false**

Default value: **true**

SCHEDD_SUBMIT_AFFINITY

Specifies whether job submissions are directed to a locally running Schedd daemon. When the keyword is set to **true**, job submissions are directed to a Schedd daemon running on the same machine where the submission takes place, provided there is a Schedd daemon running on that machine. In this case the submission is said to have “affinity” for the local Schedd daemon. If there is no Schedd daemon running on the machine where the submission takes place, or if this keyword is set to **false**, the job submission will only be directed to a Schedd daemon serving as a public scheduling machine. In this case, if there are no public scheduling machines configured the job cannot be submitted. A public scheduling machine accepts job submissions from other machines in the LoadLeveler

cluster. To configure a machine as a public scheduling machine, see the **schedd_host** keyword description in “Administration file keyword descriptions” on page 306.

Installations with a large number of nodes should consider setting this keyword to **false** to more evenly distribute dispatching of jobs among the Schedd daemons. For more information, see “Scaling considerations” on page 666.

Syntax:

SCHEDD_SUBMIT_AFFINITY = **true** | false

Default value: true

SCHEDD_STATUS_PORT

Specifies the port number used when connecting to the daemon.

Syntax:

SCHEDD_STATUS_PORT = *port number*

Default value: The default is 9606.

For more information related to using this keyword, see “Defining network characteristics” on page 45.

SCHEDD_STREAM_PORT

Specifies the port number used when connecting to the daemon.

Syntax:

SCHEDD_STREAM_PORT = *port number*

Default value: The default is 9605.

For more information related to using this keyword, see “Defining network characteristics” on page 45.

SCHEDULE_BY_RESOURCES

Specifies which consumable resources are considered by the LoadLeveler schedulers. Each consumable resource name may be an administrator-defined alphanumeric string, or may be one of the following predefined resources:

- **ConsumableCpus**
- **ConsumableMemory**
- **ConsumableVirtualMemory**
- **RDMA**

Each string may only appear in the list once. These resources are either floating resources, or machine resources. If any resource is specified incorrectly with the **SCHEDULE_BY_RESOURCES** keyword, then all scheduling resources will be ignored.

Syntax:

SCHEDULE_BY_RESOURCES = *name name ... name*

Default value: No default value is set.

SCHEDULER_TYPE

Specifies the LoadLeveler scheduling algorithm:

Configuration file reference

LL_DEFAULT

Specifies the default LoadLeveler scheduling algorithm. If SCHEDULER_TYPE has not been defined, LoadLeveler will use the default scheduler (LL_DEFAULT).

BACKFILL

Specifies the LoadLeveler BACKFILL scheduler. When you specify this keyword, you should use only the default settings for the **START** expression and the other job control expressions described in “Managing job status through control expressions” on page 63.

API Specifies that you will use an external scheduler. External schedulers communicate to LoadLeveler through the job control API. For more information on setting an external scheduler, see “Using an external scheduler” on page 105.

Syntax:

SCHEDULER_TYPE = LL_DEFAULT | BACKFILL | API

Default value: LL_DEFAULT

Notes:

1. If a scheduler type is not set, LoadLeveler will start, but it will use the default scheduler.
2. If you have set **SCHEDULER_TYPE** with an option that is not valid, LoadLeveler will not start.
3. If you change the scheduler option specified by **SCHEDULER_TYPE**, you must stop and restart LoadLeveler using **llctl** or recycle using **llctl**.

For more information related to using this keyword, see “Defining a LoadLeveler cluster” on page 41.

SEC_ADMIN_GROUP

When security services are enabled, this keyword points to the name of the UNIX group that contains the local identities of the LoadLeveler administrators.

Restriction: CtSec security is not supported on LoadLeveler for Linux.

Syntax:

SEC_ADMIN_GROUP = *name of lladmin group*

Default value: No default value is set.

For more information related to using this keyword, see “Configuring LoadLeveler to use cluster security services” on page 53.

SEC_ENABLEMENT

Specifies the security mechanism to be used.

Restriction: Do not set this keyword to CtSec in the configuration file for a Linux machine. CtSec security is not supported on LoadLeveler for Linux.

Syntax:

SEC_ENABLEMENT = COMPAT | CTSEC

Default value: No default value is set.

SEC_SERVICES_GROUP

When security services are enabled, this keyword specifies the name of the LoadLeveler services group.

Restriction: CtSec security is not supported on LoadLeveler for Linux.

Syntax:

SEC_SERVICES_GROUP=*group name*

Where *group name* defines the identities of the LoadLeveler daemons.

Default value: No default value is set.

SEC_IMPOSED_MECHS

Specifies a blank-delimited list of LoadLeveler's permitted security mechanisms when Cluster Security (CtSec) services are enabled.

Restriction: CtSec security is not supported on LoadLeveler for Linux.

Syntax: Specify a blank delimited list containing combinations of the following values:

- none** If this is the only value specified, then users *will* run unauthenticated and, if authorization is necessary, the job will fail. If this is not the only value specified, then users *may* run unauthenticated and, if authorization is necessary, the job will fail.
- unix** If this is the only value specified, then UNIX host-based authentication will be used; otherwise, other mechanisms may be used.

Default value: No default value is set.

Example:

SEC_IMPOSED_MECHS = none unix

SPOOL

Defines the local directory where LoadLeveler keeps the local job queue and checkpoint files

Syntax:

SPOOL = *local directory/spool*

Default value: \$(tilde)/spool

START

Determines whether a machine can run a LoadLeveler job.

Syntax:

START: *expression that evaluates to T or F (true or false)*

When the expression evaluates to **T**, LoadLeveler considers dispatching a job to the machine. When you use a START expression that is based on the CPU load average, the negotiator may evaluate the expression as **F** even though the load average indicates the machine is Idle. This is because the negotiator adds a compensating factor to the startd machine's load average every time the negotiator assigns a job. For more information, see "the NEGOTIATOR_INTERVAL keyword" on page 270.

Default value: No default value is set, which means that no jobs will be started.

Configuration file reference

For information about time-related variables that you may use for this keyword, see “Variables to use for setting times” on page 299.

START_CLASS

Specifies the rule for starting a job of the *incoming_class*. The START_CLASS rule is applied whenever the BACKFILL scheduler decides whether a job step of the *incoming_class* should start or not.

Syntax:

```
START_CLASS[incoming_class] = (start_class_expression) [ && (start_class_expression) ...]
```

Where *start_class_expression* takes the form:

run_class < number_of_tasks

Which indicates that a job step of the *incoming_class* is only allowed to run on a node when the number of tasks of *run_class* running on that node is less than *number_of_tasks*.

Notes:

1. START_CLASS [**allclasses**] will be ignored.
2. The job class specified by *run_class* may be the same as or different from the class specified by *incoming_class*.
3. You can also define *run_class* as **allclasses**. If you do, the total number of all job tasks running on that node cannot exceed the value specified by *number_of_tasks*.
4. A class name or **allclasses** should not appear twice on the right-hand side of the keyword statement. However, you can use other class names with **allclasses** on the right hand side of the statement.
5. If there is more than one *start_class_expression*, you must use && between adjacent *start_class_expressions*.
6. Both the START keyword and the START_CLASS keyword have to be true before a new job can start.
7. Parenthesis () are optional around *start_class_expression*.

For information related to using this keyword, see “Planning to preempt jobs” on page 117.

Default value: No default value is set.

Examples:

```
START_CLASS[Class_A] = (Class_A < 1)
```

This statement indicates that a Class_A job can only start on nodes that do not have any Class_A jobs running.

```
START_CLASS[Class_B] = allclasses < 5
```

This statement indicates that a Class_B job can only start on nodes with maximum 4 tasks running.

START_DAEMONS

Specifies whether to start the LoadLeveler daemons on the node.

Syntax:

```
START_DAEMONS = true | false
```

Default value: true

When **true**, the daemons are started. In most cases, you will probably want to set this keyword to **true**. An example of why this keyword would be set to **false** is if you want to run the daemons on most of the machines in the cluster but some individual users with their own local configuration files do not want their machines to run the daemons. The individual users would modify their local configuration files and set this keyword to **false**. Because the global configuration file has the keyword set to **true**, their individual machines would still be able to participate in the LoadLeveler cluster.

Also, to define the machine as strictly a submit-only machine, set this keyword to **false**.

STARTD

Location of the startd executable (**LoadL_startd**).

Syntax:

STARTD = *directory*

Default value: \$(BIN)/LoadL_startd

For more information related to using this keyword, see “How LoadLeveler daemons process jobs” on page 7.

STARTD_COREDUMP_DIR

Local directory for storing **LoadL_startd** core dump files.

Syntax:

STARTD_COREDUMP_DIR = *directory*

Default value: The **/tmp** directory.

For more information related to using this keyword, see “Specifying file and directory locations” on page 45.

STARTD_DGRAM_PORT

Specifies the port number used when connecting to the daemon.

Syntax:

STARTD_DGRAM_PORT = *port number*

Default value: The default is 9615.

For more information related to using this keyword, see “Defining network characteristics” on page 45.

STARTD_RUNS_HERE = **true** | **false**

Specifies whether the startd daemon runs on the host. If you do not want to run the startd daemon, specify **false**.

Syntax:

STARTD_RUNS_HERE = **true** | **false**

Default value: **true**

STARTD_STREAM_PORT

Specifies the port number used when connecting to the daemon.

Syntax:

STARTD_STREAM_PORT = *port number*

Configuration file reference

Default value: The default is 9611.

For more information related to using this keyword, see “Defining network characteristics” on page 45.

STARTER

Location of the starter executable (**LoadL_starter**).

Syntax:

STARTER = *directory*

Default value: \$(BIN)/LoadL_starter

For more information related to using this keyword, see “How LoadLeveler daemons process jobs” on page 7.

STARTER_COREDUMP_DIR

Local directory for storing **LoadL_starter** coredump files.

Syntax:

STARTER_COREDUMP_DIR = *directory*

Default value: The /tmp directory.

For more information related to using this keyword, see “Specifying file and directory locations” on page 45.

SUBMIT_FILTER

Specifies the program you want to run to filter a job script when the job is submitted.

Syntax:

SUBMIT_FILTER = *full_path_to_executable*

Where *full_path_to_executable* is called with the job command file as the standard input. The standard output is submitted to LoadLeveler. If the program returns with a nonzero exit code, the job submission is canceled. A submit filter can only make changes to LoadLeveler job command file keyword statements.

Default value: No default value is set.

Multicluster use: In a multicluster environment, if you specified a valid cluster list with either the **llsubmit -X** option or the **ll_cluster** API, then the **SUBMIT_FILTER** will instead be invoked with a modified job command file that contains a **cluster_list** keyword generated from either the **llsubmit -X** option or the **ll_cluster** API.

The modified job command file will contain an inserted `# @ cluster_list = cluster` statement just prior to the first `# @ queue` statement. This **cluster_list** statement takes precedence and overrides all previous specifications of any **cluster_list** statements from the original job command file.

Example: SUBMIT_FILTER in a multicluster environment

The following job command file, `job.cmd`, requests to be run remotely on `cluster1`:

```
#!/bin/sh
# @ cluster_list = cluster1
# @ error = job1.${Host}.${Cluster}.${Process}.err
# @ output = job1.${Host}.${Cluster}.${Process}.out
# @ queue
```

After issuing `llsubmit -X cluster2 job.cmd`, the modified job command file statements will be run on cluster2:

```
#!/bin/sh
# @ cluster_list = cluster1
# @ error = job1.${Host}.${Cluster}.${Process}.err
# @ output = job1.${Host}.${Cluster}.${Process}.out
# @ cluster_list = cluster2
# @ queue
```

For more information related to using this keyword, see “Filtering a job script” on page 70.

SUSPEND

Determines whether running jobs should be suspended.

Syntax:

`SUSPEND: expression that evaluates to T or F (true or false)`

When `T`, LoadLeveler temporarily suspends jobs currently running on the machine. Suspended LoadLeveler jobs will either be continued or vacated. This keyword is not supported for parallel jobs.

Default value: No default value is set.

For information about time-related variables that you may use for this keyword, see “Variables to use for setting times” on page 299.

SYSPRIO

System priority expression.

Syntax:

`SYSPRIO : expression`

You can use the following LoadLeveler variables to define the **SYSPRIO** expression:

- **ClassSysprio**
- **GroupQueuedJobs**
- **GroupRunningJobs**
- **GroupSysprio**
- **GroupTotalJobs**
- **GroupTotalShares**
- **GroupUsedBgShares**
- **GroupUsedShares**
- **JobIsBlueGene**
- **QDate**
- **UserPrio**
- **UserQueuedJobs**
- **UserRunningJobs**
- **UserSysprio**
- **UserTotalJobs**
- **UserTotalShares**
- **UserUsedBgShares**
- **UserUsedShares**

Configuration file reference

For detailed descriptions of these variables, see “LoadLeveler variables” on page 294.

Default value: 0 (zero)

Notes:

1. The **SYSPRIO** keyword is valid only on the machine where the central manager is running. Using this keyword in a local configuration file has no effect.
2. It is recommended that you do not use **UserPrio** in the **SYSPRIO** expression, since user jobs are already ordered by **UserPrio**.
3. The string **SYSPRIO** can be used as both the name of an expression (**SYSPRIO: *value***) and the name of a variable (**SYSPRIO = *value***).

To specify the expression to be used to calculate job priority you must use the syntax for the **SYSPRIO** expression. If the variable is mistakenly used for the **SYSPRIO** expression, which requires a colon (:) after the name, the job priority value will always be 0 because the **SYSPRIO** expression has not been defined.

4. When the **UserRunningJobs**, **GroupRunningJobs**, **UserQueuedJobs**, **GroupQueuedJobs**, **UserTotalJobs**, **GroupTotalJobs**, **GroupTotalShares**, **GroupUsedShares**, **UserTotalShares**, **UserUsedShares**, **GroupUsedBgShares**, **JobIsBlueGene**, and **UserUsedBgShares** variables are used to prioritize the queue based on current usage, you should also set **NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL** so that the priorities are adjusted according to current usage rather than usage only at submission time.

Examples:

- Example 1

This example creates a FIFO job queue based on submission time:

```
SYSPRIO : 0 - (QDate)
```

- Example 2

This example accounts for Class, User, and Group system priorities:

```
SYSPRIO : (ClassSysprio * 100) + (UserSysprio * 10) + (GroupSysprio * 1) - (QDate)
```

- Example 3

This example orders the queue based on the number of jobs a user is currently running. The user who has the fewest jobs running is first in the queue. You should set

NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL in conjunction with this **SYSPRIO** expression.

```
SYSPRIO : 0 - UserRunningJobs
```

- Example 4

This example shows one possible way to set up the **SYSPRIO** expression for fair share scheduling. For those jobs whose owner has no unused shares ($\$(UserHasShares)= 0$), that job priority depends only on **QDate**, making it a simple FIFO queue as in Example 1.

For those jobs whose owner has unused shares ($\$(UserHasShares)= 1$), job priority depends not only on **QDate**, but also on a uniform boost of 31 536 000 (the equivalent to the job being submitted one year earlier).

These jobs still have priority differences because of submit time

differences. It is like forming two priority tiers: the higher priority tier for jobs with unused shares and the lower priority tier for jobs without unused shares.

`SYSPRIO: 31536000 * $(UserHasShares) - QDate`

- Example 5

This example divides the jobs into three priority tiers:

- Those jobs whose owner and group both have unused shares are at the top tier
- Those jobs whose owner or group has unused shares are at the middle tier
- Those jobs whose owner and group both have no shares remaining are at the bottom tier

A user can submit two jobs to two different groups, the first job to a group with shares remaining and the second job to a group without any unused shares. If the user has unused shares, the first job will belong to the top tier and the second job will belong to the middle tier. If the user has no shares remaining, the first job will belong to the middle tier and the second job will belong to the bottom tier. The jobs in the top tier will be considered to run first, then the jobs in the middle tier, and lastly the jobs in the bottom tier.

`SYSPRIO: 31536000 * ($(UserHasShares)+$(GroupHasShares)) - (QDate)`

For more information related to using this keyword, see “Setting negotiator characteristics and policies” on page 43.

SYSPRIO_THRESHOLD_TO_IGNORE_STEP

Specifies a threshold value for system priority. When the system priority assigned to a job step is less than the value set for this keyword, the scheduler ignores the job, which will remain in Idle state.

Syntax:

`SYSPRIO_THRESHOLD_TO_IGNORE_STEP = integer`

Any integer is a valid value.

Default value: INT_MIN

For more information related to using this keyword, see “Controlling the central manager scheduling cycle” on page 67.

TRUNC_GSMONITOR_LOG_ON_OPEN

When **true**, specifies that the log file is restarted with every invocation of the daemon.

Syntax:

`TRUNC_GSMONITOR_LOG_ON_OPEN = true | false`

Default value: false

For more information related to using this keyword, see “Configuring recording activity and log files” on page 46.

TRUNC_KBDD_LOG_ON_OPEN

When **true**, specifies the log file is restarted with every invocation of the daemon.

Syntax:

Configuration file reference

TRUNC_KBDD_LOG_ON_OPEN = true | false

Default value: false

For more information related to using this keyword, see “Configuring recording activity and log files” on page 46.

TRUNC_MASTER_LOG_ON_OPEN

When **true**, specifies the log file is re started with every invocation of the daemon.

Syntax:

TRUNC_MASTER_LOG_ON_OPEN = true | false

Default value: false

For more information related to using this keyword, see “Configuring recording activity and log files” on page 46.

TRUNC_NEGOTIATOR_LOG_ON_OPEN

When **true**, specifies the log file is restarted with every invocation of the daemon.

Syntax:

TRUNC_NEGOTIATOR_LOG_ON_OPEN = true | false

Default value: false

For more information related to using this keyword, see “Configuring recording activity and log files” on page 46.

TRUNC_SCHEDD_LOG_ON_OPEN

When **true**, specifies the log file is restarted with every invocation of the daemon.

Syntax:

TRUNC_SCHEDD_LOG_ON_OPEN = true | false

Default value: false

For more information related to using this keyword, see “Configuring recording activity and log files” on page 46.

TRUNC_STARTD_LOG_ON_OPEN

When **true**, specifies the log file is restarted with every invocation of the daemon.

Syntax:

TRUNC_STARTD_LOG_ON_OPEN = true | false

Default value: false

For more information related to using this keyword, see “Configuring recording activity and log files” on page 46.

TRUNC_STARTER_LOG_ON_OPEN

When **true**, specifies the log file is restarted with every invocation of the daemon.

Syntax:

TRUNC_STARTER_LOG_ON_OPEN = true | false

Default value: false

For more information related to using this keyword, see “Configuring recording activity and log files” on page 46.

UPDATE_ON_POLL_INTERVAL_ONLY

Specifies whether or not the LoadLeveler **startd** daemons will send machine update transactions to the Central Manager. Normally the LoadLeveler **startd** daemons running on executing nodes will send transactions to the Central Manager to provide updates of machine information at various times. An update is sent every polling interval. The polling interval is calculated by multiplying the values for the two keywords, **POLLING_FREQUENCY** and **POLLS_PER_UPDATE**, specified in the LoadLeveler configuration file.

In addition, updates are sent at other times such as when new jobs are started and when jobs terminate on the executing node. If you have a large and highly active cluster (the workload consists of a large number of short running jobs), the normal method for updating the central manager can add excessive network traffic. **UPDATE_ON_POLL_INTERVAL_ONLY** can help reduce this source of network traffic.

When **true** is specified, the LoadLeveler **startd** daemon will only send machine updates to the Central Manager at every polling interval and not at other times.

Syntax:

UPDATE_ON_POLL_INTERVAL_ONLY = false | true

Default value: false

VACATE

Determines whether suspended jobs should be vacated.

Syntax:

VACATE: *expression that evaluates to T or F (true or false)*

When **T**, suspended LoadLeveler jobs are removed from the machine and placed back into the queue (provided you specify **restart=yes** in the job command file). If a checkpoint was taken, the job restarts from the checkpoint. Otherwise, the job restarts from the beginning.

Default value: No default value is set.

For information about time-related variables that you may use for this keyword, see “Variables to use for setting times” on page 299.

VM_IMAGE_ALGORITHM

Specifies the virtual memory algorithm, which is used for checking the **image_size** requirement. This keyword is used together with the **large_page** job command file keyword to specify which algorithm the Central Manager uses to decide whether a machine has enough virtual memory to run a job step.

This keyword is critical for job steps that must use Large Page memory (specified by the job command file keyword **large_page=M**). If the **VM_IMAGE_ALGORITHM** keyword is set to FREE_PAGING_SPACE,

Configuration file reference

the Large Page job step will never be scheduled to run. This keyword must be set to **FREE_PAGING_SPACE_PLUS_FREE_REAL_MEMORY** to run Large Page jobs.

When **FREE_PAGING_SPACE** is specified, LoadLeveler considers only free paging space when determining if a machine has enough virtual memory to run a job step.

When **FREE_PAGING_SPACE_PLUS_FREE_REAL_MEMORY** is specified and the job step specifies:

- **large_page=N** (does not use Large Page memory), LoadLeveler considers free paging space and free regular memory when determining if a machine has enough virtual memory to run a job step.
- **large_page=Y** (uses Large Page memory, if available), LoadLeveler considers free paging space, free regular memory, and free Large Page memory when determining if a machine has enough virtual memory to run a job step, although Large Page memory is only considered for machines configured to exploit the Large Page feature.
- **large_page=M** (must use Large Page memory), LoadLeveler considers only Large Page memory when determining if a machine has enough virtual memory to run a job step. Only machines configured to exploit the Large Page feature are considered.

IBM suggests that you set this keyword to the value **FREE_PAGING_SPACE_PLUS_FREE_REAL_MEMORY** since more types of virtual memory are considered, increasing the chances of finding a machine with enough virtual memory to run the job step.

Syntax:

VM_IMAGE_ALGORITHM = **FREE_PAGING_SPACE** | FREE_PAGING_SPACE_PLUS_FREE_REAL_MEMORY

Default value: FREE_PAGING_SPACE

WALLCLOCK_ENFORCE

Specifies whether the job command file keyword **wall_clock_limit** will be enforced for this job. The **WALLCLOCK_ENFORCE** keyword is valid only when an external scheduler is enabled.

Syntax:

WALLCLOCK_ENFORCE = **true** | false

Default value: true

X_RUNS_HERE

Specifies whether the kbdd (keyboard) daemon runs on the host. If you do not want to run the kbdd daemon, specify **false**.

Syntax:

X_RUNS_HERE = **true** | false

Default value: true

User-defined keywords

This type of variable, which is generally created and defined by the user, can be named using any combination of letters and numbers. A user-defined variable is set equal to values, where the *value* defines conditions, names files, or sets numeric values. For example, you can create a variable named **MY_MACHINE** and set it equal to the name of your machine named *iron* as follows:

```
MY_MACHINE = iron.ore.met.com
```

You can then identify the keyword using a dollar sign (\$) and parenthesis. For example, the literal **\$(MY_MACHINE)** following the definition in the previous example results in the automatic substitution of **iron.ore.met.com** in place of **\$(MY_MACHINE)**.

User-defined definitions may contain references, enclosed in parenthesis, to previously defined keywords. Therefore:

```
A = xxx
C = $(A)
```

is a valid expression and the resulting value of **C** is *xxx*. Note that **C** is actually bound to **A**, not to its value, so that

```
A = xxx
C = $(A)
A = yyy
```

is also legal and the resulting value of **C** is *yyy*.

The sample configuration file shipped with the product defines and uses the following “user-defined” variables.

BackgroundLoad

Defines the variable **BackgroundLoad** and assigns to it a floating point constant. This might be used as a noise factor indicating no activity.

CPU_Busy

Defines the variable **CPU_Busy** and reassigns to it at each evaluation the Boolean value True or False, depending on whether the Berkeley one-minute load average is equal to or greater than the saturation level of 1.5.

CPU_Idle

Defines the variable **CPU_Idle** and reassigns to it at each evaluation the Boolean value True or False, depending on whether the Berkeley one-minute load average is equal or less than 0.7.

HighLoad

Is a keyword that the user can define to use as a saturation level at which no further jobs should be started.

HOUR

Defines the variable **HOUR** and assigns to it a constant integer value.

JobLoad

Defines the variable **JobLoad** which defines the load on the machine caused by running the job.

KeyboardBusy

Defines the variable **KeyboardBusy** and reassigns to it at each evaluation the Boolean value True or False, depending on whether the keyboard and mouse have been idle for fifteen minutes.

Configuration file reference

LowLoad

Defines the variable **LowLoad** and assigns to it the value of **BackgroundLoad**. This might be used as a restart level at which jobs can be started again and assumes only running 1 job on the machine.

mail

Specifies a local program you want to use in place of the LoadLeveler default mail notification method.

MINUTE

Defines the variable **MINUTE** and assigns to it a constant integer value.

StateTimer

Defines the variable **StateTimer** and reassigns to it at each evaluation the number of seconds since the current state was entered.

LoadLeveler variables

LoadLeveler provides the following variables that you can use in your configuration file statements. LoadLeveler variables are evaluated by the LoadLeveler daemons at various stages. They do not require you to use any special characters (such as a parenthesis or a dollar sign) to identify them.

Arch

Indicates the system architecture. Note that **Arch** is a special case of a LoadLeveler variable called a machine variable. You specify a machine variable using the following format:

```
variable :  $\$(value)$ 
```

ClassSysprio

The priority for the class of the job step, defined in the class stanza in the administration file.

Default: 0

For additional information about using this variable, see the **SYSPRIO** keyword description.

Connectivity

The ratio of the number of active switch adapters on a node to the total number of switch adapters on the node. The value ranges from 0.0 (all switch adapters are down) to 1.0 (all switch adapters are active). A node with no switch adapters has a connectivity of 0.0. Connectivity can be used in a **MACHPRIO** expression to favor nodes that do not have any down switch adapters or in a job's **REQUIREMENTS** to require only nodes with a certain connectivity.

For additional information about using this variable, see the **MACHPRIO** keyword description.

ConsumableCpus

The number of **ConsumableCpus** currently available on the machine, if **ConsumableCpus** is defined in the configuration file keyword, **SCHEDULE_BY_RESOURCES**. If it is not defined in **SCHEDULE_BY_RESOURCES**, then it is equivalent to **Cpus**.

For additional information about using this variable, see the **MACHPRIO** keyword description.

ConsumableMemory

The amount of **ConsumableMemory**, in megabytes, currently available on the machine, if **ConsumableMemory** is defined in the configuration file keyword,

SCHEDULE_BY_RESOURCES. If it is not defined in **SCHEDULE_BY_RESOURCES**, then it is equivalent to **Memory**.

For additional information about using this variable, see the **MACHPRIO** keyword description.

ConsumableVirtualMemory

The amount of **ConsumableVirtualMemory**, in megabytes, currently available on the machine, if **ConsumableVirtualMemory** is defined in the configuration file keyword, **SCHEDULE_BY_RESOURCES**. If it is not defined in **SCHEDULE_BY_RESOURCES**, then it is equivalent to **VirtualMemory**.

For additional information about using this variable, see the **MACHPRIO** keyword description.

Cpus

The number of processors of the machine, reported by the `startd` daemon.

For additional information about using this variable, see the **MACHPRIO** keyword description.

CurrentTime

The **UNIX date**; the current system time, in seconds, since January 1, 1970, as returned by the `time()` function.

CustomMetric

Sets a relative priority number for one or more machines, based on the value of the **CUSTOM_METRIC** keyword.

For additional information about using this variable, see the **MACHPRIO** keyword description.

Disk

The free disk space in kilobytes on the file system where the executables for the LoadLeveler jobs assigned to this machine are stored. This refers to the file system that is defined by the `execute` keyword.

For additional information about using this variable, see the **MACHPRIO** keyword description.

domain or domainname

Dynamically indicates the official name of the domain of the current host machine where the program is running. Whenever a machine name can be specified or one is assumed, a domain name is assigned if none is present.

EnteredCurrentState

The value of **CurrentTime** when the current state (START, SUSPEND, etc) was entered.

FreeRealMemory

The amount of free real memory, in megabytes, on the machine. This value should track very closely with the "fre" value of the `vmstat` command and the "free" value of the `svmon -G` command (units are 4K blocks).

For additional information about using this variable, see the **MACHPRIO** keyword description.

GroupQueuedJobs

The number of job steps associated with a LoadLeveler group which are either running or queued. (That is, job steps which are in one of these states: Checkpointing, Preempted, Preempt Pending, Resume Pending, Running, Starting, Pending, or Idle.)

Configuration file reference

For additional information about using this variable, see the **SYSPRIO** keyword description.

GroupRunningJobs

The number of job steps for the LoadLeveler group which are in one of these states: Checkpointing, Preempted, Preempt Pending, Resume Pending, Running, Starting, or Pending.

For additional information about using this variable, see the **SYSPRIO** keyword description.

GroupSysprio

The priority for the group of the job step, defined in the group stanza in the administration file.

Default: 0

For additional information about using this variable, see the **SYSPRIO** keyword description.

GroupTotalJobs

The total number of job steps associated with this LoadLeveler group. Total job steps are all job steps reported by the **llq** command.

For additional information about using this variable, see the **SYSPRIO** keyword description.

GroupTotalShares

The total number of shares allocated to a group as specified by the **fair_shares** keyword in the group stanza.

For additional information about using this variable, see the **SYSPRIO** keyword description.

GroupUsedBgShares

The number of Blue Gene shares already used by a group or jobs owned by the group.

For additional information about using this variable, see the **SYSPRIO** keyword description.

GroupUsedShares

The number of shares already used by a group or jobs of the LoadLeveler group.

For additional information about using this variable, see the **SYSPRIO** keyword description.

host or **hostname**

Dynamically indicates the standard host name as returned by `gethostname()` for the machine where the program is running. **host** and **hostname** are equivalent, and contain the name of the machine without the domain name appended to it. If administrators need to specify the domain name in the configuration file, they may use **domain** or **domainname** along with **host** or **hostname**. For example:

```
$(host).$(domain)
```

JobIsBlueGene

Indicates whether the job whose priority is being calculated using the **SYSPRIO** keyword is a Blue Gene job.

For additional information about using this variable, see the **SYSPRIO** keyword description.

KeyboardIdle

The number of seconds since the keyboard or mouse was last used. It also includes any telnet or interactive activity from any remote machine.

LoadAvg

The Berkely one-minute load average, a measure of the CPU load on the system. The load average is the average of the number of processes ready to run or waiting for disk I/O to complete. The load average does not map to CPU time.

For additional information about using this variable, see the **MACHPRIO** keyword description.

Machine

Indicates the name of the current machine. Note that **Machine** is a special case of a LoadLeveler variable called a machine variable. See the description of the **Arch** variable for more information.

MasterMachPriority

A value that is equal to 1 for nodes which are master nodes (those with **master_node_exclusive = true**); this value is equal to 0 for nodes which are not master nodes. Assigning a high priority to master nodes may help job scheduling performance for parallel jobs which require master node features.

For additional information about using this variable, see the **MACHPRIO** keyword description.

Memory

The size of real memory, in megabytes, of the machine, reported by the startd daemon.

For additional information about using this variable, see the **MACHPRIO** keyword description.

OpSys

Indicates the operating system on the host where the program is running. This value is automatically determined and need not be defined in the configuration file. Note that **OpSys** is a special case of a LoadLeveler variable called a machine variable. See the description of the **Arch** variable for more information.

PagesFreed

The number of pages freed per second by the page replacement algorithm of the virtual memory manager.

For additional information about using this variable, see the **MACHPRIO** keyword description.

PagesScanned

The number of pages scanned per second by the page replacement algorithm of the virtual memory manager.

For additional information about using this variable, see the **MACHPRIO** keyword description.

QDate

The difference in seconds between the UNIX date when the job step enters the queue and the UNIX date when the negotiator daemon starts up.

For additional information about using this variable, see the **SYSPRIO** keyword description.

Configuration file reference

Speed

The relative speed of the machine, defined in a machine stanza in the administration file.

Default: 1

For additional information about using this variable, see the **MACHPRIO** keyword description.

State

The state of the startd daemon.

tilde

The home directory for the LoadLeveler user ID.

UserPrio

The user defined priority of the job step, specified in the job command file with the **user_priority** keyword. The priority ranges from 0 to 100, with higher numbers corresponding to greater priority.

Default: 50

For additional information about using this variable, see the **SYSPRIO** keyword description.

UserQueuedJobs

The number of job steps either running or queued for the user. (That is, job steps that are in one of these states: Checkpointing, Preempted, Preempt Pending, Resume Pending, Running, Starting, Pending, or Idle.)

For additional information about using this variable, see the **SYSPRIO** keyword description.

UserRunningJobs

The number of job step steps for the user which are in one of these states: Checkpointing, Preempted, Preempt Pending, Resume Pending, Running, Starting, or Pending.

For additional information about using this variable, see the **SYSPRIO** keyword description.

UserSysprio

The priority of the user who submitted the job step, defined in the user stanza in the administration file.

Default: 0

For additional information about using this variable, see the **SYSPRIO** keyword description.

UserTotalJobs

The total number of job steps associated with this user. Total job steps are all job steps reported by the **llq** command.

For additional information about using this variable, see the **SYSPRIO** keyword description.

UserTotalShares

The total number of shares allocated to a user as specified by the **fair_shares** keyword in the user stanza.

For additional information about using this variable, see the **SYSPRIO** keyword description.

UserUsedBgShares

The number of Blue Gene shares already used by a user or jobs owned by the user.

For additional information about using this variable, see the **SYSPRIO** keyword description.

UserUsedShares

The number of shares already used by a user or jobs owned by the user.

For additional information about using this variable, see the **SYSPRIO** keyword description.

VirtualMemory

The size of available swap space (free paging space) on the machine, in kilobytes, reported by the startd daemon.

For additional information about using this variable, see the **MACHPRIO** keyword description.

Variables to use for setting dates

You can use the following date variables:

tm_mday

The number of the day of the month (1-31).

tm_mon

Number of months since January (0-11).

tm_wday

Number of days since Sunday (0-6).

tm_yday

Number of days since January 1 (0-365).

tm_year

The number of years since 1900 (0-9999). For example:

```
tm_year == 100
```

Denotes the year 2000.

tm4_year

The integer representation of the current year. For example:

```
tm4_year == 2010
```

Denotes the year 2010.

Variables to use for setting times

You can use the following time variables in the **START**, **SUSPEND**, **CONTINUE**, **VACATE**, and **KILL** expressions. If you use these variables in the **START** expression and you are operating across multiple time zones, unexpected results may occur. This is because the negotiator daemon evaluates the **START** expressions and this evaluation is done in the time zone in which the negotiator resides. Your executing machine also evaluates the **START** expression and if your executing machine is in a different time zone, the results you may receive may be inconsistent. To prevent this inconsistency from occurring, ensure that both your negotiator daemon and your executing machine are in the same time zone.

tm_hour

The number of hours since midnight (0-23).

Configuration file reference

tm_isdst

Daylight Savings Time flag: positive when in effect, zero when not in effect, negative when information is unavailable. For example, to start jobs between 5 PM and 8 AM during the month of October, factoring in an adjustment for Daylight Savings Time, you can issue:

```
START: (tm_mon == 9) && (tm_hour < 8) && (tm_hour > 17) && (tm_isdst = 1)
```

tm_min

Number of minutes after the hour (0-59).

tm_sec

Number of seconds after the minute (0-59).

Chapter 13. Administration file reference

The administration file lists and defines the machines in the LoadLeveler cluster, as well as the characteristics of classes, users, groups, and clusters. LoadLeveler does not prevent you from having multiple copies of administration files, but having only one administration file prevents confusion and avoids potential problems that might arise from having multiple files to update. To use only one administration file that is available to all machines in a cluster, you must place the file in a shared file system.

Table 72 lists the administration file subtasks:

Table 72. Administration file subtasks

Subtask	Associated information (see . . .)
To find out what administrator tasks you can accomplish by using the administration file	Part 2, "Configuring and managing the TWS LoadLeveler environment," on page 37
To learn how to correctly specify the contents of an administration file	<ul style="list-style-type: none">• "Administration file structure and syntax"• "Administration file keyword descriptions" on page 306

Administration file structure and syntax

The administration file is called **LoadL_admin** and it lists and defines the *machine*, *user*, *class*, *group*, and *adapter* stanzas.

Machine stanza

Defines the roles that the machines in the LoadLeveler cluster play. See "Defining machines" on page 78 for more information.

User stanza

Defines LoadLeveler users and their characteristics. See "Defining users" on page 91 for more information.

Class stanza

Defines the characteristics of the job classes. To define characteristics that apply to specific users, user substanzas can be added within a class stanza. See "Defining classes" on page 83 and "Defining user substanzas in class stanzas" on page 88 for more information.

Group stanza

Defines the characteristics of a collection of users that form a LoadLeveler group. See "Defining groups" on page 92 for more information.

Adapter stanza

Defines the network adapters available on the machines in the LoadLeveler cluster. See "Defining adapters" on page 80 for more information.

Cluster stanza

Defines the characteristics of a LoadLeveler cluster for use in a Multicluster environment. See "Defining clusters" on page 93 for more information.

Stanzas have the following general format:

Administration file reference

```
label: type = type_of_stanza
keyword1 = value1
keyword2 = value2
...
```

Figure 37. Format of administration file stanzas

Substanzas have the following general format:

```
label: {
  type = type_of_stanza
  keyword1 = value1
  keyword2 = value2
  ...
  stanza_label: {
    type = type_of_substanza
    keyword3 = value3
  }
}
```

Figure 38. Format of administration file substanzas

Keywords are *not* case sensitive. This means you can enter them in lower case, upper case, or mixed case.

The following is a simple example of an administration file illustrating several stanzas:

```
machine_a: type = machine
           central_manager = true      # defines this machine as the central manager
           adapter_stanzas = adapter_a # identifies an adapter stanza

class_a: type = class
         priority = 50 # priority of this class

user_a: type = user
       priority = 50 # priority of this user

group_a: type = group
        priority = 50 # priority of this group

adapter_a: type = adapter
          adapter_name = en0 #defines an adapter
```

Figure 39. Sample administration file stanzas

The following is a simple example of an administration file illustrating a class stanza that contains user substanzas:

```

default:
    type = machine
    central_manager = false
    schedd_host = true

default:
    type = class
    wall_clock-limit = 60:00, 30:00

parallel: {
    type = class

    # Allow at most 50 running jobs for class parallel
    maxjobs = 50

    # Allow at most 10 running jobs for any single
    # user of class parallel
    default: {
        type = user
        maxjobs = 10
    }

    # Allow user dept_head to run as many as 20 jobs
    # of class parallel
    dept_head: {
        type = user
        maxjobs = 20
    }
}

dept_head:
    type = user
    maxjobs = 30

```

Figure 40. Sample administration file stanza with user substanzas

Stanza characteristics

The characteristics of a stanza are:

- Every stanza has a label associated with it. The label specifies the name you give to the stanza.
- Every stanza has a **type** field that specifies it as a user, class, machine, group, or adapter stanza.
- New line characters are ignored. This means that separate parts of a stanza can be included on the same line. However, it is not recommended to have parts of a stanza cross line boundaries.
- White space is ignored, other than to delimit keyword identifiers. This eliminates confusion between tabs and spaces at the beginning of lines.
- A crosshatch sign (#) identifies a comment and can appear anywhere on the line. All characters following this sign on that line are ignored.
- Multiple stanzas of the same label are allowed, but only the first label is used.
- Default stanzas specify the default values for any keywords which are not otherwise specified. Each stanza type can have an associated default stanza. A default stanza must appear in the administration file ahead of any specific stanza entries of the same type. For example, a default class stanza must appear ahead of any specific class stanzas you enter.
- Stanzas can be nested within other stanzas (these are known as *substanzas*). See “Defining user substanzas in class stanzas” on page 88 for more information.

Administration file reference

- The use of opening and closing braces ({ and }) to mark the beginning and end of a stanza is optional for stanzas that *do not* contain substanzas. A stanza that contains substanzas *must* be specified using braces as delimiting characters. Only user substanzas within class stanzas are supported. No types of stanzas other than class support substanzas and no types of stanzas other than user can be provided as substanzas within a class.
- If a syntax error is encountered, the remainder of the stanza is ignored and processing resumes with the next stanza.

Syntax for limit keywords

The syntax for setting a limit is:

```
limit_type = hardlimit,softlimit
```

For example:

```
core_limit = 120kb,100kb
```

To specify only a hard limit, you can enter, for example:

```
core_limit = 120kb
```

To specify only a soft limit, you can enter, for example:

```
core_limit = ,100kb
```

In a keyword statement, you cannot have any blanks between the numerical value (100 in the above example) and the units (kb). Also, you cannot have any blanks to the left or right of the comma when you define a limit in a job command file.

For limit keywords that refer to a data limit — such as **data_limit**, **core_limit**, **file_limit**, **stack_limit**, and **rss_limit** — the hard limit and the soft limit are expressed as:

```
integer[.fraction][units]
```

The allowable units for these limits are:

```
b bytes  
w words  
kb kilobytes (2**10 bytes)  
kw kilowords (2**12 bytes)  
mb megabytes (2**20 bytes)  
mw megawords (2**22 bytes)  
gb gigabytes (2**30 bytes)  
gw gigawords (2**32 bytes)  
tb terabytes (2**40 bytes)  
tw terawords (2**42 bytes)  
pb petabytes (2**50 bytes)  
pw petawords (2**52 bytes)  
eb exabytes (2**60 bytes)  
ew exawords (2**62 bytes)
```

If no units are specified for data limits, then bytes are assumed.

For limit keywords that refer to a time limit — such as **ckpt_time_limit**, **cpu_limit**, **job_cpu_limit**, and **wall_clock_limit** — the hard limit and the soft limit are expressed as:

```
[[hours:]minutes:]seconds[.fraction]
```

Fractions are rounded to seconds.

You can use the following character strings with all limit keywords except the **copy** keyword for **wall_clock_limit**, **job_cpu_limit**, and **ckpt_time_limit**:

rlim_infinity	Represents the largest positive number.
unlimited	Has same effect as rlim_infinity .
copy	Uses the limit currently active when the job is submitted.

64-bit support for administration file keywords

Administrators can assign 64-bit integer values to selected keywords in the administration file. System resource limits, with the exception of CPU limits, are treated by LoadLeveler daemons and commands as 64-bit limits.

Table 73 describes 64-bit support for specific administration file keywords.

Table 73. Notes on 64-bit support for administration file keywords

Keyword	Stanza	Notes
core_limit	Class	64-bit integer values can be assigned to these limits. Fractional specifications are allowed and will be converted to 64-bit integer values. Unit specifications are accepted and can be one of the following: b, w, kb, kw, mb, mw, gb, gw, tb, tw, pb, pw, eb, ew. Example: core_limit = 8gb,4.25gb
data_limit		
default_resources	Class	Consumable resources associated with the default_resources keyword can be assigned 64-bit integer values. Fractional specifications are not allowed. Unit specifications are valid only when specifying the values of the predefined ConsumableMemory and ConsumableVirtualMemory resources. Example: default_resources = ConsumableVirtualMemory(12 gb) db2_license(112)
file_limit	Class	See the notes for core_limit and data_limit .
resources	Machine	Consumable resources associated with the resources keyword can be assigned 64-bit integer values. Fractional specifications are not allowed. Unit specifications are valid only when specifying the values of the predefined ConsumableMemory and ConsumableVirtualMemory resources. Examples: resources = spice2g6(9123456789012) ConsumableMemory(10 gw) resources = ConsumableVirtualMemory(15 pb) db2_license(1234567890)
rss_limit	Class	See the notes for core_limit and data_limit . Example: rss_limit = 1.25eb,3.33pw
stack_limit		

64-bit limits on Linux systems

Applications managed by LoadLeveler for AIX can be 64-bit applications if the hardware architecture on which AIX is running is capable of supporting 64-bit processes. Resource limits, such as data limits and stack limits, can be 64-bit limits. When a value of *unlimited* is specified for a process limit (**cpu_limit** excepted) in the LoadLeveler administration file or job command file, the AIX version of LoadLeveler stores this value internally as INT64_MAX. Before starting the user job, **LoadL_starter** sets the appropriate limit to this value. This behavior is correct because, on AIX, RLIM64_INFINITY is the same as INT64_MAX (= 0x7FFFFFFFFFFFFFFFLL).

Administration file reference

On Linux systems, RLIM64_INFINITY is equal to UINT64_MAX (= 0xFFFFFFFFFFFFFFFFULL). To maintain compatibility with AIX, LoadLeveler for Linux also stores *unlimited* internally as INT64_MAX. However, **LoadL_starter** on Linux sets all process limits (**cpu_limit** excepted) that are in the range (INT64_MAX, UINT64_MAX) to UINT64_MAX before starting the jobs managed by LoadLeveler.

For historical reasons, LoadLeveler for AIX treats the hard and soft time limits, such as **cpu_limit**, **job_cpu_limit**, and **wall_clock_limit**, as 32-bit limits and *unlimited* means INT32_MAX. For consistency reasons, LoadLeveler for Linux assumes the same behavior.

Administration file keyword descriptions

account

Specifies a list of account numbers available to a user submitting jobs.

Syntax:

```
account =list
```

Where *list* is a blank-delimited list of account numbers that identifies the account numbers a user can use when submitting jobs.

Default: A null list.

adapter_name

Specifies the name the operating system uses to refer to an interface card installed on a node.

Syntax:

```
adapter_name = string
```

Where *string* is the name of a particular interface card installed on the node. Some examples are **en0** and **tk1**. This keyword defines the adapters a user can specify in a job command file using the **network** keyword.

adapter_stanzas

Specifies a list of adapter stanza names that define the adapters on a machine that can be requested.

Syntax:

```
adapter_stanzas = stanza_list
```

Where *stanza_list* is a blank-delimited list of one or more adapter stanza names which specify adapters available on this machine. To take advantage of dynamic adapter configuration you must *exclude* this keyword from the machine stanza. LoadLeveler will then dynamically obtain the adapter configuration for this machine from the RSCT. All adapter stanzas you define must be specified on this keyword. If the keyword is specified without defining any adapter stanza names no adapter will be configured for the machine.

adapter_type

Specifies the type of switch adapter to be used. This keyword is used for the High Performance Switch in a peer domain. The **llextrPD** command will not generate an **adapter_type** statement if no AdapterType is found in the cluster.

Syntax:

```
adapter_type = type
```

Where *type* is the designation for the type of switch adapter.

admin Specifies a list of administrators for a group or class.

Syntax:

`admin = list`

Where *list* is a blank-delimited list of administrators for either this class or this group, depending on whether this keyword appears in a class or group stanza, respectively. These administrators can hold, release, and cancel jobs in this class or this group.

alias Lists one or more alias names to associate with the machine name.

Syntax:

`alias = machine_name`

Where *machine_name* is a blank-delimited list of one or more machine names. Depending upon your network configurations, you can need to add **alias** keywords for machines that have multiple interfaces.

In general, if your cluster is configured with machine host names which match the host names corresponding to the IP address configured for the LAN adapters which LoadLeveler is expected to use, you will not have to specify the **alias** keyword. For example, if all of the machines in your cluster are configured like this sample machine, you should not have to specify the **alias** keyword.

Machine `porsche.kgn.ibm.com`

- The hostname command returns `porsche.kgn.ibm.com`.
- The Ethernet adapter address `129.40.8.20` resolves to hostname `porsche.kgn.ibm.com`.

However, if any machine in your cluster is configured like either of the following two sample machines, then you will have to specify the **alias** keyword for those machines:

1. Machine `yugo.kgn.ibm.com`

- The hostname command returns `yugo.kgn.ibm.com`.
- The Ethernet adapter address `129.40.8.21` resolves to hostname `chevy.kgn.ibm.com`.
- No adapter address resolves to `yugo`.

You need to code the machine stanza as:

```
chevy: type = machine
alias = yugo
```

2. Machine `rover.kgn.ibm.com`

- The hostname command returns `rover.kgn.ibm.com`.
- The FDDI adapter address `129.40.9.22` resolves to hostname `rover.kgn.ibm.com`.
- The Ethernet adapter address `129.40.8.22` resolves to hostname `bmw.kgn.ibm.com`.
- No route exists via the FDDI adapter to the clusters central manager machine.
- A route exists from this machine to the central manager via the Ethernet adapter.

You need to code the machine stanza as:

Administration file reference

```
bmw:  type = machine
      alias = rover
```

central_manager

Determines whether the machine is the LoadLeveler central manager.

Syntax:

```
central_manager = true | false | alt
```

Where:

- **true** designates this machine as the LoadLeveler central manager host, where the negotiator daemon runs. You must specify one and only one machine stanza identifying the central manager. For example:

```
machine_a: type = machine
          central_manager = true
```

- **false** specifies that this machine is not the central manager.
- **alt** specifies that this machine can serve as an alternate central manager in the event that the primary central manager is not functioning. For more information on recovering if the primary central manager is not operating, refer to “What happens if the central manager isn’t operating?” on page 658. Submit-only machines cannot have their machine stanzas set to this value.

If you are going to select machines to serve as alternate central managers, you should look at the following keywords in the configuration file:

- **CENTRAL_MANAGER_HEARTBEAT_INTERVAL**
- **CENTRAL_MANAGER_TIMEOUT**

For information on setting these keywords, see “Specifying alternate central managers” on page 44.

Default: false

ckpt_dir

Specifies the directory to be used for checkpoint files for jobs that did not specify this directory in the job command file.

Syntax:

```
ckpt_dir = directory
```

Where *directory* is the directory location to be used for checkpoint files that did not have a directory name specified in the job command file. If the value specified does not have a fully qualified directory path (including the beginning forward slash), the initial working directory will be inserted before the specified value.

The value specified by the **ckpt_dir** keyword is only used when the **ckpt_file** keyword in the job command file does not contain a full path name and the **ckpt_dir** keyword in the job command file is not specified. For more information on determining the checkpoint directory, see “Naming checkpoint files and directories” on page 134.

Default: Initial working directory

ckpt_time_limit

Specifies the hard limit, soft limit, or both limits for the elapsed time that checkpointing a job can take.

Syntax:

```
ckpt_time_limit = hardlimit,softlimit
```

Where *hardlimit,softlimit* defines the maximum time that checkpointing a job can take. When LoadLeveler detects that the softlimit has been exceeded, it attempts to end the checkpoint and allow the job to continue. If this is not possible, and the hard limit is exceeded, LoadLeveler will terminate the job. The start time of the checkpoint is defined as the time when the Startd daemon receives status from the starter that a checkpoint has started.

Default: Unlimited

Examples:

```
ckpt_time_limit = 30:45           #hardlimit - 30 minutes 45 seconds
ckpt_time_limit = 30:45,25:00    #hardlimit - 30 minutes 44 seconds
                                #softlimit  - 25 minutes
```

For additional information about limit keywords, see the following topics:

- “Syntax for limit keywords” on page 304
- “Using limit keywords” on page 83

class_comment

Text characterizing the class.

Syntax:

```
class_comment = "string"
```

Where *string* is text characterizing the class. This information appears when the user is building a job command file using the GUI and requests Choice information on the classes to which he or she is authorized to submit jobs. The comment string associated with this keyword cannot contain an equal sign (=) or a colon (:). The length of the string cannot exceed 1024 characters.

Default: No default value is set.

core_limit

Specifies the hard limit, soft limit, or both limits for the size of a core file a job can create.

Syntax:

```
core_limit = hardlimit,softlimit
```

Examples:

```
core_limit = unlimited
core_limit = 30mb
```

For additional information about limit keywords, see the following topics:

- “Syntax for limit keywords” on page 304
- “Using limit keywords” on page 83

cpu_limit

Specifies hard limit, soft limit, or both limits for the CPU time to be used by each individual process of a job step.

Syntax:

```
cpu_limit = hardlimit,softlimit
```

Administration file reference

For example, if you impose a **cpu_limit** of five hours and you have a job step composed of five processes, each process can consume five CPU hours; the entire job step can therefore consume 25 total hours of CPU.

Examples:

```
cpu_limit = 12:56:21      # hardlimit = 12 hours 56 minutes 21 seconds
cpu_limit = 56:00,50:00  # hardlimit = 56 minutes 0 seconds
                        # softlimit = 50 minutes 0 seconds
cpu_limit = 1:03         # hardlimit = 1 minute 3 seconds
cpu_limit = unlimited    # hardlimit = 2,147,483,647 seconds
                        # (X'7FFFFFFF')
cpu_limit = rlim_infinity # hardlimit = 2,147,483,647 seconds
                        # (X'7FFFFFFF')
cpu_limit = copy         # current CPU hardlimit value on the
                        # submitting machine.
```

For additional information about limit keywords, see the following topics:

- “Syntax for limit keywords” on page 304
- “Using limit keywords” on page 83

cpu_speed_scale

Determines whether CPU time is normalized according to machine speed.

Syntax:

```
cpu_speed_scale = true | false
```

Where **true** specifies that CPU time (which is used, for example, in setting limits, in accounting information, and reported by the **llq -x** command), is in normalized units for each machine. **false** specifies that CPU time is in native units for each machine. For an example of using this keyword to normalize accounting information, see “Example: Setting up job accounting files” on page 61.

Default: false

data_limit

Specifies hard limit, soft limit, or both for the data segment to be used by each process of the submitted job.

Syntax:

```
data_limit = hardlimit,softlimit
```

Examples:

```
data_limit = 125621      # hardlimit = 125621 bytes
data_limit = 5621kb     # hardlimit = 5621 kilobytes
data_limit = 2mb        # hardlimit = 2 megabytes
data_limit = 2.5mw      # hardlimit = 2.5 megawords
data_limit = unlimited  # hardlimit = 9,223,372,036,854,775,807 bytes
                        # (X'7FFFFFFFFFFFFFFF')
data_limit = rlim_infinity # hardlimit = 9,223,372,036,854,775,807 bytes
                        # (X'7FFFFFFFFFFFFFFF')
data_limit = copy       # copy data hardlimit value from
                        # submitting machine.
```

For additional information about limit keywords, see the following topics:

- “Syntax for limit keywords” on page 304
- “Using limit keywords” on page 83

default_class

Specifies a class name that is the default value assigned to jobs submitted by users for which no class statement appears.

Syntax:

```
default_class = list
```

Where *list* is a blank-delimited list of class names used for jobs which do not include a **class** statement in the job command file. If you specify only one default class name, this class is assigned to the job. If you specify a list of default class names, LoadLeveler searches the list to find a class which satisfies the resource limit requirements. If no class satisfies these requirements, LoadLeveler rejects the job.

Suppose a job requests a CPU limit of 10 minutes. Also, suppose the default class list is `default_class = short long`, where `short` is a class for jobs up to five minutes in length and `long` is a class for jobs up to one hour in length. LoadLeveler will select the `long` class for this job because the `short` class does not have sufficient resources.

Default: If no **default_class** is specified in the user stanza, or if there is no user stanza at all, then jobs submitted without a **class** statement are assigned to the **default_class** that appears in the default user stanza. If you do not define a **default_class**, jobs are assigned to the class called **No_Class**.

default_group

Specifies the default group name to which the user belongs.

Syntax:

```
default_group = group_name
```

Where *group_name* is the default group assigned to jobs submitted by the user.

If you specify **default_group = Unix_Group**, LoadLeveler sets the user's LoadLeveler group to the user's current UNIX group.

Default: If a **default_group** statement does not appear in the user stanza, or if there is no user stanza at all, then jobs submitted by the user without a **group** statement are assigned to the **default_group** that appears in the default user stanza. If you do not define a **default_group**, jobs are assigned to the group called **No_Group**.

default_interactive_class

Specifies a class to which interactive jobs are assigned for jobs submitted by users who do not specify a class using the `LOADL_INTERACTIVE_CLASS` variable. You can specify only one default interactive class name.

Syntax:

```
default_interactive_class = class_name
```

Where *class_name* is the class to which an interactive job submitted by this user is assigned if the user does not specify a class using the `LOADL_INTERACTIVE_CLASS` environment variable.

Default: If you do not set a **default_interactive_class** value in the user stanza, or if there is no user stanza at all, then interactive jobs submitted without a **class** statement are assigned to the **default_interactive_class** that

Administration file reference

appears in the default user stanza. If you do not define a **default_interactive_class**, interactive jobs are assigned to the class called **No_Class**.

See “Examples: User stanzas” on page 91 for more information on how LoadLeveler assigns a default interactive class to jobs.

default_resources

Specifies the default amount of resources consumed by a task of a job step, provided that no **resources** keyword is coded for the step in the job command file. If a resources keyword is coded for a job step, then it overrides any **default_resources** associated with the associated job class.

Syntax:

```
default_resources = name(count) name(count)...name(count)
```

The administrator defines the name and count values for **default_resources**. In addition, *name(count)* could be **ConsumableCpus**(*count*), **ConsumableMemory**(*count units*), or **ConsumableVirtualMemory**(*count units*).

ConsumableMemory and **ConsumableVirtualMemory** are the only two consumable resources that can be specified with both a count and units. The count for each specified resource must be an integer greater than or equal to zero. The allowable units are those normally used with LoadLeveler data limits:

```
b bytes
w words
kb kilobytes (2**10 bytes)
kw kilowords (2**12 bytes)
mb megabytes (2**20 bytes)
mw megawords (2**22 bytes)
gb gigabytes (2**30 bytes)
gw gigawords (2**32 bytes)
tb terabytes (2**40 bytes)
tw terawords (2**42 bytes)
pb petabytes (2**50 bytes)
pw petawords (2**52 bytes)
eb exabytes (2**60 bytes)
ew exawords (2**62 bytes)
```

The **ConsumableMemory** and **ConsumableVirtualMemory** values are stored in MB (megabytes) and rounded up. Therefore, the smallest amount of **ConsumableMemory** or **ConsumableVirtualMemory** which you can request is one megabyte. If no units are specified, then megabytes are assumed. Resources defined here that are not in the **SCHEDULE_BY_RESOURCES** list in the global configuration file will not effect the scheduling of the job.

default_wall_clock_limit

Sets a default value for jobs not specifying a wall clock limit in the job command file. The **wall_clock_limit** keyword serves only as the maximum value allowed for the class. The **default_wall_clock_limit** value can be overridden by a job using the **wall_clock_limit** job command file keyword, but that limit cannot exceed the **wall_clock_limit** configured in the class stanza.

Note: If **default_wall_clock_limit** is not specified, it will be assigned the value of **wall_clock_limit** for the same class.

Syntax:

```
default_wall_clock_limit = hardlimit,softlimit
```

An example is:

```
default_wall_clock_limit = 5:00,4:30
```

For more information about the values and units you can use with this keyword, and how limits are enforced, see “Using limit keywords” on page 83 and “Syntax for limit keywords” on page 304.

device_driver_name

Specifies the device driver interface needed for user space function.

Syntax:

```
device_driver_name = name
```

Where *name* specifies the device driver interface. A **device_driver_name** will be present for all adapter stanzas whose name begins with **sn**. This keyword is for peer domain switch adapters.

env_copy

Specifies a default value for the job command file **env_copy** keyword for the class, group or user stanza containing the keyword.

Syntax:

```
env_copy = all | master
```

Table 74 states the value that LoadLeveler uses depending on the combination of values set in the user, group, or class stanzas.

Table 74. Summary of possible values set for the **env_copy** keyword in the administration file

env_copy keyword setting in applicable stanzas in the administration file	Resulting LoadLeveler default behavior for copying the job environment
All stanzas that set the env_copy keyword specify env_copy = master	master becomes the default value for the job command file env_copy keyword.
One or more stanzas explicitly set env_copy = all	all becomes the default value for the job command file env_copy keyword.
The env_copy keyword is not specified in any stanza	

Default value: No default value is set.

For more information, see:

- The job command file **env_copy** keyword description.
- “Steps for reducing job launch overhead for parallel jobs” on page 96.

exclude_classes

exclude_classes can be specified within a cluster stanza.

Specifies a blank-delimited list of one or more job classes that will not accept remote jobs within the cluster.

Syntax:

```
exclude_classes = class_name[(cluster_name)] ...
```

Administration file reference

Where *class_name* specifies a class to be excluded and *cluster_name* can be used to specify that remote jobs from *cluster_name* submitted under *class_name* will be excluded but any other jobs submitted under *class_name* from other clusters will be allowed.

Do not specify a list of **exclude_classes** and **include_classes**. Only one of these keywords can be used within any cluster stanza. **exclude_classes** takes precedence over **include_classes** if both are specified.

Default: The default is that no classes are excluded.

exclude_groups

exclude_groups can be specified within a class stanza and a cluster stanza.

Class stanza:

When used within a class stanza, **exclude_groups** specifies a list of group names identifying those who cannot submit jobs of a particular class.

Syntax:

```
exclude_groups = list
```

Where *list* is a blank-delimited list of groups who are *not* allowed to submit jobs of *class name*.

This list can contain individual user names. To allow a list of users to be included with the list of group names, add a plus sign (+) to each user name that you add to the list. LoadLeveler treats these names as implicit groups.

For example, to add user **mike** to a list of group names, specify:

```
exclude_groups = prod +mike
```

If the string **+mike** is also the actual name of a group stanza, LoadLeveler treats this name as a group, not an implicit group. In this case, LoadLeveler will not prevent user **mike** from submitting jobs to this class unless the user is a member of the **prod** or **+mike** group.

If this keyword is specified, this list limits groups and users of that class to those on the list.

Do not specify both a list of included groups and a list of excluded groups. Only one of these may be used for any class stanza. **exclude_groups** takes precedence over **include_groups** if both are specified.

Default: The default is that no groups are excluded.

Cluster stanza:

When used within a cluster stanza, **exclude_groups** specifies a blank-delimited list of one or more groups that will not accept remote jobs within the cluster.

Syntax:

```
exclude_groups = group_name[(cluster_name)] ...
```

Where *group_name* specifies a group that is not allowed to submit remote jobs and *cluster_name* can be used to specify that remote jobs from *cluster_name* submitted under *group_name* will be excluded but any other jobs submitted under *group_name* from other clusters will be allowed.

Do not specify a list of **exclude_groups** and **include_groups**. Only one of these may be used within any cluster stanza. **exclude_groups** takes precedence over **include_groups** if both are specified.

Default: The default is that no groups are excluded.

exclude_users

exclude_users may be specified within a class, group, and cluster stanza.

Class or group stanza:

When used within a class or group stanza **exclude_users** specifies a list of user names identifying those who cannot submit jobs of a particular class or who are not members of the group.

Syntax:

```
exclude_users = list
```

The definition of this keyword varies slightly, depending on the type of administration file stanza in which the keyword appears:

- In a class stanza: *list* is a blank-delimited list of users who are *not* permitted to submit jobs of *class_name*.
- In a group stanza: *list* is a blank-delimited list of users who do not belong to the group.

Do not specify both a list of included users and a list of excluded users. Only one of these may be used for any class or group. **exclude_users** takes precedence over **include_users** if both are specified. In a class stanza, **exclude_users** also takes precedence over any user substanzas.

Default: The default is that no users are excluded.

Cluster stanza:

When used within a cluster stanza, **exclude_users** specifies a blank-delimited list of one or more users who cannot submit jobs to the cluster.

Syntax:

```
exclude_users = user_name[(cluster_name)] ...
```

Where *user_name* specifies a user that is not allowed to submit remote jobs and *cluster_name* can be used to specify that remote jobs from *cluster_name* submitted under the *user_name* will be excluded but any other jobs submitted under that *user_name* from other clusters will be allowed.

Do not specify a list of **exclude_users** and **include_users**. Only one of these may be used within any cluster stanza. **exclude_users** takes precedence over **include_users** if both are specified.

Default: The default is that no users are excluded.

Administration file reference

fair_shares

Specifies the number of shares allocated to jobs of this user or group for fair share scheduling. If the user or group stanza does not specify **fair_shares**, or if there is no user or group stanza at all, the value in the default user or group stanza is used (which defaults to zero if not explicitly specified). The user or group has this number of shares of the cluster CPU resources as well as this number of shares of the Blue Gene resources (if Blue Gene resources are also available in the cluster).

Syntax:

`fair_shares = number`

For additional information about the fair share scheduling keyword, see “Using fair share scheduling” on page 148.

- “Syntax for limit keywords” on page 304
- “Using limit keywords” on page 83

file_limit

Specifies the hard limit, soft limit, or both limits for the size of a file that a job can create.

Syntax:

`file_limit = hardlimit,softlimit`

For additional information about limit keywords, see the following topics:

- “Syntax for limit keywords” on page 304
- “Using limit keywords” on page 83

inbound_hosts

Specifies a blank-delimited list of hostnames that define the machines configured for inbound connections from other clusters.

Syntax:

`inbound_hosts = hostname[(cluster_name)] ...`

Where *hostname* specifies a machine configured for inbound connections from other clusters and *cluster_name* can be used to specify a specific cluster if the host is not connected to all clusters in the multicluster. These hostnames must be fully qualified with domain names if the machines exist in a different domain. This keyword is required in a multicluster environment.

Note: The same machine can be defined as both an **inbound_host** and an **outbound_host**.

inbound_schedd_port

Specifies the port number to use to connect to the Schedd for inbound transactions to this cluster.

Syntax:

`inbound_schedd_port = port_number`

Where *port_number* is a positive integer which specifies the port number used to connect to the Schedd for inbound transactions to this cluster.

Default: The default port is 9605.

include_classes

include_classes can be specified within a cluster stanza.

Specifies a blank-delimited list of one or more job classes that will accept remote jobs within the cluster.

Syntax:

```
include_classes = class_name[(cluster_name)] ...
```

Where *class_name* specifies a class to be included and *cluster_name* can be used to specify that remote jobs from *cluster_name* will be included but any other jobs submitted under *class_name* from other clusters will not be allowed.

Do not specify a list of **exclude_classes** and **include_classes**. Only one of these can be used within any cluster stanza. **exclude_classes** takes precedence over **include_classes** if both are specified.

Default: The default is that all classes are included.

include_groups

include_groups can be specified within a class stanza and a cluster stanza.

Class stanza:

When used within a class stanza, **include_groups** specifies a list of group names identifying those who can submit jobs of a particular class.

Syntax:

```
include_groups = list
```

Where *list* is a blank-delimited list of groups who are allowed to submit jobs of *class name*.

This list can contain individual user names. To allow a list of users to be included with the list of group names, add a plus sign (+) to each user name that you add to the list. LoadLeveler treats these names as implicit groups.

For example, to add user **mike** to a list of group names, specify:

```
include_groups = prod +mike
```

If the string **+mike** is also the actual name of a group stanza, LoadLeveler treats this name as a group, not an implicit group. In this case, LoadLeveler will not allow user **mike** to submit jobs to this class unless the user is a member of the **prod** or **+mike** group.

If this keyword is specified, this list limits groups and users of that class to those on the list.

Do not specify both a list of included groups and a list of excluded groups. Only one of these may be used for any class stanza. **exclude_groups** takes precedence over **include_groups** if both are specified.

Default: The default is that all groups are included.

Cluster stanza:

Administration file reference

When used within a cluster stanza, **include_groups** specifies a blank-delimited list of one or more groups that will accept remote jobs within the cluster.

Syntax:

```
include_groups = group_name[(cluster_name)] ...
```

Where *group_name* specifies a group that is allowed to submit remote jobs and *cluster_name* can be used to specify that remote jobs from *cluster_name* submitted under *group_name* will be included but any other jobs submitted under *group_name* from other clusters will not be allowed.

Do not specify a list of **exclude_groups** and **include_groups**. Only one of these may be used within any cluster stanza. **exclude_groups** takes precedence over **include_groups** if both are specified.

Default: The default is that all groups are included.

include_users

include_users may be specified within a class, group, and cluster stanza.

Class or group stanza:

When used within a class or group stanza **include_users** specifies a list of user names identifying those who can submit jobs of a particular class or who are members of the group.

Syntax:

```
include_users = list
```

The definition of this keyword varies slightly, depending on the type of administration file stanza in which the keyword appears:

- In a class stanza: *list* is a blank-delimited list of users who are permitted to submit jobs of *class_name*.
- In a group stanza: *list* is a blank-delimited list of users who belong to the group.

Do not specify both a list of included users and a list of excluded users. Only one of these may be used for any class or group. **exclude_users** takes precedence over **include_users** if both are specified. In a class stanza, users in user substanzas are also permitted to submit jobs of *class_name*, even if those users are not in the **include_users** list.

Default: The default is that all users are included.

Cluster stanza:

When used within a cluster stanza, **include_users** specifies a blank-delimited list of one or more users who can submit jobs to the cluster.

Syntax:

```
include_users = user_name[(cluster_name)] ...
```

Where *user_name* specifies a user that is allowed to submit remote jobs and *cluster_name* can be used to specify that remote jobs from *cluster_name*

submitted under the *user_name* will be included but any other jobs submitted under that *user_name* from other clusters will not be allowed.

Do not specify a list of **exclude_users** and **include_users**. Only one of these may be used within any cluster stanza. **exclude_users** takes precedence over **include_users** if both are specified.

Default: The default is that all users are included.

interface_address

Specifies the IP address by which the adapter is known to other nodes in the network.

Syntax:

```
interface_address = string
```

Where *string* is the IP address by which the adapter is known to other nodes in the network. For example: 7.14.21.28. This keyword is required.

interface_name

Specifies the name by which the adapter is known to other nodes in the network.

Syntax:

```
interface_name = string
```

Where *string* is the name by which the adapter is known by other nodes in the network.

job_cpu_limit

Specifies the hard limit, soft limit, or both limits for the total amount of CPU time that all tasks of an individual job step can use per machine.

Syntax:

```
job_cpu_limit = hardlimit,softlimit
```

Example:

```
job_cpu_limit = 10000
```

For more information on this keyword, see:

- JOB_LIMIT_POLICY keyword
- For additional information about limit keywords, see the following topics:
 - Syntax for limit keywords
 - Using limit keywords

local Specifies the scope of the cluster definition.

Syntax:

```
local = true| false
```

This keyword is required in the local cluster's administration file in a multicluster environment.

Default: false

logical_id

Specifies the logical ID that uniquely identifies the adapter on its network.

Syntax:

Administration file reference

```
logical_id = id
```

This keyword is for peer domain switch adapters.

machine_mode

Specifies the type of jobs this machine can run.

Syntax:

```
machine_mode = batch | interactive | general
```

Where:

- batch** Specifies this machine can run only batch jobs.
- interactive** Specifies this machine can run only interactive jobs. Only POE is currently enabled to run interactively.
- general** Specifies this machine can run both batch jobs and interactive jobs.

Default: general

master_node_exclusive

Specifies whether or not this machine is used only as a master node.

Syntax:

```
master_node_exclusive = true| false
```

Where **true** specifies that the machine accepts only jobs (serial or parallel) submitted to classes that have **master_node_requirement** set to **true**. If the job type is parallel, only the master task is run on a machine with **master_node_exclusive** set to **true**.

Default: false

master_node_requirement

Specifies whether or not parallel jobs in this class require the master node feature.

Syntax:

```
master_node_requirement = true|false
```

Where **true** specifies that parallel jobs do require the master node feature. For these jobs, LoadLeveler allocates the first node (called the “master”) on a machine having the **master_node_exclusive = true** setting in its machine stanza. If most or all of your parallel jobs require this feature, you should consider placing the statement **master_node_requirement = true** in your default class stanza. Then, for classes that do not require this feature, you can use the statement **master_node_requirement = false** in their class stanzas to override the default setting. One machine per class should have the **true** setting; if more than one machine has this setting, normal scheduling selection is performed.

Default: false

max_jobs_scheduled

Specifies the maximum number of job steps that this machine can run.

Syntax:

```
max_jobs_scheduled = number
```

Where *number* is the maximum number of jobs submitted from this scheduling (schedd) machine that can run (or start running) in the LoadLeveler cluster at one time. If *number* of jobs are already running, no other jobs submitted from this machine will run, even if resources are available in the LoadLeveler cluster. When one of the running jobs completes, any waiting jobs then become eligible to be run.

Default: The default is -1, which means there is no maximum.

max_node

Specifies the maximum number of nodes that can be requested for a particular class or by a particular user or group for a parallel job.

Syntax:

`max_node = number`

Where *number* specifies the maximum number of nodes for a parallel job in a job command file using the **node** keyword. The **max_node** keyword will not affect the use of the **min_processors** and **max_processors** keywords in the job command file.

Default: The default is -1, which means there is no limit.

max_processors

Specifies the maximum number of processors that can be requested for a particular class or by a particular user or group for a parallel job.

Syntax:

`max_processors = number`

Where *number* specifies the maximum number of processors for a parallel job in a job command file using the **min_processors** and **max_processors** keywords.

Default: The default is -1 which means that there is no limit.

max_protocol_instances

Specifies the maximum number of instances on the network statement.

Syntax:

`max_protocol_instances = number`

Where *number* specifies the maximum value allowed on the instances keyword on the network statement for jobs submitted on this class.

Default: The default is 2.

max_reservation_duration

Specifies the maximum time, in minutes, that advance reservations made for this user or group can last.

Syntax:

`max_reservation_duration = number of minutes`

When the duration is defined in both the user and group stanza for a specific user, LoadLeveler uses the more restrictive of the two values to determine the maximum duration.

Administration file reference

Default: The default is -1, which means that no limit is placed on the duration of the reservation.

For more information, see “Steps for configuring reservations in a LoadLeveler cluster” on page 121.

max_reservations

Specifies the maximum number of advance reservations that this user or group can make.

Syntax:

`max_reservations = number of reservations`

This number includes all reservations except those in COMPLETE or CANCEL state.

Table 75 summarizes the resulting behavior for various sample combinations of **max_reservations** settings in user and group stanzas.

Table 75. Sample user and group settings for the `max_reservations` keyword

When the user stanza value is:	And the group stanza value is:	Then the user can create this number of reservations in this group:
Not defined	Not defined	0 (zero)
2	Not defined	2 (with any group as the owning group)
Not defined	1	1
3	1	1 (the user can create more reservations in other groups)
1	2	1
0	2	0
1	0	0 (the user can create one reservation in another group)

Default: Undefined, which means that no reservations will be authorized or disallowed. LoadLeveler considers this keyword undefined if negative values are set for it.

max_top_dogs

Specifies the maximum total number of top dogs that the central manager daemon will allocate per class.

Syntax:

`max_top_dogs = number`

where *number* is any positive integer.

Default: The default value for this keyword is 1 for each class, unless a default is specified in the default class stanza

max_total_tasks

Specifies the maximum number of tasks that the BACKFILL scheduler allows a user, group, or class to run at any given time.

Syntax:

`max_total_tasks = number`

where *number* is -1, 0, or any positive integer.

Note: This keyword can be specified in a user stanza within a class when the BACKFILL scheduler is in use. This limits the number of tasks that user can run in that class at any given time.

Default: The default value for this keyword is -1, which allows an unlimited number of tasks.

maxidle

Specifies the maximum number of idle job steps this user or group can have simultaneously.

Syntax:

`maxidle = number`

Where *number* is the maximum number of idle jobs either this user or this group can have in queue, depending on whether this keyword appears in a user or group stanza. That is, *number* is the maximum number of jobs which the negotiator will consider for dispatch for the user or group. Jobs above this maximum are placed in the NotQueued state. This action prevents one of the following situations:

- Individual users from dominating the number of jobs that are either running or are being considered to run.
- Groups from flooding the job queue.

Notes:

1. This keyword can be specified in a user stanza within a class.
2. For the purposes of enforcing the number of idle job steps this user or group can have in queue, a job step is considered idle even if `llq` reports the state as Pending or Starting.

Default: If the user or group stanza does not specify **maxidle** or if there is no user or group stanza at all, the maximum number of jobs that can be simultaneously in queue for the user or group is defined in the default stanza. If no value is found, or the limit found is -1, then no limit is placed on the number of jobs that can be simultaneously idle for the user or group.

For more information, see “Controlling the mix of idle and running jobs” on page 668.

maxjobs

Specifies the maximum number of job steps this user, class, or group can have running simultaneously.

Syntax:

`maxjobs = number`

Note: This keyword can be specified in a user stanza within a class when the BACKFILL scheduler is in use.

Default: If the stanza does not specify **maxjobs**, or if there is no class, user, or group stanza at all, the maximum jobs is defined in the default stanza. The default is -1.

For more information, see “Controlling the mix of idle and running jobs” on page 668.

maxqueued

Specifies the maximum number of job steps a single group or user can have queued at the same time.

Administration file reference

Syntax:

`maxqueued = number`

Where *number* is the maximum number of jobs allowed in the queue for this user or group, depending on whether this keyword appears in a user or group stanza. This is the maximum number of jobs which can be either running or being considered to be dispatched by the negotiator for that user or group. Jobs above this maximum are placed in the NotQueued state. This action prevents one of the following situations:

- Individual users from dominating the number of jobs that are either running or are being considered to run.
- Groups from flooding the job queue.

Note: This keyword can be specified in a user stanza within a class.

Default: If the user or group stanza does not specify **maxqueued** or if there is no user or group stanza at all, the maximum number of jobs that can be simultaneously in queue for the user or group is defined in the default stanza. If no value is found, or the limit found is -1, then no limit is placed on the number of jobs that can be simultaneously idle for the user or group. Regardless of this limit, there is no limit to the number of jobs a user or group can submit.

For more information, see “Controlling the mix of idle and running jobs” on page 668.

multicluster_security

Specifies a security mechanism to use for authentication and authorization of intercluster communications.

Syntax:

`multicluster_security = SSL`

The only valid specification for this keyword is **SSL**. When **SSL** is specified, LoadLeveler uses the OpenSSL library to provide secure intercluster transactions. If this keyword is omitted or left blank and the **MACHINE_AUTHENTICATE** in the configuration file is set to **true**, then LoadLeveler will accept intercluster transactions only from machines listed as **inbound_hosts** or **outbound_hosts** in the administration file. Otherwise, intercluster transactions are accepted from any machine.

For more information, see “Steps for securing communications within a LoadLeveler multicluster” on page 143.

multilink_address

Specifies the multilink address used for IP striping on the associated adapter.

Syntax:

`multilink_address = ip_address`

Where *ip_address* indicates the IP address that includes the adapters that can be striped across.

multilink_list

Specifies the IP addresses of the adapters that this multilink device stripes across.

Syntax:

```
multilink_list = adapter_name <, adapter_name>*
```

Where *adapter_name* indicates multilinked devices which stripes IP addresses across the adapters given in the list.

name_server

Specifies a list of name servers used for a machine.

Syntax:

```
name_server = list
```

Where *list* is a blank-delimited list of character strings that is used to specify which nameservers are used for the machine. Valid strings are DNS, NIS, and LOCAL. LoadLeveler uses the list to determine when to append a DNS domain name for machine names specified in LoadLeveler commands issued from the machine described in this stanza.

If DNS is specified alone, LoadLeveler will always append the DNS domain name to machine names specified in LoadLeveler commands. If NIS or LOCAL is specified, LoadLeveler will never append a DNS domain name to machine names specified in LoadLeveler commands. If DNS is specified with either NIS or LOCAL, LoadLeveler will always look up the name in the administration file to determine whether to append a DNS domain name. If the name is specified with a trailing period, it doesn't append the domain name.

network_id

Specifies a unique numerical network identifier. This value is set by the `llextRPD` command and should not be changed.

Syntax:

```
network_id = number
```

Default: No default value is set.

network_type

Syntax:

```
network_type = string
```

Where *string* specifies the type of network that the adapter supports (for example, Ethernet). This should be unique for each communication path. This is an administrator defined name. This keyword defines the types of networks a user can specify in a job command file using the **network** keyword.

Default: No default value is set.

nice Increments the *nice* value of a job.

Syntax:

```
nice = value
```

Where *value* is the amount by which the current UNIX *nice* value is incremented. The *nice* value is one factor in a job's run priority. The lower the number, the higher the run priority. If two jobs are running on a machine, the *nice* value determines the percentage of the CPU allocated to each job.

Administration file reference

This value ranges from -20 to 20. Values out of this range are placed at the top (or bottom) of the range. For example, if your current *nice* value is 15, and you specify *nice* = 10, the resulting value is 20 (the upper limit) rather than 25. The default is 0.

If the administrator has decided to enforce consumable resources, the *nice* value will only adjust priorities of processes within the same WLM class. Because LoadLeveler defines a single class for every job step, the *nice* value has no effect.

For more information, consult the appropriate UNIX documentation.

outbound_hosts

Blank-delimited list of hostnames that define the machines configured for outbound connections to other clusters.

Syntax:

```
outbound_hosts = hostname[(cluster_name)] ...
```

Where *hostname* specifies a machine configured for outbound connections to other clusters and *cluster_name* can be used to specify a specific cluster if the host is not connected to all clusters in the multicluster. These hostnames must be fully qualified with domain names if the machines exist in a different domain. This keyword is required in a multicluster environment.

Note: The same machine can be defined as both an **outbound_host** and an **inbound_host**.

pool_list

Specifies a list of pool numbers to which the machine belongs. Do not use negative numbers in a machine *pool_list*.

Syntax:

```
pool_list = pool_numbers
```

Where *pool_numbers* is a blank-delimited list of non-negative numbers identifying pools to which the machine belongs. These numbers can be any positive integers including zero.

port_number

Specifies the port number of the InfiniBand adapter port.

Syntax:

```
port_number = number
```

Where *port_number* is the port number of the InfiniBand adapter port. The adapter stanza for InfiniBand support only contains the adapter port number; there is no InfiniBand adapter information in the adapter stanza.

priority

Identifies the priority of the appropriate user, class, or group.

Syntax:

```
priority = number
```

Where *number* is an integer that specifies the priority for jobs either in this class, or submitted by this user or group, depending on whether this keyword appears in a class, user, or group stanza, respectively.

The number specified for priority is referenced as either **ClassSysprio**, **UserSysprio**, or **GroupSysprio** in the configuration file. You can use **ClassSysprio**, **UserSysprio**, or **GroupSysprio** when assigning job priorities. If the variable **ClassSysprio**, **UserSysprio**, or **GroupSysprio** does not appear in the **SYSPRIO** expression in the configuration file, then the priority specified in the administration file is ignored. See “LoadLeveler variables” on page 294 for more information about the **ClassSysprio**, **UserSysprio**, or **GroupSysprio** keywords.

Default: The default is 0.

reservation_permitted

Specifies whether the machine can be reserved through new reservation requests.

Syntax:

```
reservation_permitted = true | false
```

If the value of this keyword is changed to false for this machine when it already is reserved through existing reservations, LoadLeveler will reserve this machine until those existing reservations complete or are canceled.

Default: true, which means that this machine can be reserved through new reservation requests.

resources

Specifies quantities of the consumable resources initially available on the machine.

Syntax:

```
resources = name(count) name(count) ... name(count)
```

Where *name(count)* is an administrator-defined name and count, or could also be **ConsumableCpus(count)**, **ConsumableMemory(count units)**, or **ConsumableVirtualMemory(count units)**. **ConsumableMemory** and **ConsumableVirtualMemory** are the only two consumable resources that can be specified with both a count and units. The count for each specified resource must be an integer greater than or equal to zero. The allowable units are those normally used with LoadLeveler data limits:

```
b bytes
w words
kb kilobytes (2**10 bytes)
kw kilowords (2**12 bytes)
mb megabytes (2**20 bytes)
mw megawords (2**22 bytes)
gb gigabytes (2**30 bytes)
gw gigawords (2**32 bytes)
tb terabytes (2**40 bytes)
tw terawords (2**42 bytes)
pb petabytes (2**50 bytes)
pw petawords (2**52 bytes)
eb exabytes (2**60 bytes)
ew exawords (2**62 bytes)
```

The **ConsumableMemory** and **ConsumableVirtualMemory** resource values are stored in mb (megabytes) and rounded up. Therefore, the smallest amount of **ConsumableMemory** or **ConsumableVirtualMemory** which you can request is one megabyte. If no units are specified, then megabytes

Administration file reference

are assumed. Resources defined here that are not in the **SCHEDULE_BY_RESOURCES** list in the global configuration file will not effect the scheduling of the job.

For the **ConsumableCPUs** resource, a value of **all** can be specified instead of count. This indicates that the CPU resource value will be obtained from the Startd daemons. However, these resources will not be available for scheduling until the first **Startd** update.

Also for the **ConsumableCPUs**, when the **RSET_SUPPORT** keyword is set to one of the options enabling affinity, a list of CPU IDs can be specified. A list within < > angle brackets indicates a list of CPU IDs. Only CPUs with logical IDs specified in the list will be considered available for LoadLeveler jobs. The following example specifies a list of CPUs:

```
resources = consumableCPUs< 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 >
```

CPU IDs can also be specified using a list of ranges:

```
resources = consumableCPUs< 0-6 10-16 >
```

If Rset support is enabled with either the **RSET_MCM_AFFINITY** or **RSET_CONSUMABLE_CPUS** option, this keyword must be used to specify the exact CPU logical IDs of consumable CPUs in the administration file. If the **all** reserved word is used, all CPUs will be considered by LoadLeveler.

The logical IDs of the CPUs available on a machine can be found issuing the `bindprocessor -q` command.

Default: No default value is set.

rss_limit

Specifies the hard limit, soft limit, or both limits for the resident set size for a job.

Syntax:

```
rss_limit = hardlimit,softlimit
```

For additional information about limit keywords, see the following topics:

- “Syntax for limit keywords” on page 304
- “Using limit keywords” on page 83

schedd_fenced

Specifies whether or not the central manager is to ignore connections from the schedd daemon running on this machine.

Syntax:

```
schedd_fenced = true | false
```

Where **true** specifies that the central manager ignores connections from the schedd daemon running on this machine. Use the **true** setting together with the `llctl -h host purgeschedd` command when you want to attempt to recover resources lost when a node running the schedd daemon fails. A **true** setting prevents conflicts from arising when a schedd machine is restarted while a purge is taking place. For more information, see “How do I recover resources allocated by a Schedd machine?” on page 660.

Default: **false**

schedd_host

Specifies whether or not this machine is used to help submit-only machines access LoadLeveler hosts that run LoadLeveler jobs.

Syntax:

`schedd_host = true | false`

When **true** this keyword specifies that if a schedd is running on a machine that it will serve as a public scheduling machine. A public scheduling machine accepts job submissions from other machines in the LoadLeveler cluster. Jobs are submitted to a public scheduling machine if:

- The submission occurs on a machine which does not run the schedd daemon. These include submit-only machines and machines which are configured to run other LoadLeveler daemons but not the schedd daemon.
- The submission occurs on a machine which runs the schedd daemon but is configured to submit jobs to a public scheduling machine by having the **SCHEDD_SUBMIT_AFFINITY** keyword set to **false** in the global or local configuration file.

This keyword does not configure LoadLeveler to run the schedd daemon on a node. Use the configuration keyword **SCHEDD_RUNS_HERE** to run the schedd daemon on a node. Refer to the **SCHEDD_RUNS_HERE** keyword for more information.

Default: false

secure_schedd_port

Specifies the port number to use to connect to the Schedd for secure inbound transactions to this cluster.

Syntax:

`secure_schedd_port = port_number`

Where *port_number* is a positive integer that specifies the port number used to connect to the Schedd for secure inbound transactions to this cluster. This port is only used if the **multicluster_security** keyword is set to **SSL**. The secure schedd port should be different from the normal schedd port.

Default: 9607

smt

Specifies the parallel job steps that require dynamic simultaneous multithreading (SMT) to be turned on or off before they are run.

Syntax:

`smt = yes | no`

where **yes** indicates that LoadLeveler will invoke AIX service to turn on SMT on the node. This will be true for all the nodes where the parallel job will be running.

Default value: no.

Examples:

`smt = yes`

Administration file reference

spacct_exclude_enable

Specifies whether the SP accounting function is informed whenever this machine is being used exclusively by a particular job.

Syntax:

```
spacct_exclude_enable = true | false
```

Where **true** specifies that the accounting function on an SP system is informed that a job step has exclusive use of this machine. Note that your SP system must have exclusive user accounting enabled in order for this keyword to have an effect. For more information on SP accounting, see *Parallel System Support Programs for AIX: Administration Guide*, GC23-3899.

Default: false

speed Specifies the weight associated with the machine for scheduling purposes.

Syntax:

```
speed = number
```

Where *number* is a floating point number that is used for machine scheduling purposes in the **MACHPRIO** expression. For more information on machine scheduling and the MACHPRIO expression, see “Setting negotiator characteristics and policies” on page 43. In addition, the **speed** keyword is also used to define the weight associated with the machine. This weight is used when gathering accounting data on a machine basis.

To distinguish speed among different machines, you must include this value in the local configuration file. For information on how the **speed** keyword can be used to schedule machines, refer to “Setting negotiator characteristics and policies” on page 43.

Default: The default is 1.0.

ssl_cipher_list

Specifies a cipher list defining what encryption methods are available to OpenSSL when securing multicluster connections.

Syntax:

```
ssl_cipher_list = cipher_list
```

Where *cipher_list* is a valid cipher list as documented by the OpenSSL **ciphers** command.

Default: This keyword will default to the “ALL:eNULL:!aNULL” string.

stack_limit

Specifies the hard limit, soft limit, or both limits for the size of a stack.

Syntax:

```
stack_limit = hardlimit,softlimit
```

For additional information about limit keywords, see the following topics:

- Syntax for limit keywords
- Using limit keywords

submit_only

Specifies whether or not this machine is a submit-only machine.

Syntax:

```
submit_only = true| false
```

Where **true** designates this as a submit-only machine. If you set this keyword to **true**, in the administration file set **central_manager** and **schedd_host** to **false**.

Default: false

total_tasks

Specifies the maximum number of tasks that can be requested for a particular class or by a particular user or group for a parallel job.

Syntax:

```
total_tasks = number
```

Where *number* specifies the maximum number of tasks for a parallel job in a job command file using the **total_tasks** keyword.

Default: The default is -1, which means there is no limit.

type Identifies the type of stanza in the administration file.

Syntax:

```
type = stanza_type
```

Where *stanza_type* is one of the following:

- Adapter
- Class
- Group
- Machine
- User

Default: No default value is set.

wall_clock_limit

Specifies the hard limit, soft limit, or both limits for the amount of elapsed time for which a job can run.

Syntax:

```
wall_clock_limit = hardlimit,softlimit
```

Note that LoadLeveler uses the time the negotiator daemon dispatches the job as the start time of the job. When a job is checkpointed, vacated, and then restarted, the **wall_clock_limit** is not adjusted to account for the amount of time that elapsed before the checkpoint occurred.

If you are running the BACKFILL scheduler, you must set a wall clock limit either in the job command file or in a class stanza (for the class associated with the job you submit). LoadLeveler administrators should consider setting a default wall clock limit in a default class stanza. For more information on setting a wall clock limit when using the BACKFILL scheduler, see “Choosing a scheduler” on page 42.

For additional information about limit keywords, see the following topics:

- “Syntax for limit keywords” on page 304
- “Using limit keywords” on page 83

Administration file reference

Chapter 14. Job command file reference

A LoadLeveler job consists of one or more job steps, each of which is defined in a single job command file. A job command file specifies the name of the job, as well as the job steps that you want to submit, and can contain other LoadLeveler statements.

Table 76 lists the job command file subtasks:

Table 76. Job command file subtasks

Subtask	Associated information (see . . .)
To find out how to work with a job command file	Chapter 8, "Building and submitting jobs," on page 165
To learn how to correctly specify the contents of a job command file	<ul style="list-style-type: none">• "Job command file syntax"• "Job command file keyword descriptions" on page 336

Job command file syntax

The following general rules apply to job command files.

- Keyword statements begin with # @. There can be any number of blanks between the # and the @.
- Comments begin with #. Any line whose first non-blank character is a pound sign (#) and is not a LoadLeveler keyword statement is regarded as a comment.
- Statement components are separated by blanks. You can use blanks before or after other delimiters to improve readability but they are not required if another delimiter is used.
- The back-slash (\) is the line continuation character. Note that the continued line must not begin with # @. If your job command file is the script to be executed, you must start the continued line with a #. See Example 2 and Example 3 in topic Examples: Job command files for examples that use the back-slash for line continuation.
- Keywords are *not* case sensitive. This means you can enter them in lower case, upper case, or mixed case.

Serial job command file

Figure 41 on page 334 is an example of a simple serial job command file which is run from the current working directory. The job command file reads the input file, **longjob.in1**, from the current working directory and writes standard output and standard error files, **longjob.out1** and **longjob.err1**, respectively, to the current working directory.

Job command file reference

```
# The name of this job command file is file.cmd.
# The input file is longjob.in1 and the error file is
# longjob.err1. The queue statement marks the end of
# the job step.
#
# @ executable = longjob
# @ input = longjob.in1
# @ output = longjob.out1
# @ error = longjob.err1
# @ queue
```

Figure 41. Serial job command file

Parallel job command file

In addition to building job command files to submit serial jobs, you can also build job command files to submit parallel jobs. Before constructing parallel job command files, consult your LoadLeveler system administrator to see if your installation is configured for parallel batch job submission.

For more information on submitting parallel jobs, see “Working with parallel jobs” on page 178.

Syntax for limit keywords

The syntax for setting a limit is:

```
limit_type = hardlimit,softlimit
```

For example:

```
core_limit = 120kb,100kb
```

To specify only a hard limit, you can enter, for example:

```
core_limit = 120kb
```

To specify only a soft limit, you can enter, for example:

```
core_limit = ,100kb
```

In a keyword statement, you cannot have any blanks between the numerical value (100 in the above example) and the units (kb). Also, you cannot have any blanks to the left or right of the comma when you define a limit in a job command file.

For limit keywords that refer to a data limit — such as **data_limit**, **core_limit**, **file_limit**, **stack_limit**, and **rss_limit** — the hard limit and the soft limit are expressed as:

```
integer[.fraction][units]
```

The allowable units for these limits are:

```
b bytes
w words
kb kilobytes (2**10 bytes)
kw kilowords (2**12 bytes)
mb megabytes (2**20 bytes)
mw megawords (2**22 bytes)
gb gigabytes (2**30 bytes)
gw gigawords (2**32 bytes)
tb terabytes (2**40 bytes)
tw terawords (2**42 bytes)
```

pb petabytes (2**50 bytes)
 pw petawords (2**52 bytes)
 eb exabytes (2**60 bytes)
 ew exawords (2**62 bytes)

If no units are specified for data limits, then bytes are assumed.

For limit keywords that refer to a time limit — such as **ckpt_time_limit**, **cpu_limit**, **job_cpu_limit**, and **wall_clock_limit** — the hard limit and the soft limit are expressed as:

[[hours:]minutes:]seconds[.fraction]

Fractions are rounded to seconds.

You can use the following character strings with all limit keywords except the **copy** keyword for **wall_clock_limit**, **job_cpu_limit**, and **ckpt_time_limit**:

rlim_infinity	Represents the largest positive number.
unlimited	Has same effect as rlim_infinity .
copy	Uses the limit currently active when the job is submitted.

64-bit support for job command file keywords

Users can assign 64-bit integer values to selected keywords in the job command file. System resource limits, with the exception of CPU limits, are treated by LoadLeveler daemons and commands as 64-bit limits.

Table 77 describes 64-bit support for specific job command file keywords.

Table 77. Notes on 64-bit support for job command file keywords

Keyword name	Notes
ckpt_time_limit	Not supported. The hard and soft time limits associated with this keyword are 32-bit integers. If a value that cannot be contained in a 32-bit integer is assigned to this limit, the value will be truncated to either 2147483647 or -2147483648.
core_limit	64-bit integer values may be assigned to this limit. Fractional specifications are allowed and will be converted to 64-bit integer values. Refer to the allowable units for these limits listed under “Syntax for limit keywords” on page 334.
cpu_limit	Not supported. The hard and soft time limits associated with this keyword are 32-bit integers. If a value that cannot be contained in a 32-bit integer is assigned to this limit, the value will be truncated to either 2147483647 or -2147483648.
data_limit	64-bit integer values may be assigned to these limits. Fractional specifications are allowed and will be converted to 64-bit integer values.
file_limit	Refer to the allowable units for these limits listed under “Syntax for limit keywords” on page 334.
image_size	64-bit integer values may be assigned to this keyword. Fractional and unit specifications are not allowed. The default unit of image_size is kb. Example: image_size = 12345678901

Job command file reference

Table 77. Notes on 64-bit support for job command file keywords (continued)

Keyword name	Notes
job_cpu_limit	Not supported. The hard and soft time limits associated with this keyword are 32-bit integers. If a value that cannot be contained in a 32-bit integer is assigned to this limit, the value will be truncated to either 2147483647 or -2147483648.
preferences	64-bit integer values may be associated with the LoadLeveler variables "Memory" and "Disk" in the expressions assigned to these keywords. Fractional and unit specifications are not allowed.
requirements	Examples: requirements = (Arch == "R6000") && (Disk > 500000000) && (Memory > 6000000000) preferences = (Disk > 6000000000) && (Memory > 9000000000)
resources	Consumable resources associated with the resources keyword may be assigned 64-bit integer values. Fractional specifications are not allowed. Unit specifications are valid only when specifying the values of the predefined ConsumableMemory and ConsumableVirtualMemory resources. Examples: resources = spice2g6(123456789012) ConsumableMemory(10 gb) resources = ConsumableVirtualMemory(15 pb) db2_license(1)
rss_limit	64-bit integer values may be assigned to these limits. Fractional specifications are allowed and will be converted to 64-bit integer values.
stack_limit	Refer to the allowable units for these limits listed under "Syntax for limit keywords" on page 334.
wall_clock_limit	Not supported. The hard and soft time limits associated with this keyword are 32-bit integers. If a value that cannot be contained in a 32-bit integer is assigned to this limit, the value will be truncated to either 2147483647 or -2147483648.

Job command file keyword descriptions

This topic provides an alphabetical list of the keywords you can use in a LoadLeveler script. It also provides examples of statements that use these keywords. For most keywords, if you specify the keyword in a job step of a multistep job, its value is inherited by all proceeding job steps. Exceptions to this are noted in the keyword description.

If a blank value is used after the equal sign, it is as if no keyword was specified.

account_no

Supports centralized accounting. Allows you to specify an account number to associate with a job. This account number is stored with job resource information in local and global history files. It may also be validated before LoadLeveler allows a job to be submitted. For more information, see "Gathering job accounting data" on page 57.

Syntax:

account_no = *string*

where *string* is a text string that can consist of a combination of numbers and letters.

Default value: No default value is set.

Example: If the job accounting group charges for job time based upon the department to which you belong, your account number would be similar to:

```
account_no = dept34ca
```

arguments

Specifies the list of arguments to pass to your program when your job runs.

Syntax:

```
arguments = arg1 arg2 ...
```

Default value: No default arguments are set.

Example: If your job requires the numbers 5, 8, 9 as input, your arguments keyword would be similar to:

```
arguments = 5 8 9
```

bg_connection

Specifies the type of wiring requested for the Blue Gene partition in which the job step will run.

Syntax:

```
bg_connection = TORUS | MESH | PREFER_TORUS
```

where:

TORUS

Specifies that the admissible partitions must be wireable as a torus.

MESH

Specifies that the admissible partitions must be wireable as a mesh.

PREFER_TORUS

Specifies that the admissible partitions should be wireable as a torus, but if there are no such partitions then the selected partition must be wireable as a mesh.

This keyword is only valid for job type **bluegene**. This keyword cannot be used if the **bg_partition** keyword is specified. This keyword is not inherited by other job steps.

Default value: **MESH** is the default value.

bg_partition

Specifies the ID of the Blue Gene partition that the job will run in.

Syntax:

```
bg_partition = partition_id
```

where *partition_id* is a string identifying a partition in the Blue Gene system.

Job command file reference

This keyword is only valid for job type **bluegene**. This keyword cannot be used if the **bg_connection**, **bg_requirements**, **bg_shape**, or the **bg_size** keyword is specified. This keyword is not inherited by other job steps.

Default value: No default is set.

bg_requirements

Specifies the requirements which a Blue Gene base partition in the LoadLeveler cluster must meet to run any job steps.

Syntax:

bg_requirements = Boolean_expression

The only requirement supported at this time is memory, where memory specifies the amount, in megabytes, of regular physical memory required in the C-nodes of the Blue Gene base partition where you want your job step to run.

Example 1: To require Blue Gene base partitions with 512 megabytes of physical memory in their C-nodes, enter:

```
bg_requirements = (Memory == 512)
```

Example 2: To require Blue Gene base partitions with more than 512 megabytes of physical memory in their C-nodes, enter:

```
bg_requirements = (Memory > 512)
```

This keyword is only valid for job type **bluegene**. This keyword cannot be used if the **bg_partition** keyword is specified. This keyword is not inherited by other job steps.

Default value: No default value is set.

bg_rotate

Specifies whether the scheduler should consider all possible rotations of the given shape of the job when searching for a partition for the job.

Syntax:

bg_rotate = true | false

where **true** implies that the shape can be rotated to fit some free resource and **false** implies that the shape will not be rotated.

Assigning a value of **true** to this keyword will increase the likelihood of the scheduler finding a partition to run the job and optimizes overall scheduling of Blue Gene resources. **bg_rotate** must be set to **false** when using the **mapfile** argument of **mpirun** to specify how the job's tasks are to be assigned to the allocated compute nodes.

This keyword is only valid for job type **bluegene**. This keyword is not inherited by other job steps.

Note: This keyword can only be used in conjunction with the **bg_shape** job command file keyword. If **bg_shape** is not present, this keyword is ignored.

Default value: The default value is true.

bg_shape

Specifies the requested shape of the Blue Gene job to be started in the system.

Syntax:

bg_shape = *XxYxZ*

where *X*, *Y*, and *Z* are positive integers indicating the number of base partitions in the *X*-direction, *Y*-direction, and *Z*-direction, respectively, of the requested job shape. The values of *X*, *Y*, and *Z* or their rotations, if **bg_rotate** is **true**, must not be greater than the corresponding *X*, *Y*, and *Z* sizes of the Blue Gene system, otherwise the job will never be able to start.

This keyword is only valid for job type **bluegene**. This keyword can not be used if the **bg_partition** or **bg_size** keyword is specified. This keyword is not inherited by other job steps.

Note: The *X*, *Y*, and *Z* dimensions of the allocated partition will be exactly as defined by the **bg_shape** job command file keyword *unless* the job command file keyword **bg_rotate** is specified as **true**, in which case all possible rotations of the dimensions are possible.

Default value: No default is set.

bg_size

Specifies the requested size of the Blue Gene job to be started in the system.

Syntax:

bg_size = *bg_size*

where *bg_size* is an integer indicating the size of the job in units of compute nodes. No guarantees are made as to the shape of the allocated partition for a given size. The only guarantee is that the size of the allocated shape will be no smaller than the requested size and as close to the request size as possible.

This keyword is only valid for job type **bluegene**. This keyword can not be used if the **bg_partition** or **bg_shape** keyword is specified. This keyword is not inherited by other job steps.

Note: Not all values given for **bg_size** are representable. For example, consider an 8x4x4 Blue Gene system in units of base partitions and a requested **bg_size** of 5632 (equivalent to 11 base partitions). Since 11 is a prime number, it cannot be decomposed. Furthermore, it is greater than any one dimension of the system. In this case, a 3x4x1 partition is allocated, since it is the smallest number of base partitions larger than the requested size.

Default value: If `bg_size`, `bg_shape`, or `bg_partition` are not specified then `bg_size` defaults to the configured minimum partition size. This is the value of the `BG_MIN_PARTITION_SIZE` keyword in the configuration file.

blocking

Blocking specifies that tasks be assigned to machines in multiples of a certain integer. Unlimited blocking specifies that tasks be assigned to each machine until it runs out of initiators, at which time tasks will be assigned to the machine which is next in the order of priority. If the total number of tasks are not evenly divisible by the blocking factor, the remainder of tasks are allocated to a single node.

This keyword is supported by the BACKFILL and API schedulers.

Syntax:

blocking = *integer* | **unlimited**

where:

integer

Specifies the blocking factor to be used. The blocking factor must be a positive integer. With a blocking factor of 4, LoadLeveler will allocate 4 tasks at a time to each machine with at least 4 initiators available. This keyword must be specified with the `total_tasks` keyword. **Example:**

```
blocking = 4
total_tasks = 17
```

LoadLeveler will allocate tasks to machines in an order based on the values of their MACHPRIO expressions (beginning with the highest MACHPRIO value). In cases where `total_tasks` is not a multiple of the blocking factor, LoadLeveler assigns the remaining number of tasks as soon as possible (even if that means assigning the remainder to a machine at the same time as it assigns another block).

unlimited

Specifies that LoadLeveler allocate as many tasks as possible to each machine, until all of the tasks have been allocated. LoadLeveler will prioritize machines based on the number of initiators each machine currently has available. Unlimited blocking is the only means of allocating tasks to nodes that does not prioritize machines primarily by MACHPRIO expression.

Default value: No default is set, which means that no blocking is requested.

bulkxfer

Indicates whether the communication subsystem will use bulk data transfer for user space communication.

Syntax:

bulkxfer = *yes* | **no**

Default: **no**

For additional information about bulk data transfer, see “Using bulk data transfer” on page 173.

checkpoint

Indicates if a job is able to be checkpointed. Checkpointing a job is a way of saving the state of the job so that if the job does not complete it can be restarted from the saved state rather than starting the job from the beginning.

If you specify a value that is not valid for the **checkpoint** keyword, an error message is generated and the job is not submitted.

Syntax:

`checkpoint = interval | yes | no`

Where:

interval

Specifies that LoadLeveler will automatically checkpoint your program at preset intervals. The time interval is specified by the settings in the **MIN_CKPT_INTERVAL** and **MAX_CKPT_INTERVAL** keywords in the configuration file. Since a job with a setting of **interval** is considered checkpointable, you can initiate a checkpoint using any method in addition to the automatic checkpoint. The difference between **interval** and **yes** is that **interval** enables LoadLeveler to automatically take checkpoints on the specified intervals while the value **yes** does not enable that ability.

yes Enables a job step to be checkpointed. With this setting, a checkpoint can be initiated either under the control of an application or by a method external to the application. With a setting of **yes**, LoadLeveler will not checkpoint on the intervals specified by the **MIN_CKPT_INTERVAL** and **MAX_CKPT_INTERVAL** keywords in the configuration file. The difference between **yes** and **interval** is that **interval** enables LoadLeveler to automatically take checkpoints on the specified intervals while the value **yes** does not enable that ability.

no The step cannot be checkpointed.

Default value: no

Restriction: On Linux machines only: If a job with **checkpoint = interval** or **checkpoint = yes** is dispatched, it is rejected.

Example: If a checkpoint is initiated from within the application but checkpoints are not to be taken automatically by LoadLeveler you can use:

```
checkpoint = yes
```

For detailed information on checkpointing, see “LoadLeveler support for checkpointing jobs” on page 129.

ckpt_dir

Specifies the directory which contains the checkpoint file.

Job command file reference

Checkpoint files can become quite large. When specifying `ckpt_dir`, make sure that there is sufficient disk space to contain the files. Guidelines can be found in “LoadLeveler support for checkpointing jobs” on page 129.

Syntax:

`ckpt_dir = pathname`

The values for `ckpt_dir` are case sensitive.

Default value: The value of the `ckpt_dir` keyword in the class stanza of the administration file

Restriction: The keyword `ckpt_dir` is not allowed in the command file for interactive POE sessions.

Example: If checkpoint files were to be stored in the `/tmp` directory the job command file would include:

```
ckpt_dir = /tmp
```

For more information on naming directories for checkpointing, see “Naming checkpoint files and directories” on page 134.

`ckpt_execute_dir`

Specifies the directory where the job step’s executable will be saved for checkpointable jobs. You may specify this keyword in either the configuration file or the job command file; different file permissions are required depending on where this keyword is set. For additional information, see “Planning considerations for checkpointing jobs” on page 130.

Syntax:

`ckpt_execute_dir = directory`

This directory cannot be the same as the current location of the executable file, or LoadLeveler will not stage the executable. In this case, the user must have execute permission for the current executable file.

Default value: No default value is set.

`ckpt_file`

Used to specify the base name of the checkpoint file. The checkpoint file is created by the AIX checkpoint functions and is derived from the filename specified in the `ckpt_file` keyword in the job command file or the default file name.

Syntax:

`ckpt_file = filename`

The value for the `ckpt_file` keyword is case sensitive.

Default value: `[jobname.]job_step_id.ckpt`

Restriction: The keyword `ckpt_file` is not allowed in the command file for interactive POE sessions.

Example: If you are storing checkpoint files in a file with the base name "myckptfiles" which is placed in the directory named by the **ckpt_dir** keyword, the job command file would contain:

```
ckpt_file = myckptfiles
```

Alternatively, if you are naming the checkpoint files "myckptfiles" and storing them in the directory /tmp, the keyword in the job command file can contain:

```
ckpt_file = /tmp/myckptfiles
```

Or the combination of **ckpt_dir** and **ckpt_file** keywords can be used, producing the same result.

```
ckpt_dir = /tmp
ckpt_file = myckptfiles
```

For more information on naming files for checkpointing, see "Naming checkpoint files and directories" on page 134.

ckpt_time_limit

Specifies the hard or soft limit, or both limits for the elapsed time checkpointing a job can take. When the soft limit is exceeded, LoadLeveler will attempt to stop the checkpoint and allow the job to continue. If the checkpoint is not able to be stopped and the hard limit is exceeded, LoadLeveler will terminate the job.

Syntax:

```
ckpt_time_limit = hardlimit,softlimit
```

Default value: The value of the **ckpt_time_limit** keyword in the class stanza of the administration file

Examples:

```
ckpt_time_limit = 00:10:00,00:05:00
ckpt_time_limit = 12:30,7:10
ckpt_time_limit = rlim_infinity
ckpt_time_limit = unlimited
```

For more information about the values and units you can use with this keyword, and how limits are enforced, see "Using limit keywords" on page 83.

class

Specifies the name of a job class defined locally in your cluster. You can use the **llclass** command to find out information on job classes.

Syntax:

```
class = name
```

Default value: If you do not specify a value for this keyword, the default job class, **No_Class**, is assigned.

Example: If you are allowed to submit jobs belonging to a class called "largejobs", your class keyword would look like the following:

```
class = largejobs
```

Job command file reference

cluster_input_file

Specifies an individual file to be copied from the local pathname to the remote pathname when the job is run.

Syntax:

cluster_input_file = *local_pathname*, *remote_pathname*

where:

local_pathname

Specifies the full pathname of the file to be copied from the local cluster. This file must be accessible by the submitting user on the node where the local gateway Schedd runs.

local_pathname must be specified.

remote_pathname

Specifies the full pathname the file will be copied to on the assigned cluster. This file must be accessible by the mapped user on the Schedd node of the selected cluster.

remote_pathname must be specified. Normally the file specified by *remote_pathname* will be deleted following the job termination. It will not be deleted if the cluster selected to run the job is the same cluster where the job was submitted and *remote_pathname* resolves to the same pathname specified as *local_pathname*.

If LoadLeveler fails to copy an input file to the selected cluster, the assignment of the job to the selected cluster will fail. If the cluster was assigned by the administrator using the **llmovejob** command, an error message will be displayed in the command response describing the reason for failure and the job will remain in the cluster it was in and be placed in system hold. If the cluster was assigned during job submission, the job submission fails and an error message will be displayed in the command response describing the reason for failure.

Default value: No default value is set.

cluster_list

Allows you to specify that a job is to run on a particular cluster or that LoadLeveler is to decide which cluster is best from the list of clusters specified. If this keyword is specified, it must be in the first job step of a multistep job. Any definitions in other steps are ignored.

Syntax:

cluster_list = *cluster_list*

Where *cluster_list* is a blank-delimited list of cluster names or the reserved word **any**. Depending on the specified value, **cluster_list** can have one of three effects:

- Specifying a single cluster name indicates that a job is to be submitted to that cluster.
- Specifying a list of multiple cluster names indicates that the job is to be submitted to one of the clusters specified with the installation exit **CLUSTER_METRIC** choosing from the list.

- Specifying the reserved word **any** indicates the job is to be submitted to any cluster defined by the installation exit **CLUSTER_METRIC**.

Note: If a cluster list is specified using either the **llsubmit -X** command or the **ll_cluster** API, then that cluster list takes precedence over a **cluster_list** specified in the job command file.

cluster_output_file

Specifies an individual output file to be copied to the submitting cluster from the cluster selected to run the job after the job completes.

Syntax:

cluster_output_file = *local_pathname*, *remote_pathname*

where:

local_pathname

Specifies the full pathname the file will be copied to on the local cluster. This file must be accessible by the submitting user on the node where the local gateway Schedd runs.

local_pathname must be specified.

remote_pathname

Specifies the full pathname of the file that will be copied from the assigned cluster. This file must be accessible by the mapped user on the Schedd node of the selected cluster.

remote_pathname must be specified. Normally the file specified by *remote_pathname* will be deleted following the job termination. It will not be deleted if the cluster selected to run the job is the same cluster where the job was submitted and *remote_pathname* resolves to the same pathname specified as *local_pathname*.

If LoadLeveler fails to copy an output file from a selected cluster to the local cluster during job termination, the job termination will proceed and the remote file will not be deleted. Mail will be sent to the user describing the reason for the failed copy.

Default value: No default value is set.

comment

Specifies text describing characteristics or distinguishing features of the job.

core_limit

Specifies the hard limit, soft limit, or both limits for the size of a core file. This limit is a per process limit.

Syntax:

core_limit = *hardlimit*,*softlimit*

This keyword accepts both 32-bit and 64-bit integer values.

Default value: No default value is set.

Examples:

```
core_limit = 125621,10kb
core_limit = 5621kb,5000kb
core_limit = 2mb,1.5mb
core_limit = 2.5mw
core_limit = unlimited
core_limit = rlim_infinity
core_limit = copy
```

For more information about the values and units you can use with this keyword, and how limits are enforced, see “Using limit keywords” on page 83.

coschedule

Specifies the steps within a job that are to be scheduled and dispatched at the same time.

This keyword is supported only by the BACKFILL scheduler.

Syntax:

```
coschedule = true | false
```

where **true** implies that the step is to be coscheduled with all other steps in the job that have the value of this keyword set to true. This keyword is not inherited by other job steps.

Default value: The default value is set to false.

Examples:

```
coschedule = true
```

cpu_limit

Specifies the hard limit, soft limit, or both limits for the amount of CPU time that a submitted job step can use. This limit is a per process limit.

Syntax:

```
cpu_limit = hardlimit,softlimit
```

Default value: No default value is set.

Examples:

```
cpu_limit = 12:56:21,12:50:00
cpu_limit = 56:21.5
cpu_limit = 1:03,21
cpu_limit = unlimited
cpu_limit = rlim_infinity
cpu_limit = copy
```

For more information about the values and units you can use with this keyword, and how limits are enforced, see “Using limit keywords” on page 83.

data_limit

Specifies the hard limit, soft limit, or both limits for the size of the data segment to be used by the job step. This limit is a per process limit.

Syntax:

data_limit = *hardlimit,softlimit*

This keyword accepts both 32-bit and 64-bit integer values.

Default value: No default value is set.

Examples:

```
data_limit = ,125621
data_limit = 5621kb
data_limit = 2mb
data_limit = 2.5mw,2mb
```

For more information about the values and units you can use with this keyword, and how limits are enforced, see “Using limit keywords” on page 83.

dependency

Specifies the dependencies between job steps. A job dependency, if used in a given job step, must be explicitly specified for that step.

Syntax:

dependency = *step_name operator value*

where:

step_name

Is the name of a previously defined job step (as described in the *step_name* keyword).

operator

Is one of the following:

==	Equal to
!=	Not equal to
<=	Less than or equal to
>=	Greater than or equal to
<	Less than
>	Greater than
&&	And
	Or

value

Is usually a number that specifies the job return code to which the *step_name* is set. It can also be one of the following

LoadLeveler defined job step return codes:

CC_NOTRUN

The return code set by LoadLeveler for a job step which is not run because the dependency is not met.

The value of CC_NOTRUN is 1002.

CC_REMOVED

The return code set by LoadLeveler for a job step which is removed from the system (because, for example, **llcancel** was issued against the job step). The value of CC_REMOVED is 1001.

Default value: No default value is set.

Examples: The following are examples of dependency statements:

- **Example 1:** In the following example, the step that contains this dependency statement will run if the return code from step 1 is zero:

```
dependency = (step1 == 0)
```

- **Example 2:** In the following example, step1 will run with the executable called **myprogram1**. Step2 will run only if LoadLeveler removes step1 from the system. If step2 does run, the executable called **myprogram2** gets run.

```
# Beginning of step1
# @ step_name = step1
# @ executable = myprogram1
# @ ...
# @ queue
# Beginning of step2
# @ step_name = step2
# @ dependency = step1 == CC_REMOVED
# @ executable = myprogram2
# @ ...
# @ queue
```

- **Example 3:** In the following example, step1 will run with the executable called **myprogram1**. Step2 will run if the return code of step1 equals zero. If the return code of step1 does not equal zero, step2 does not get executed. If step2 is not run, the dependency statement in step3 gets evaluated and it is determined that step2 did not run. Therefore, **myprogram3** gets executed.

```
# Beginning of step1
# @ step_name = step1
# @ executable = myprogram1
# @ ...
# @ queue
# Beginning of step2
# @ step_name = step2
# @ dependency = step1 == 0
# @ executable = myprogram2
# @ ...
# @ queue
# Beginning of step3
# @ step_name = step3
# @ dependency = step2 == CC_NOTRUN
# @ executable = myprogram3
# @ ...
# @ queue
```

- **Example 4:** In the following example, the step that contains step2 returns a non-negative value if successful. This step should take into account the fact that LoadLeveler uses a value of 1001 for CC_REMOVED and 1002 for CC_NOTRUN. This is done with the following dependency statement:

```
dependency = (step2 >= 0) && (step2 < CC_REMOVED)
```

env_copy

Specifies whether environment variables for a batch or interactive parallel job are copied to all executing nodes, or to only the master node. When **all** is specified either explicitly or by default, any environment variables (specified by the **environment** keyword in the job command file) will be copied to all nodes where the job step runs. When **master** is specified, the environment variables will be copied only to the node selected to run the master task of the parallel job.

Although a LoadLeveler administrator may set this keyword in one or more class, group, or user stanzas in the administration file, an explicit setting in the job command file overrides any settings in the administration file that are relevant for the parallel job.

LoadLeveler ignores this keyword if it is set for a serial job.

Syntax:

env_copy = all | master

Default value: LoadLeveler uses the default value all only when both of the following conditions are true:

- The **env_copy** keyword is not specified in the job command file.
- The **env_copy** keyword is not specified in any class, group, or user stanza that is relevant to the parallel job.

environment

Specifies login initial environment variables set by LoadLeveler when your job step starts. If the same environment variables are set in the user's initialization files (such as the .profile), those set by the login initialization files will supersede those set by LoadLeveler.

You may use the **env_copy** keyword to instruct LoadLeveler to copy these environment variables to all executing nodes, or to only the master executing node.

Syntax:

environment = *env1* ; *env2* ; ...

Separate environment specifications (*env1*, *env2*, and so on) with semicolons. An environment specification may be one of the following:

COPY_ALL

Specifies that all the environment variables from your shell be copied.

\$var Specifies that the environment variable *var* be copied into the environment of your job when LoadLeveler starts it.

!var Specifies that the environment variable *var* not be copied into the environment of your job when LoadLeveler starts it. This specification is most useful together with COPY_ALL.

var=value

Specifies that the environment variable *var* be set to the value "value" and copied into the environment of your job when LoadLeveler starts it.

When processing the string you specify for *var*, LoadLeveler first removes any leading or trailing blanks, and copies the remaining string, as is, into the environment.

Default value: No default value is set.

Additional considerations:

If you specify the **environment** job command file keyword with **COPY_ALL**, the **\$USER** and **\$HOME** environment variables from

Job command file reference

your shell are not copied and set when your job step starts. The **\$USER** and **\$HOME** environment variables of the user ID on the executing node will be set. If you explicitly specify **\$USER** or **\$HOME** it will be copied and set when your job step starts.

If more than one environment specification is defined for the same environment keyword, the rightmost specification takes precedence. For example if you specify:

```
environment = COPY_ALL; USER=jsmith
```

The **\$USER** environment variable will be set to `jsmith`.

However, if you specify:

```
environment = USER=jsmith; COPY_ALL
```

The **\$USER** environment variable is not set to `jsmith`. Instead, the **\$USER** environment variable of the user ID on the executing node is set.

If the **executable** keyword is not specified, the job command file is run as a shell script. In this case, LoadLeveler initializes the environment as described previously and then starts the **shell** command. Any environment variable set during shell startup overrides values initialized by LoadLeveler.

Examples:

- This example illustrates how to specify that LoadLeveler is to copy all the environment variables from your shell except for `env2`:

```
environment = COPY_ALL; !env2;
```

- This example illustrates how LoadLeveler processes the string you specify with *var*: If you specify the following:

```
environment = env3 = "quoted string"; env4 = imbedded blanks;
```

LoadLeveler uses these values:

- For `env3`: **"quoted string"**
- For `env4`: **imbedded blanks**

error

Specifies the name of the file to use as standard error (`stderr`) when your job step runs.

Syntax:

```
error = filename
```

Default value: If you do not specify a value for this keyword, the file `/dev/null` is used.

Example:

```
error = $(jobid).$(stepid).err
```

executable

Identifies the name of the program to run, which can be a shell script or a binary. For parallel jobs, **executable** must be the parallel job launcher (POE or mpirun), or the name of a program that invokes the parallel job launcher.

Note that the **executable** statement automatically sets the **\$(base_executable)** variable, which is the file name of the executable without the directory component. See Example 2 in topic Examples: Job command files for an example of using the **\$(base_executable)** variable.

Syntax:

executable = *name*

Default value: If you do not include this keyword, then it will default to the job command file that is being submitted, and LoadLeveler will assume that the file is a valid shell script.

Examples:

- # @ executable = a.out
- # @ executable = /usr/bin/poe (for POE jobs)

file_limit

Specifies the hard limit, soft limit, or both limits for the size of a file. This limit is a per process limit.

Syntax:

file_limit = *hardlimit,softlimit*

This keyword accepts both 32-bit and 64-bit integer values.

Default value: No default value is set.

Example:

file_limit = 100pb,50tb

For more information about the values and units you can use with this keyword, and how limits are enforced, see “Using limit keywords” on page 83.

group

Specifies the LoadLeveler group.

Syntax:

group = *group_name*

Default value: If you do not specify a value for this keyword, LoadLeveler uses the default group, **No_Group**.

Example:

group = my_group_name

hold

Specifies whether you want to place a hold on your job step when you submit it. There are three types of holds:

user Specifies user hold

Job command file reference

system

Specifies system hold

usersys

Specifies user and system hold

To remove the hold on the job, you can use either the GUI or the **llhold -r** command.

Syntax:

hold = user | system | usersys

Default value: No default is set, which means that no hold is requested.

Example: To put a user hold on a job, the keyword statement would be:

```
hold = user
```

image_size

Specifies the maximum virtual image size to which your program will grow during execution. LoadLeveler tries to execute your job steps on a machine that has enough resources to support executing and checkpointing your job step. If your job command file has multiple job steps, the job steps will not necessarily run on the same machine, unless you explicitly request that they do.

If you underestimate the image size of your job step, your job step may crash due to the inability to acquire more address space. If you overestimate the image size, LoadLeveler may have difficulty finding machines that have the required resources.

Syntax:

image_size = number

where *number* must be a positive integer. This keyword accepts both 32-bit and 64-bit integer values. If you do not specify the units associated with this keyword, LoadLeveler uses the default unit, which is kilobytes. For a list of allowable units, see the resources keyword description.

Default value: If you do not specify the image size of your job command file, the image size is that of the executable.

Example: To set an image size of 11 KB, the keyword statement would be:

```
image_size = 11
```

initialdir

Specifies the path name of the directory to use as the initial working directory during execution of the job step. File names mentioned in the command file which do not begin with a slash (/) are relative to the initial directory. The initial directory must exist on the submitting machine as well as on the machine where the job runs.

Syntax:

initialdir = pathname

Note: When operating in a multicluster environment, access to **initialdir** will be verified on the cluster selected to run the job. If access to **initialdir** fails, the submission or move job will fail.

Default value: If you do not specify a value for this keyword, the initial directory is the current working directory at the time you submitted the job.

Example:

```
initialdir = /var/home/mike/ll_work
```

input

Specifies the name of the file to use as standard input (stdin) when your job step runs.

Syntax:

```
input = filename
```

Default value: If you do not specify an input file, LoadLeveler uses the file **/dev/null**

Example:

```
input = input.$(process)
```

job_cpu_limit

Specifies the hard limit, soft limit, or both limits for the CPU time used by all processes of a serial job step. For example, if a job step runs as multiple processes, the total CPU time consumed by all processes is added and controlled by this limit.

For parallel job steps, LoadLeveler enforces these limits differently. Parallel job steps usually have tasks running on several different nodes and each task can have several processes associated with it. In addition, the parallel tasks running on a node are descendants of a **LoadL_starter** process. Therefore, if you specify a hard or soft CPU time limit of S seconds and if a **LoadL_starter** has N tasks running under it, then all tasks associated with that **LoadL_starter** will be terminated if the total CPU time of the **LoadL_starter** process and its children is greater than S*N seconds.

If several **LoadL_starter** processes are involved in running a parallel job step, then LoadLeveler enforces the limits associated with the **job_cpu_limit** keyword independently for each **LoadL_starter**. LoadLeveler determines how often to check the **job_cpu_limit** by looking at the values for **JOB_LIMIT_POLICY** and **JOB_ACCT_Q_POLICY**. The smaller value associated with these two configuration keywords sets the interval for checking the **job_cpu_limit**. For more information on **JOB_LIMIT_POLICY** and **JOB_ACCT_Q_POLICY** see “Collecting job resource data on serial and parallel jobs” on page 58.

Syntax:

```
job_cpu_limit = hardlimit,softlimit
```

Default value: No default is set.

Job command file reference

Example:

```
job_cpu_limit = 12:56,12:50
```

For more information about the values and units you can use with this keyword, and how limits are enforced, see “Using limit keywords” on page 83.

job_name

Specifies the name of the job. This keyword must be specified in the first job step. If it is specified in other job steps in the job command file, it is ignored.

The job_name only appears in the long reports of the **llq**, **llstatus**, and **llsummary** commands, and in mail related to the job.

Syntax:

```
job_name = job_name
```

You can name the job using any combination of letters, numbers, or both.

Default value: No default value is set.

Example:

```
job_name = my_first_job
```

job_type

Specifies the type of job step to process.

Syntax:

```
job_type = serial | parallel | bluegene | MPICH
```

Default value: serial

large_page

Specifies whether or not a job step requires Large Page support from AIX.

Restriction: Large Page memory is not supported in LoadLeveler for Linux. In this case, specifying **M** would cause the job to never be sent.

Syntax:

```
large_page = value
```

where *value* can be **Y**, **M**, or **N**. **Y** informs LoadLeveler to use Large Page memory, if available, but to otherwise use regular memory. **M** means use of Large Page memory is mandatory.

Default value: N, which means to not use Large Page memory.

Example: To ask LoadLeveler to use Large Page memory for the job step, if available, specify:

```
large_page = Y
```

max_processors

Specifies the maximum number of nodes requested for a parallel job, regardless of the number of processors contained in the node. This keyword is equivalent to the maximum value you specify on the **node** keyword. In any new job command files you create for parallel jobs, you should use the **node** keyword to request nodes/processors. Note that if you specify in a job command file both the **max_processors** keyword and the **node** keyword, the job is not submitted.

This keyword is supported by the LL_DEFAULT scheduler.

Syntax:

max_processors = *number*

Default value: No default is set.

Example:

max_processors = 6

mcm_affinity_options

Specifies the affinity options for a job.

Syntax:

mcm_affinity_options = *affinity_option*

Where *affinity_option* is a blank-delimited list of one, two, or three keywords chosen from the three groupings of keywords in the list that follows. Only one option from each group may be specified.

task affinity options

The following options are task affinity options. These options are mutually exclusive.

mcm_accumulate

Specifying this option tells the Central Manager to accumulate tasks on the same MCM whenever possible.

mcm_distribute

Specifying this option tells the Central Manager to distribute tasks across all available MCMs on a machine.

memory affinity options

The following options are memory affinity options. These options are mutually exclusive.

mcm_mem_none

Specifying this option indicates the job has no memory affinity requirement.

mcm_mem_pref

Specifying this option indicates the job requests memory affinity.

mcm_mem_req

Specifying this option indicates the job requires memory affinity.

adapter affinity options

The following options are adapter affinity options. These options are mutually exclusive.

mcm_sni_none

Specifying this option indicates the job has no adapter affinity requirement.

mcm_sni_pref

Specifying this option indicates the job request adapter affinity.

mcm_sni_req

Specifying this option indicates the job requires adapter affinity.

Your job containing the keyword **mcm_affinity_options** will not be submitted to LoadLeveler unless the **rset** keyword is set to **RSET_MCM_AFFINITY**.

Default value: **mcm_accumulate mcm_mem_req mcm_sni_pre**

Example:

```
mcm_affinity_options = mcm_mem_req mcm_sni_pref mcm_distribute
```

This example shows how to have a job set memory affinity as a requirement, adapter affinity as a preference, and MCM task allocation method as distribute.

min_processors

Specifies the minimum number of nodes requested for a parallel job, regardless of the number of processors contained in the node. This keyword is equivalent to the minimum value you specify on the **node** keyword. In any new job command files you create for parallel jobs, you should use the **node** keyword to request nodes/processors. Note that if you specify in a job command file both the **min_processors** keyword and the **node** keyword, the job is not submitted.

This keyword is supported by the LL_DEFAULT scheduler.

Syntax:

```
min_processors = number
```

Default value: No default is set.

Example:

```
min_processors = 4
```

network

Specifies communication protocols, adapters, and their characteristics. You need to specify this keyword when you want a task of a parallel job step to request a specific adapter that is defined in the LoadLeveler administration file. You do not need to specify this keyword when you want a task to access a shared, default adapter through TCP/IP. (A default adapter is an adapter whose name matches a machine stanza name.)

Note that you cannot specify both the **network** statement and the **Adapter** requirement in a job command file. Also, the value of the **network** keyword applies only to the job step in which you specify the keyword. (That is, this keyword is not inherited by other job steps.)

This keyword is supported by the BACKFILL and API schedulers.

Syntax:

```
network.protocol = type[, usage[, mode[, comm_level[, instances=<number|max> \
[, rcxtblocks=number]]]]]
```

where:

protocol

Specifies the communication protocols that are used with an adapter, and can be the following:

MPI Specifies the message passing interface (MPI). You can specify in a job step both **network.MPI** and **network.LAPI**.

LAPI Specifies the low-level application programming interface (LAPI). You can specify in a job step both **network.MPI** and **network.LAPI**.

LAPI is not supported on LoadLeveler for Linux.

MPI_LAPI

Specifies sharing adapter windows between MPI and LAPI. When you specify **network.MPI_LAPI** in a job step, you cannot specify any other network statements in that job step.

LAPI is not supported on LoadLeveler for Linux.

type

This field is required and specifies one of the following:

adapter_name

The possible values are the names associated with the interface cards installed on a node (for example, en0 and tk1).

network_type

Specifies a **network_type** as specified in the LoadLeveler administration file. The LoadLeveler administrator must specify values used as **network_type** in the adapter stanza of the LoadLeveler administration file using the **network_type** keyword. For example, an installation can define a network type of "switch" to identify adapters on a common network. For more information on specifying **network_type**, see "Defining adapters" on page 80.

sn_single

When used for the HPS switch it specifies that LoadLeveler use a common, single switch network.

sn_all

Specifies that striped communication should be used over all available switch networks. The networks specified must be accessible by all machines selected to run the job. For more information on striping, see "Submitting jobs that use striping" on page 182.

The following are optional and if omitted their position must be specified with a comma:

usage

Specifies whether the adapter can be shared with tasks of other job steps. Possible values are **shared**, which is the default, or **not_shared**. If **not_shared** is specified, LoadLeveler can only guarantee that the adapter will not be shared by other jobs running on the same OSI. If the

adapter is shared by more than one OSI, LoadLeveler can not guarantee that the adapter is not shared with jobs running on a different OSI.

mode Specifies the communication subsystem mode used by the communication protocol that you specify, and can be either **IP** (Internet Protocol), which is the default, or **US** (User Space). Note that each instance of the **US** mode requested by a task running on the High Performance Switch (HPS) requires an adapter window. For example, if a task requests both the MPI and LAPI protocols such that both protocol instances require **US** mode, two adapter windows will be used.

comm_level

Note: This keyword is obsolete and will be ignored, however it is being retained for compatibility and because the parameters in the network statement are positional.

The **comm_level** keyword should be used to suggest the amount of inter-task communication that users *expect* to occur in their parallel jobs. This suggestion is used to allocate adapter device resources. Specifying a level that is higher than what the job actually needs will not speed up communication, but may make it harder to schedule a job (because it requires more resources). The **comm_level** keyword can only be specified with **US** mode. The three communication levels are:

LOW Implies that minimal inter-task communication will occur.

AVERAGE

This is the default value. Unless you know the specific communication characteristics of your job, the best way to determine the **comm_level** is through trial-and-error.

HIGH Implies that a great deal of inter-task communication will occur.

instances=<number | max>

If **instances** is specified as a number, it indicates the number of parallel communication paths made available to the protocol on each network. The number actually used will depend on the implementation of the protocol subsystem. If **instances** is specified by **max**, the actual value used is determined by the **MAX_PROTOCOL_INSTANCES** for the class to which the job is submitted. The default value for **instances** is 1.

For the best performance set **MAX_PROTOCOL_INSTANCES** so that the communication subsystem uses every available adapter before it reuses any of the adapters.

rcxtblocks=number

Integer value specifying the number of user rCxt blocks requested for each window used by the associated protocol. The values of this keyword are not inherited between steps in a multistep job.

Note: Use of this keyword will prevent adapters from the SP Switch2 family from being used by the job.

Default value: If you do not specify the **network** keyword, LoadLeveler allows the task to access a shared, default adapter through TCP/IP. The default adapter is the adapter associated with the machine name.

Examples:

- **Example 1:** To use the MPI protocol with an adapter in User Space mode without sharing the adapter, enter the following:
`network.MPI = sn_single,not_shared,US,HIGH`
- **Example 2:** To use the MPI protocol with a shared adapter in IP mode, enter the following:
`network.MPI = sn_single,,IP`

Because a shared adapter is the default, you do not need to specify **shared**.

- **Example 3:** A communication level can only be specified if User Space mode is also specified:
`network.MPI = sn_single,,US,AVERAGE`

Note that LoadLeveler can ensure that an adapter is dedicated (not shared) if you request the adapter in US mode, since any user who requests a user space adapter must do so using the **network** statement. However, if you request a dedicated adapter in IP mode, the adapter will only be dedicated if all other LoadLeveler users who request this adapter do so using the **network** statement.

node

Specifies the minimum and maximum number of nodes requested by a job step. You must specify at least one of these values. The value of the **node** keyword applies only to the job step in which you specify the keyword. (That is, this keyword is not inherited by other job steps.)

When you use the **node** keyword together with the **total_tasks** keyword, the *min* and *max* values you specify on the **node** keyword must be equal, or you must specify only one value. For example:

```
node = 6
total_tasks = 12
```

This keyword is supported by the BACKFILL and API schedulers.

Syntax:

node = [*min*] [,*max*]

where:

- min* Specifies the minimum number of nodes requested by the job step.
- max* Specifies the maximum number of nodes requested by the job step. The maximum number of nodes a job step can request is limited by the **max_node** keyword in the administration file (provided this keyword is specified).

Job command file reference

That is, the maximum must be less than or equal to any **max_node** value specified in a user, group, or class stanza.

Default value: The default value for *min* is 1; the default value for *max* is the *min* value for this keyword.

Example: To specify a range of six to twelve nodes, enter the following:

```
node = 6,12
```

To specify a maximum of seventeen nodes, enter the following:

```
node = ,17
```

For information on specifying the number of tasks you want to run on a node, see “Task-assignment considerations” on page 180,370, and 371.

node_usage

Specifies whether this job step shares nodes with other job steps.

This keyword is supported by the BACKFILL and API schedulers.

Syntax:

```
node_usage = shared | not_shared
```

where:

shared

Specifies that nodes can be shared with other tasks of other job steps.

not_shared

Specifies that nodes are not shared. No other job steps are scheduled on this node.

Default value: shared

notification

Specifies when the user specified in the **notify_user** keyword is sent mail.

Syntax:

```
notification = always|error|start|never|complete
```

where:

always

Notify the user when the job begins, ends, or if it incurs error conditions.

error Notify the user only if the job fails.

start Notify the user only when the job begins.

never Never notify the user.

complete

Notify the user only when the job ends.

Default value: complete

Examples:

- If you want to be notified with mail only when your job step completes, your notification keyword would be:

notification = complete

- When a LoadLeveler job ends, you may receive mail notification indicating the job exit status. For example, you could get the following mail message:

```
Your LoadLeveler job
myjob1
exited with status 4.
```

The return code 4 is from the user's job. LoadLeveler retrieves the return code and returns it in the mail message, but it is not a LoadLeveler return code.

notify_user

Specifies the user to whom mail is sent based on the **notification** keyword.

Syntax:

notify_user = *userID*

Default value: The default is the submitting user at the submitting machine.

Example: If you are the job step owner but you want a coworker whose name and user ID is **bob**, to receive mail regarding the job step, your notify keyword would be:

```
notify_user = bob@mailserv.pok.ibm.com
```

output

Specifies the name of the file to use as standard output (stdout) when your job step runs.

Syntax:

output = *filename*

Default value: If you do not specify this keyword, LoadLeveler uses the file **/dev/null**

Example:

```
output = out.%(jobid)
```

preferences

Specifies the characteristics that you prefer be available on the machine that executes the job steps. LoadLeveler attempts to run the job steps on machines that meet your preferences. If such a machine is not available, LoadLeveler will then assign machines that meet only your requirements.

The values you can specify in a **preferences** statement are the same values you can specify in a **requirements** statement, with the exception of the **Adapter** requirement.

Syntax:

preferences = *Boolean_expression*

Default value: No default preferences are set.

Examples:

Job command file reference

```
preferences = (Memory <=16) && (Arch == "R6000")
preferences = Memory >= 64
```

queue

Places one copy of the job step in the queue. This statement is required. The **queue** statement essentially marks the end of the job step. Note that you can specify statements between **queue** statements.

Syntax:

queue

requirements

Specifies the requirements which a machine in the LoadLeveler cluster must meet to execute any job steps. You can specify multiple requirements on a single requirements statement.

Syntax:

requirements = *Boolean_expression*

When strings are used as part of a Boolean expression that must be enclosed in double quotes. Sample requirement statements are included following the descriptions of the supported requirements, which are:

Adapter

Specifies the predefined type of network you want to use to run a parallel job step. In any new job command files you create, you should use the **network** keyword to request adapters and types of networks.

It is also the way to specify when running with the default LoadLeveler scheduler. When using the default scheduler, the **Adapter** requirement is specified as the physical name of the device, such as *en0*.

This keyword is supported by the LL_DEFAULT and BACKFILL schedulers.

Note that you cannot specify both the **Adapter** requirement and the **network** statement in a job command file.

For the BACKFILL scheduler you can use the predefined network types. The predefined network types are:

ethernet

Refers to Ethernet.

fddi Refers to Fiber Distributed Data Interface (FDDI).

tokenring

Refers to Token Ring.

fcs Refers to Fiber Channel Standards.

Note that LoadLeveler converts the network types to the **network** statement.

Arch

Specifies the machine architecture on which you want your job step to run. It describes the particular kind of platform for which your executable has been compiled.

Connectivity

Connectivity is the ratio of the number of active switch adapters on a node to the total number of switch adapters on the node. The value ranges from 0.0 (all switch adapters are down) to 1.0 (all switch adapters are active). A node with no switch adapters has a connectivity of 0.0 . Connectivity can be used in a **MACHPRIO** expression to favor nodes that do not have any down switch adapters or in a job **REQUIREMENTS** statement to require only nodes with a certain connectivity.

Disk

Specifies the amount of disk space in kilobytes you believe is required in the LoadLeveler **execute** directory to run the job step.

Note: The Disk variable in an expression associated with the **requirements** and **preferences** keywords are 64-bit integers.

Feature

Specifies the name of a feature defined on a machine where you want your job step to run. Be sure to specify a feature in the same way in which the feature is specified in the configuration file. To find out what features are available, use the **llstatus** command.

LargePageMemory

Specifies the amount, in megabytes, of Large Page Memory required to run the job.

Note: The Memory variable in an expression associated with the **requirements** and **preferences** keywords are 64-bit integers.

LL_Version

Specifies the LoadLeveler version, in dotted decimal format, on which you want your job step to run. For example, LoadLeveler Version 3 Release 4 (with no modification levels) is written as 3.4.0.0.

Machine

Specifies the names of machines on which you want the job step to run. Be sure to specify a machine in the same way in which it is specified in the machine configuration file.

If you have a mixed LoadLeveler cluster where the OpSys values of the machines may be either AIX53 or AIX52, using the **requirements** keyword to specify a Machine requirement may result in an expression that always evaluates to false. If the OpSys value of the submitting machine is AIX53, the **llsubmit** command automatically adds (OpSys == "AIX53") to the other job requirements unless an OpSys requirement has already been explicitly specified. This behavior means that the specification:

```
requirements = (Machine == "jupiter")
```

automatically becomes:

```
requirements = (Machine == "jupiter") && (OpSys == "AIX53")
```

This requirement cannot be satisfied unless the OpSys value of "jupiter" is also AIX53. In this case, a better strategy would be to use an expression such as:

```
requirements =  
  (Machine == "jupiter") && ((OpSys == "AIX52") || (OpSys == "AIX53"))
```

Memory

Specifies the amount, in megabytes, of regular physical memory required in the machine where you want your job step to run.

Note: The Memory variable in an expression associated with the **requirements** and **preferences** keywords are 64-bit integers.

OpSys

Specifies the operating system on the machine where you want your job step to run. It describes the particular kind of platform for which your executable has been compiled.

Pool

Specifies the number of a pool where you want your job step to run.

TotalMemory

Specifies the amount, in megabytes, of regular physical memory and Large Page memory required in the machine where you want your job step to run.

Note: The Memory variable in an expression associated with the **requirements** and **preferences** keywords are 64-bit integers.

Default value: No default requirements are set.

Examples:

- **Example 1:** To specify a memory requirement and a machine architecture requirement, enter:

```
requirements = (Memory >=16) && (Arch == "R6000")
```

- **Example 2:** To specify that your job requires multiple machines for a parallel job, enter:

```
requirements = (Machine == { "116" "115" "110" })
```

- **Example 3:** You can set a machine equal to a job step name. This setting means that you want the job step to run on the same machine on which the previous job step ran. For example:

```
requirements = (Machine == machine.step_name)
```

Where *step_name* is a step name previously defined in the job command file. The use of **Machine == machine.step_name** is limited to serial jobs.

Example:

```
# @ step_name      = step1  
# @ executable     = c1  
# @ output         = $(executable).$(jobid).$(step_name).out  
# @ queue  
# @ step_name      = step2  
# @ dependency     = (step1 == 0)
```

```
# @ requirements = (Machine == machine.step1)
# @ executable   = c2
# @ output       = $(executable).$(jobid).$(step_name).out
# @ queue
```

- **Example 4:** To specify a requirement for a specific pool number, enter:

```
requirements = (Pool == 7)
```

- **Example 5:** To specify a requirement that the job runs on LoadLeveler Version 3 Release 4 or any follow-on release, enter:

```
requirements = (LL_Version >= "3.4")
```

Note that the statement **requirements = (LL_Version == "3.4")** matches only the value 3.4.0.0.

- **Example 6:** To specify the job runs if all switch connections are up, enter:

```
# @ requirements = (Connectivity == 1.0)
```

To specify the job runs if at least half of the switch connections are up, enter:

```
# @ requirements = (Connectivity >= .5)
```

To specify the job runs if there is at least some connectivity, enter:

```
# @ requirements = (Connectivity > 0)
```

resources

Specifies quantities of the consumable resources consumed by each task of a job step. The resources may be machine resources or floating resources.

Syntax:

```
resources=name(count) name(count) ... name(count)
```

where *name(count)* is one of the following:

- An administrator defined name and count
- **ConsumableCpus**(*count*)
- **ConsumableMemory**(*count units*)
- **ConsumableVirtualMemory**(*count units*)

ConsumableMemory and **ConsumableVirtualMemory** are the only two consumable resources that can be specified with both a count and units.

The count for each specified resource must be an integer greater than or equal to zero, except for the following instances in which the integer must be greater than zero:

- **ConsumableMemory**
- **ConsumableVirtualMemory**
- **ConsumableCpus** when the enforcement policy is *hard* or *soft*

ConsumableCpus can have a value of zero when the administrator has not requested that consumable resources be enforced, or when the enforcement policy is *shares*.

When you set **ConsumableCpus** to zero, the meaning varies depending on whether use is being enforced. With no enforcement,

zero means the job is requesting a negligible amount of CPU. With an enforcement policy of *shares*, it means the job is requesting a tiny percentage of available shares.

If the count is not valid then LoadLeveler will issue a message and the job will not be submitted. The allowable units are those normally used with LoadLeveler data limits:

```
b bytes
w words (4 bytes)
kb kilobytes (2**10 bytes)
kw kilowords (2**12 bytes)
mb megabytes (2**20 bytes)
mw megawords (2**22 bytes)
gb gigabytes (2**30 bytes)
gw gigawords (2**32 bytes)
tb terabytes (2**40 bytes)
tw terawords (2**42 bytes)
pb petabytes (2**50 bytes)
pw petawords (2**52 bytes)
eb exabytes (2**60 bytes)
ew exawords (2**62 bytes)
```

The **resources** keyword accepts both 32-bit and 64-bit integer values. These values, however, are assigned to the consumable resources defined in the **resources** keyword and not to the keyword itself.

ConsumableMemory and **ConsumableVirtualMemory** values are stored in mb (megabytes) and rounded up. Therefore, the smallest amount of **ConsumableMemory** or **ConsumableVirtualMemory** which you can request is one megabyte. If no units are specified, then megabytes are assumed. However, **image_size** units are in kilobytes. Resources defined here that are not in the **SCHEDULE_BY_RESOURCES** list in the global configuration file will not affect the scheduling of the job.

When resource usage and resource submission is enforced, the **resources** keyword must specify requirements for the resources defined in the **ENFORCE_RESOURCE_USAGE** keyword.

Default value: If the **resources** keyword is not specified in the job step, then the **default_resources** (if any) defined in the administration file for the class will be used for each task of the job step.

restart

Specifies whether LoadLeveler considers a job to be “restartable.”

Syntax:

```
restart = yes|no
```

If **restart=yes**, and the job is vacated from its executing machine before completing, the central manager requeues the job. It can start running again when a machine on which it can run becomes available. If **restart=no**, a vacated job is canceled rather than requeued.

Note that jobs which are checkpointable (**checkpoint = yes | interval**) are always considered "restartable".

Default value: yes

restart_from_ckpt

Indicates whether a job step is to be restarted from a checkpoint file.

Restriction: This keyword is ignored by LoadLeveler for Linux.

Syntax:

restart_from_ckpt = yes | no

where:

yes Indicates LoadLeveler will restart the job step from the checkpoint file specified by the job command file keyword **ckpt_file**. The location of the **ckpt_file** will be determined by the values of the job command file keyword **ckpt_file** or **ckpt_dir**, the administrator defined location or the default location. See "Naming checkpoint files and directories" on page 134 for a description of how the checkpoint directory location is determined. This value is valid only when a job is being restarted from a previous checkpoint.

no The job step will be started from the beginning, not from the checkpoint file.

Default value: no

If you specify an invalid value for this keyword, the system generates an error message and the job is not submitted.

restart_on_same_nodes

Indicates that a job step is to be restarted on the same set of nodes that it was run on previously. This keyword applies only to restarting a job step after a vacate (this condition is when the job step is terminated and then returned to the LoadLeveler job queue).

Syntax:

restart_on_same_nodes = yes | no

where:

yes Indicates that the job step is to be restarted on the same set of nodes on which it had run.

no Indicates that it is not required to restart a vacated job on the same nodes.

Default value: no

rset

This keyword indicates that the job tasks need to be attached to RSets with CPUs selected by different LoadLeveler scheduling algorithms or RSets created by users.

Syntax:

Job command file reference

rset = *value*

Value can be one of the following keywords or a user defined RSet name.

RSET_CONSUMABLE_CPUS

Specifying this option indicates that the job needs to be scheduled to machines where **RSET_SUPPORT** is set to **RSET_CONSUMABLE_CPUS** and the tasks need to be attached to RSets with a number of CPUs equal to the number of **ConsumableCPUs** requested by the job without two tasks sharing a CPU.

RSET_MCM_AFFINITY

Specifying this value requests affinity scheduling with memory affinity as a requirement, adapter affinity as a preference, and the task MCM allocation method set to accumulate. The affinity options may be changed from these defaults by using the **mcm_affinity_options** keyword.

When anything other than these values are specified, LoadLeveler considers the value to be a user defined RSet name and schedules the job to nodes with **RSET_SUPPORT** set to **RSET_USER_DEFINED**.

Default value: No default is set.

Example:

```
rset=RSET_MCM_AFFINITY
```

This example shows how to request affinity scheduling for a job.

shell

Specifies the name of the shell to use for the job step.

Syntax:

```
shell = name
```

Default value: If you do not specify a value for this keyword, LoadLeveler uses the shell used in the owner's password file entry. If none is specified, LoadLeveler uses `/bin/sh`

Example: If you want to use the Korn shell, the shell keyword would be:

```
shell = /bin/ksh
```

smt

Specifies the parallel job steps that require dynamic simultaneous multithreading (SMT) to be turned on or off before they are run.

Syntax:

```
smt = yes | no
```

where **yes** indicates that LoadLeveler will invoke AIX service to turn on SMT on the node. This will be true for all the nodes where the parallel job will be running.

Default value: no.

Examples:

smt = yes

stack_limit

Specifies the hard limit, soft limit, or both limits for the size of the stack that is created.

Syntax:

stack_limit = *hardlimit,softlimit*

This keyword accepts both 32-bit and 64-bit integer values.

Default value: No default is set.

Example:

stack_limit = 120000,100000

Because no units have been specified in this example, LoadLeveler assumes that the figure represents a number of bytes.

For more information about the values and units you can use with this keyword, and how limits are enforced, see “Using limit keywords” on page 83.

startdate

Specifies when you want to run the job step.

Syntax:

startdate = *date time*

date is expressed as *MM/DD/YYYY*, and *time* is expressed as *HH:mm(:ss)*.

Default value: If you do not specify a start date, LoadLeveler uses the current date and time.

Example: If you want the job to run on August 28th, 2010 at 1:30 PM, issue:

startdate = 08/28/2010 13:30

If you specify a start date that is in the future, your job is kept in the Deferred state until that start date.

step_name

Specifies the name of the job step. You can name the job step using any combination of letters, numbers, underscores (`_`) and periods (`.`). You cannot, however, name it T or F, or use a number in the first position of the step name. The step name you use must be unique and can be used only once.

Syntax:

step_name = *step_name*

Default value: If you don't specify a step name, by default the first job step is named the character string "0", the second is named the character string "1", and so on.

Example:

```
step_name = step_3
```

task_geometry

The **task_geometry** keyword allows you to group tasks of a parallel job step to run together on the same node. Although **task_geometry** allows for a great deal of flexibility in how tasks are grouped, you cannot specify the particular nodes that these groups run on; the scheduler will decide which nodes will run the specified groupings.

This keyword is supported by the BACKFILL and API schedulers.

Syntax:

```
task_geometry={(task id,task id,...)(task id,task id, ...) ... }
```

Default value: No default value is set.

Example: A job with 6 tasks will run on 4 different nodes:

```
task_geometry={(0,1) (3) (5,4) (2)}
```

Each number in this example represents a task ID in a job, each set of parenthesis contains the task IDs assigned to one node. The entire range of tasks specified must begin with 0, and must be complete; no number can be skipped (the largest task id number should end up being the value that is one less than the total number of tasks). The entire statement following the keyword must be enclosed in braces, and each grouping of nodes must be enclosed in parenthesis. Commas can only appear between task IDs, and spaces can only appear between nodes and task IDs.

The **task_geometry** keyword cannot be specified under **any** of the following conditions:

- The step is serial.
- **job_type** is anything other than **parallel**
- Any of the following keywords are specified:
 - **tasks_per_node**
 - **total_tasks**
 - **node**
 - **min_processors**
 - **max_processors**
 - **blocking**

For more information, see "Task-assignment considerations" on page 180.

tasks_per_node

Specifies the number of tasks of a parallel job you want to run per node. Use this keyword together with the **node** keyword. The

value you specify on the **node** keyword can be a range or a single value. If the node keyword is not specified, then the default value is one node.

The maximum number of tasks a job step can request is limited by the **total_tasks** keyword in the administration file (provided this keyword is specified). That is, the maximum must be less than any **total_tasks** value specified in a user, group, or class stanza.

The value of the **tasks_per_node** keyword applies only to the job step in which you specify the keyword. (That is, this keyword is not inherited by other job steps.)

Also, you cannot specify both the **tasks_per_node** keyword and the **total_tasks** keyword within a job step.

This keyword is supported by the BACKFILL and API schedulers.

Syntax:

tasks_per_node = *number*

where *number* is the number of tasks you want to run per node.

Default value: The default is one task per node.

Example: To specify a range of seven to 14 nodes, with four tasks running on each node, enter the following:

```
node = 7,14
tasks_per_node = 4
```

This job step runs 28 to 56 tasks, depending on the number of nodes allocated to the job step.

total_tasks

Specifies the total number of tasks of a parallel job you want to run on all available nodes. Use this keyword together with the **node** keyword. The value you specify on the **node** keyword must be a single value rather than a range of values. If the node keyword is not specified, then the default value is one node.

The maximum number of tasks a job step can request is limited by the **total_tasks** keyword in the administration file (provided this keyword is specified). That is, the maximum must be less than any **total_tasks** value specified in a user, group, or class stanza. The value of the **total_tasks** keyword applies only to the job step in which you specify the keyword. (That is, this keyword is not inherited by other job steps.) Also, you cannot specify both the **total_tasks** keyword and the **tasks_per_node** keyword within a job step.

If you specify an unequal distribution of tasks per node, LoadLeveler allocates the tasks on the nodes in a round-robin fashion. For example, if you have three nodes and five tasks, two tasks run on the first two nodes and one task runs on the third node.

This keyword is supported by the BACKFILL and API schedulers.

Syntax:

total_tasks = *number*

Job command file reference

Where *number* is the total number of tasks you want to run.

Default value: No default is set.

Example: To run two tasks on each of 12 available nodes for a total of 24 tasks, enter the following:

```
node = 12
total_tasks = 24
```

user_priority

Sets the initial priority of your job step. Priority only affects your job steps. It orders job steps you submitted with respect to other job steps submitted by you, not with respect to job steps submitted by other users.

Syntax:

user_priority = *number*

Where *number* is a number between 0 and 100, inclusive. A higher number indicates the job step will be selected before a job step with a lower number. Note that this keyword is not the UNIX *nice* priority.

This priority guarantees the order the jobs are considered for dispatch. It does not guarantee the order in which they will run.

Default value: The default priority is 50.

wall_clock_limit

Sets the hard limit, soft limit, or both limits for the elapsed time for which a job can run. In computing the elapsed time for a job, LoadLeveler considers the start time to be the time the job is dispatched.

If you are running the BACKFILL scheduler, you must either set a wall clock limit in the job command file or the administrator must define a wall clock limit value for the class to which a job is assigned. In most cases, this wall clock limit value should not be **unlimited**. For more information, see “Choosing a scheduler” on page 42.

Syntax:

wall_clock_limit = *hardlimit,softlimit*

An example is:

```
wall_clock_limit = 5:00,4:30
```

For more information about the values and units you can use with this keyword, and how limits are enforced, see “Using limit keywords” on page 83.

Job command file variables

LoadLeveler has several variables you can use in a job command file. These variables are useful for distinguishing between output and error files.

You can refer to variables in mixed case, but you must specify them using the following syntax:

`$(variable_name)`

The following variables are available to you:

\$(domain)

The domain of the host from which the job was submitted.

\$(home)

The home directory for the user on the cluster selected to run the job. Since the user may differ from the submitting user when a remote cluster is selected to run the job and user mapping is used, so may the home directory differ.

\$(host)

The hostname of the machine from which the job was submitted. In a job command file, the **\$(host)** variable and the **\$(hostname)** variable are equivalent.

\$(jobid)

The sequential number assigned to this job by the Schedd daemon. The **\$(jobid)** variable and the **\$(cluster)** variable are equivalent.

\$(schedd_host)

The hostname of the scheduling machine.

\$(schedd_hostname)

The hostname and domain name of the scheduling machine.

\$(stepid)

The sequential number assigned to this job step when multiple queue statements are used with the job command file. The **\$(stepid)** variable and the **\$(process)** variable are equivalent.

\$(user)

The user name on the cluster selected to run the job. This might be a different user name than the user name who submitted the job. It is possible for the value of this variable to differ from the submitting user name when a remote cluster is selected to run the job and user name mapping is being used.

In addition, the following keywords are also available as variables. However, you must define them in the job command file. These keywords are described in detail in “Job command file keyword descriptions” on page 336.

- **\$(executable)**
- **\$(class)**
- **\$(comment)**
- **\$(job_name)**
- **\$(step_name)**

Note that for the **\$(comment)** variable, the keyword definition must be a single string with no blanks. Also, the **executable** statement automatically sets the **\$(base_executable)** variable, which is the file name of the executable without the directory component. See Figure 22 on page 169 for an example of using the **\$(base_executable)** variable.

Run-time environment variables

The following environment variables are set by LoadLeveler for all jobs. These environment variables are also set before running prolog and epilog programs. For more information on prolog and epilog programs, see "Writing prolog and epilog programs" on page 70.

LOADLBATCH

Set to **yes** to indicate the job is running under LoadLeveler.

LOADLBATCH

Set to **yes** to indicate the job is running under LoadLeveler.

LOADL_ACTIVE

The LoadLeveler version.

LOADL_CKPT_FILE

Identifies the directory and file name for checkpointing files. LoadLeveler will only set this environmental variable if checkpointing is enabled.

LOADL_HOSTFILE

| Specifies the the full path name of the file that contains the host names
| assigned to all the tasks of the step. This environment variable is available
| only when the **job_type** is set to **MPICH**. This file is created in the execute
| directory and is deleted once the step has completed. The host names are
| stored in the file as one host name per line. The base name of this file is
| **step_hosts.step_id**.

LOADL_JOB_NAME

The three part job identifier.

LOADL_PID

The process ID of the starter process.

LOADL_PROCESSOR_LIST

A blank-delimited list of hostnames allocated for the step. This environment variable is limited to 128 hostnames. If the value is greater than the 128 limit, the environment variable is not set.

LOADL_STARTD_PORT

The port number where the startd daemon runs.

LOADL_STEP_ACCT

The account number of the job step owner.

LOADL_STEP_ARGS

Any arguments passed by the job step.

LOADL_STEP_CLASS

The job class for serial jobs.

LOADL_STEP_COMMAND

The name of the executable (or the name of the job command file if the job command file is the executable).

LOADL_STEP_ERR

The file used for standard error messages (stderr).

LOADL_STEP_GROUP

The UNIX group name of the job step owner.

LOADL_STEP_ID

The job step ID.

LOADL_STEP_IN

The file used for standard input (stdin).

LOADL_STEP_INITDIR

The initial working directory.

LOADL_STEP_NAME

The name of the job step.

LOADL_STEP_NICE

The UNIX *nice* value of the job step. This value is determined by the **nice** keyword in the class stanza. For more information, see “Defining classes” on page 83.

LOADL_STEP_OUT

The file used for standard output (stdout).

LOADL_STEP_OWNER

The job step owner.

LOADL_STEP_TYPE

The job type (SERIAL or PARALLEL)

LOADL_TOTAL_TASKS

Specifies the total number of tasks of the MPICH job step. This variable is available only when the `job_type` is set to MPICH.

Job command file examples

The following job command file creates an output file called **stance.78.out**, where `stance` is the host and `78` is the job ID.

```
# @ executable = my_job
# @ arguments  = 5
# @ output     = $(host).$(jobid).out
# @ queue
```

The following job command file creates an **output** file called **computel.step1.March05**.

```
# @ comment    = March05
# @ job_name   = computel
# @ step_name  = step1
# @ executable = my_job
# @ output     = $(job_name).$(step_name).$(comment)
# @ queue
```

For additional information, see “Examples: Job command files” on page 167.

Job command file variables

Chapter 15. Graphical user interface (GUI) reference

The LoadLeveler GUI provides an interface for users and administrators similar in function to the LoadLeveler command line. For more information Chapter 7, “Using LoadLeveler’s GUI to perform administrator tasks,” on page 155 or Chapter 11, “Using LoadLeveler’s GUI to build, submit, and manage jobs,” on page 217.

If this is the first time you are using a Motif-based GUI, you should refer to the appropriate Motif documentation for general GUI information.

In “Customizing the GUI” on page 380 you will also find information on customizing the GUI by:

- Modifying windows and buttons
- Creating pull-down menus
- Customizing window fields
- Modifying help panels
- Setting up administrative tasks

Note: LoadLeveler provides two types of graphical user interfaces. One interface is for users whose machines interact fully with LoadLeveler. The second interface is available to users of submit-only machines that participate on a limited basis with LoadLeveler.

Starting the GUI

To start the GUI, check your PATH variable to ensure that it is pointing to the LoadLeveler binaries. Also, check to see that your DISPLAY variable is set to your display. Then, type one of the following to start the GUI in the background:

- **xloadl_so &** (if you are running a submit-only machine)
- **xloadl &** (for all other users)

Note: When you invoke the GUI in a multicluster environment, an additional window appears. This window allows you to start additional local instances of **xloadl** or **xloadl_so** for each remote cluster present in your multicluster environment. These instances of **xloadl** are distinguished through the instance window titles:

- The **xloadl** instances for remote clusters have titles of the form *local_cluster_name→remote_cluster_name*, for instance, **MY_C2→MY_C3**, where MY_C2 and MY_C3 are cluster names (local and remote, respectively).
- The **xloadl** instance for the local cluster has a title of the form *local_cluster_name*, for instance, **MY_C2**, where MY_C2 is the name of the local cluster.

Specifying GUI options

In general, you can specify GUI options in any of the following ways:

- Within the GUI using menu selections
- On the **xloadl** (or **xloadl_so**) command line. Enter **xloadl -h** or **xloadl_so -h** to see a list of the available options.
- In the **Xloadl** file. See “Customizing the GUI” on page 380 for more information.

The LoadLeveler main window

LoadLeveler’s main window has three sub-windows, titled Jobs, Machines, and Messages, as shown in Figure 42. Each of these sub-windows has its own menu bar.

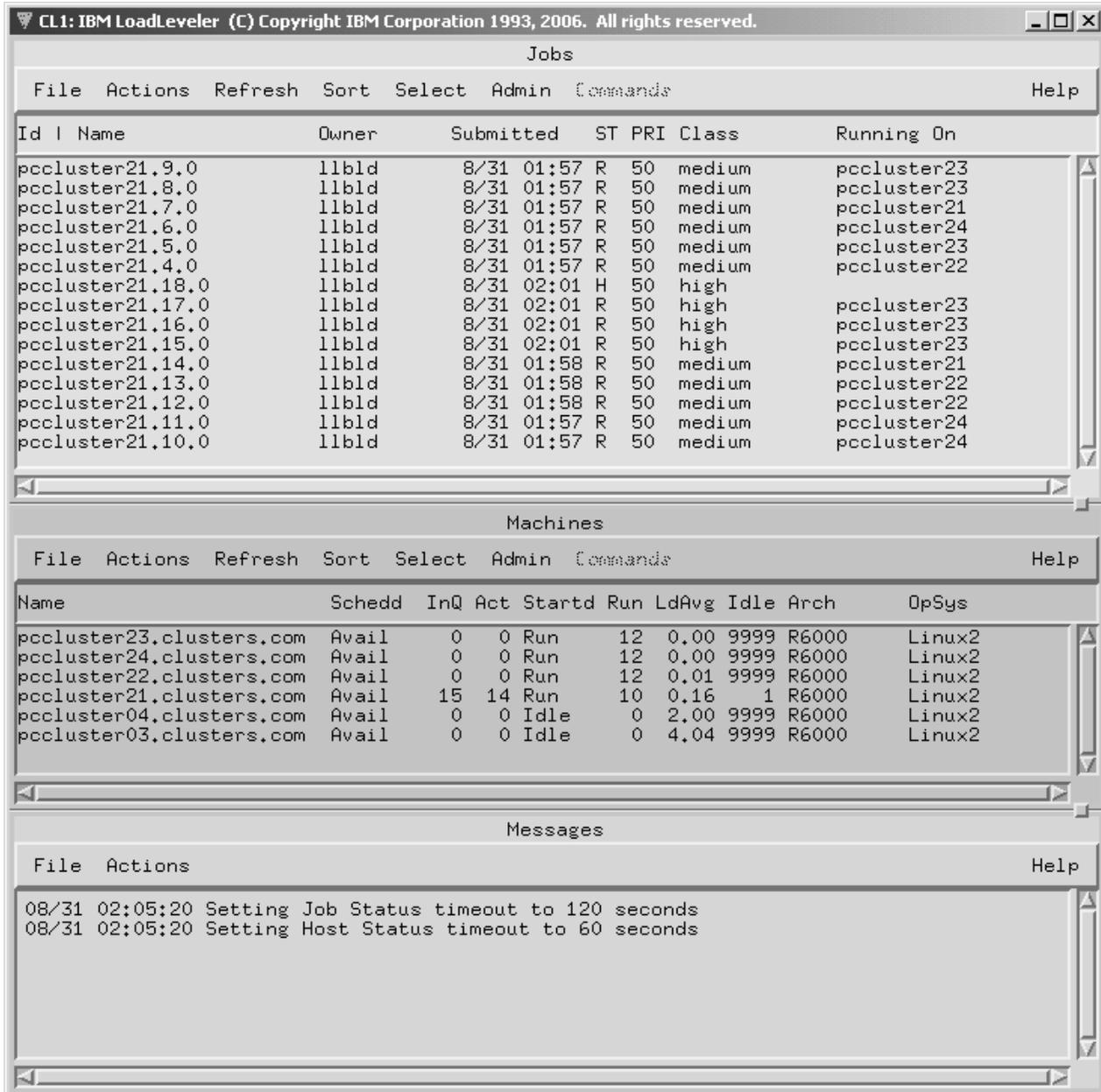


Figure 42. Main window of the LoadLeveler GUI

The menu bar on the Jobs window relates to actions you can perform on jobs. The menu bar on the Machines window relates to actions you can perform on machines. Similarly, the menu bar on the Messages window displays actions you can perform related to LoadLeveler generated messages.

When you select an item from a menu bar, a pull-down menu appears. You can select an item from the pull-down menu to carry out an action or to bring up another pull-down menu originating from the first one.

Getting help using the GUI

You can get help when using the GUI by pressing the Help key. This key is function key 1 (F1) on most keyboards. To receive help on specific parts of the LoadLeveler GUI, click the mouse on the area or field for which you want help and press F1. A help screen appears describing that area. You can also get help by using the Help pull-down menu and the Help push buttons available in pop-up windows.

Before you invoke the GUI, make sure your PATH statement includes the directory containing the LoadLeveler executables. Otherwise, some GUI functions may not work correctly.

Differences between LoadLeveler's GUI and other graphical user interfaces

LoadLeveler's GUI contains many items common to other GUIs. There are, however, some differences that you should be aware of. These differences are:

- Accelerators or mnemonics do not appear on the menu bars.
- Submerged windows do not necessarily rise to the top when refreshed.

GUI typographic conventions

The following typographic conventions are used when describing the way tasks are accomplished using the GUI.

Task step conventions

Each task step includes a user action and a system response. User actions appear in **UPPERCASE BOLDFACE** type and the system response to an action follows a ▲. For example:

SELECT

Refresh → Set Auto Refresh

▲ A window appears.

An action is sometimes represented by itself, for example:

SELECT OK

Selection table and decision table conventions

Some actions require a selection or decision. Selection and decision actions are presented in tables.

Selection tables list all possible selections in the left column of the table.

Table 78 is an example of a selection table:

Table 78. Example of a selection table

To	Do This
Submit a job	Refer to "Submitting a job command file" on page 230.
Cancel a job	Refer to "Canceling a job" on page 233.

Decision tables present a question or series of questions before indicating the action. Table 79 is an example of a decision table and Table 80 on page 380 shows the actions:

Table 79. Decision table

Did the job you submitted complete processing?
--

Table 80. Decision table actions

Decision	Action
Yes	Submit another job.
No	Check the status of the job.

Menu selection conventions

Selections from a menu bar are indicated with an \rightarrow . For example, if a menu bar included an option called **Actions** and **Actions** included an option called **Cancel**, the instructions would read:

SELECT Actions \rightarrow Cancel

64-bit support for the GUI

The LoadLeveler Graphical User Interface (**xloadl** or **xloadl_so**) accepts and displays 64-bit information where appropriate.

Customizing the GUI

You can customize the GUI to suit your needs by overriding the default settings of the LoadLeveler resource variables. For example, you can set the color, initial size, and location of the main window.

This topic tells you how to customize the GUI by modifying either (or both) of the following files:

Xloadl For fully participating machines
Xloadl_so For submit-only machines

If the LoadLeveler administrator has set up these resource files, the files are located in the **/usr/lib/X11/app-defaults** directory. Otherwise, the files are located in the **lib** directory of the LoadLeveler release directory:

- For AIX, in **/usr/lpp/LoadL/full/lib** and **/usr/lpp/LoadL/so/lib**, respectively.
- For Linux, in **/opt/ibmll/LoadL/full/lib** and **/opt/ibmll/LoadL/so/lib**, respectively.

These files contain the default values for the graphical user interface. This topic discusses the syntax of these files, and gives you an overview of some of the resources you can modify.

An administrator with root authority can make changes to the resources for the entire installation by editing the **Xloadl** file. Any user can make local changes by placing the resource names with their new values in the user's **.Xdefaults** file.

Syntax of an Xloadl file

- Comments begin with **!**
- Resource variables may begin with *****
- Colons follow resource variables
- Resource variable values follow colons.

Modifying windows and buttons

All of the windows and buttons that are part of the GUI have certain characteristics in common. For example, they all have a foreground and background color, as well as a size and a location. Each one of these characteristics is represented by a resource variable. For example, the foreground characteristic is

represented by the resource variable **foreground**. In addition, every resource variable has a value associated with it. The values of the resource variable **foreground** are a range of colors.

Before customizing a window, you need to locate the resource variables associated with the desired window. To do this, search for the window identifier in your **Xloadl** file. Table 81 lists the windows and their respective identifiers:

Table 81. Window identifiers in the Xloadl file

Window	Identifier
Account Report Data	reporter
Build a Job	builder
Checkpoint Fields	ckpt
Jobs	job_status
Limits	limits
Machines	machine_status
Messages	message_area
Network	network
Nodes	nodes
Preferences	preferences
Requirements	requirements
Script	script
Submit a Job	submit
Task Geometry	tgeometry

Table 82 lists the resource variables for all the windows and the buttons along with a description of each resource variable. Use the information in this table to modify your graphical user interface by changing the values of desired resource variables. The values of these resource variables depend upon Motif requirements.

Table 82. Resource variables for all the windows and the buttons

Resource Variable	Description
background	The background color of the object
foreground	The foreground color of the object
geometry	The location of the object
height	The height of the object
labelString	The text associated with the object
width	The width of the object

Creating your own pull-down menus

You can add a pull-down menu to both the Jobs window and the Machines window.

To add a pull-down menu to the Jobs window, in the **Xloadl** file:

1. Set **userJobPulldown** to **True**
2. Set **userJob.labelString** to the name of your menu.

Customizing the GUI

3. Fill in the appropriate information for your first menu item, **userJob_Option1**
4. To define more menu items, fill in the appropriate information for **userJob_Option2**, **userJob_Option3**, and so on. You can define up to ten menu items.

For more information, refer to the comments in the **Xloadl** file.

To add a pull-down menu to the Machines window, in the **Xloadl** file:

1. Set **userMachinePulldown** to **True**
2. Set **userMachine.labelString** to the name of your menu.
3. Fill in the appropriate information for your first menu item, **userMachine_Option1**
4. To define more menu items, fill in the appropriate information for **userMachine_Option2**, **userMachine_Option3**, and so on. You can define up to ten menu items.

Example – creating a new pull-down

Suppose you want to create a new menu bar item containing a selection which executes the **ping** command against a machine you select on the Machines window.

```
*userMachinePulldown: True
*userMachine.labelString: Commands
*userMachine_Option1: True
*userMachine_Option1_command: ping -c1
*userMachine_Option1.labelString: ping
*userMachine_Option1_parameter: True
*userMachine_Option1_output: Window
```

Figure 43. Creating a new pull-down menu

The **Xloadl** definitions shown in the Figure 43 create a menu bar item called “Commands”. The first item in the Commands pull-down menu is called “ping”. When you select this item, the command **ping -c1** is executed, with the machine you selected on the Machines window passed to this command. Your output is displayed in an informational window.

For more information, refer to the comments in the **Xloadl** file.

Customizing fields on the Jobs window and the Machines window

You can control which fields are displayed and which fields are not displayed on the Jobs window and the Machine window by changing the **Xloadl** file. Look in the **Xloadl** file for “Resources for specifying lengths of fields displayed in the Jobs and Machines windows”.

In most cases, you can remove a field from a window by setting its associated resource value to 0. To remove the Arch field from the Machines window, enter the following:

```
*mach_arch_len : 0
```

Note that the Job ID and Machine Name fields must always be displayed and therefore cannot be set to 0.

All fields have a minimum length value. If you specify a smaller value, the minimum is used.

Modifying help panels

Help panels have the same characteristics as all of the windows plus a few unique ones as shown in Table 83:

Table 83. Modifying help panels

Resource Variable	Values	Description
help*work_area.width	Any integer*	The width of the help panel.
help*work_area.height	Any integer*	The height of the help panel.
help*scrollHorizontal	[true false] The default is False.	Sets the scrolling option on or off.
help*wordWrap	[true false] The default is True.	Sets word wrapping on or off.

Note: * The work area and height depend upon your screen limitations.

Customizing the GUI

Chapter 16. Commands

LoadLeveler provides two types of commands: those that are available to all users of LoadLeveler, and those that are reserved for LoadLeveler administrators. If security services are not configured, then administrators are identified by the **LOADL_ADMIN** keyword in the configuration file. If security services are configured, the configuration file must identify the administrator's group. Refer to "Defining security mechanisms" on page 52 for more information.

The administrator commands can operate on the entire LoadLeveler job queue and all machines configured. The user commands mainly affect those jobs submitted by that user. Some commands, such as **llhold**, include options that can only be performed by an administrator.

Table 84 lists:

- All of the LoadLeveler commands
- The intended users
- The supported operating systems
- Whether the command can be issued across clusters to all clusters, a single cluster, or only within the local cluster
- A reference to the full description of each command

Table 84. LoadLeveler command summary

Command name	Intended users	Supported operating systems	Multicluster support	For more information, see...
llacctmrg	Administrators only	AIX and Linux	No	"llacctmrg - Collect machine history files" on page 387
llbind	Both administrators and general users	AIX and Linux	No	"llbind - Bind job steps to a reservation" on page 389
llcancel	Both administrators and general users	AIX and Linux	Single cluster	"llcancel - Cancel a submitted job" on page 392
llchres	Both administrators and general users	AIX and Linux	No	"llchres - Change attributes of a reservation" on page 395
llckpt	Both administrators and general users	AIX and Linux ¹	Yes	"llckpt - Checkpoint a running job step" on page 400
llclass	Both administrators and general users	AIX and Linux	Yes	"llclass - Query class information" on page 403
llclusterauth	Administrators only	AIX and Linux	No	"llclusterauth - Generates public and private keys" on page 408
llctl	Administrators only	AIX and Linux	No	"llctl - Control LoadLeveler daemons" on page 409
lldbconvert	Administrators only	AIX and Linux	No	"lldbconvert - Job migration utility" on page 414
llextRPD	Both administrators and general users	AIX and Linux	No	"llextRPD - Extract data from an RSCT peer domain" on page 415
llfavorjob	Administrators only	AIX and Linux	No	"llfavorjob - Reorder system queue by job" on page 419
llfavoruser	Administrators only	AIX and Linux	No	"llfavoruser - Reorder system queue by user" on page 421

LoadLeveler commands

Table 84. LoadLeveler command summary (continued)

Command name	Intended users	Supported operating systems	Multicluster support	For more information, see...
llfs	Administrators only	AIX and Linux	No	"llfs - Fair share scheduling queries and operations" on page 422
llhold	Both administrators and general users	AIX and Linux	Single cluster	"llhold - Hold or release a submitted job" on page 426
llinit	Administrators only	AIX and Linux	No	"llinit - Initialize machines in the LoadLeveler cluster" on page 429
llmkres	Both administrators and general users	AIX and Linux	No	"llmkres - Make a reservation" on page 431
llmodify	Both administrators and general users	AIX and Linux	Single cluster (not supported with -p , -s , -x , or -W)	"llmodify - Change attributes of a submitted job step" on page 435
llmovejob	Administrators only	AIX and Linux	No	"llmovejob - Move a single idle job from the local cluster to another cluster" on page 440
llmovespool	Administrators only	AIX and Linux	No	"llmovespool - Move job records" on page 442
llpreempt	Administrators only	AIX and Linux ²	No	"llpreempt - Preempt a submitted job step" on page 444
llprio	Both administrators and general users	AIX and Linux	Single cluster	"llprio - Change the user priority of submitted job steps" on page 447
llq	Both administrators and general users	AIX and Linux	Yes (not supported with -d , -w or -x)	"llq - Query job status" on page 449
llqres	Both administrators and general users	AIX and Linux	No	"llqres - Query a reservation" on page 468
llrmres	Both administrators and general users	AIX and Linux	No	"llrmres - Cancel a reservation" on page 474
llrunscheduler	Administrators only	AIX and Linux	No	"llrunscheduler - Run the central manager's scheduling algorithm" on page 476
llstatus	Both administrators and general users	AIX and Linux	Yes	"llstatus - Query machine status" on page 477
llsubmit	Both administrators and general users	AIX and Linux	Yes	"llsubmit - Submit a job" on page 494
llsummary	Both administrators and general users	AIX and Linux	No	"llsummary - Return job resource information for accounting" on page 496

¹ This command will run on LoadLeveler for Linux platforms, but it can only checkpoint jobs on AIX.

² On LoadLeveler for Linux platforms, the suspend preempt method is not supported.

llacctmrg - Collect machine history files

Purpose

llacctmrg – Collects individual machine history files together into a single file.

Syntax

llacctmrg [-?] [-H] [-v] [-R] [-h *hostlist*] [-d *directory*]

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- R Merges individual machine reservation history files into a single history file.
- h *hostlist*
Specifies a blank-delimited list of machines from which to collect data. The default is all machines in the LoadLeveler cluster.
- d *directory*
Specifies the directory to hold the new global history file. If not specified, the directory specified in the **GLOBAL_HISTORY** keyword in the configuration file is used.

Description

This command by default collects data from all the machines identified in the administration file. To override the default, specify a machine or a list of machines using the **-h** flag.

When the **llacctmrg** command ends, accounting information is stored in a file called **globalhist.YYYYMMDDHHmm**.

where:

- YYYY** Indicates the year
- MM** Indicates the month
- DD** Indicates the day
- HH** Indicates the hour
- mm** Indicates the minute.

Information such as the amount of resources consumed by the job and other job-related data is stored in this file.

Note that when the collection of accounting information to the global history file is complete, the accounting information is cleared in the history file.

For job data, you can use this file as input to the **llsummary** command. For example, if you created the file **globalhist.199808301050**, you can issue **llsummary globalhist.199808301050** to process the accounting information stored in this file.

When the **-R** flag is used to merge reservation history files instead of job history files, a file named **reservation_globalhist.YYYYMMDDHHmm** is created in the specified directory. You can view reservation data with any text editor. For more

llacctmrg

information on the format of the reservation history file, see the accounting information in Chapter 4, “Configuring the LoadLeveler environment,” on page 39.

Data on processes which fork child processes will be included in the file only if the parent process waits for the child process to end. Therefore, complete data may not be collected for jobs which are not composed of simple parent/child processes. For example, if a LoadLeveler job invokes an **rsh** command to execute some function on another machine, the resources consumed on the other machine will not be collected as part of the accounting data.

Examples

1. The following example collects data from machines named `mars` and `pluto`:

```
llacctmrg -h mars pluto
```
2. The following example collects data from the machine named `mars` and places the data in an existing directory called **merge**:

```
llacctmrg -h mars -d merge
```
3. The following example collects reservation history data from all machines in the LoadLeveler cluster:

```
llacctmrg -R
```

You should receive a response similar to the following:

```
llacctmrg: History transferred successfully from  
c94n04.ppd.pok.ibm.com (98 bytes).
```

A file named **reservation_globlhist.200610160915** is generated in the global history file location, assuming **llacctmrg** is issued at the time of 09:15 10/16/2006.

Results

The following shows a sample system response from the **llacctmrg -h mars -d merge** command:

```
llacctmrg: History transferred successfully from mars (10080 bytes)
```

Security

LoadLeveler administrators can issue this command.

llbind - Bind job steps to a reservation

Purpose

llbind – Binds job steps to a reservation in LoadLeveler, or unbinds job steps from the reservations to which they currently belong. The bound job steps will only be scheduled to run on the nodes reserved by the reservation.

Syntax

```
llbind { -? | -H | -v | [-q] { -r | -R reservation_ID } job_step_list }
```

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- q Specifies quiet mode: print no messages other than error messages.
- r Specifies an unbind operation. LoadLeveler eliminates any association between the job steps and a reservation.

-R reservation_ID

Specifies a reservation identifier to which the job steps will be bound. The format of a full LoadLeveler reservation identifier is *[host.]rid[r]*.

where:

- *host* is the name of the machine that assigned the reservation identifier.
- *rid* is the number assigned to the reservation when it was created. An *rid* is required.
- *r* indicates that this is a reservation ID (*r* is optional).

The reservation identifier may be specified in an abbreviated form, *rid[r]*, when the command is invoked on the same machine that assigned the reservation identifier. In this case, LoadLeveler will use the local machine's hostname to construct the full reservation identifier.

job_step_list

Is a blank-delimited list of job steps to be bound to the reservation or unbound from their respective reservations. The name of each job step should be in the form *[host].jobid[.stepid]*.

where:

- *host* is the name of the machine that assigned the job and step identifiers.
- *jobid* is the job number assigned to the job when it was submitted.
- *stepid* is the job step number assigned to the job step when it was submitted.

The job or step identifier may be specified in an abbreviated form, *jobid* or *jobid.stepid*, when the command is invoked on the same

machine that assigned the job and step identifiers. In this case, LoadLeveler will use the local machine's hostname to construct the full job or step identifier.

Note: For coscheduled job steps, even if all coscheduled job steps are not in the list of targeted job steps, the requested command is run on all coscheduled job steps.

Description

The **llbind** command is for LoadLeveler administrators and owner or users of a reservation. Regular users can only bind their own job steps to a reservation that they are allowed to use, while a LoadLeveler administrator can bind any job steps to any reservation.

Only job steps in an idle-like state can be bound to a reservation. A job step that is in an idle-like state that is already bound to a reservation can be bound to a new reservation using the **llbind** command. The command will first unbind the job step from the reservation it is currently bound to and then bind it to the requested reservation.

A Blue Gene job step can only be bound to a reservation with Blue Gene resources. A non-Blue Gene job step can only be bound to a reservation with non-Blue Gene resources.

The following conditions will cause the bind or unbind request for a job step to be ignored:

- The job step does not exist.
- The user is not the owner of the job step or a LoadLeveler administrator.
- The job step is requested to be bound to the reservation it is currently bound to.
- The job step is not associated with any reservation for a unbind request.
- Binding a Blue Gene job step to a reservation without Blue Gene resources.
- Binding a non-Blue Gene job step to a reservation with Blue Gene resources.

Note: A bind or unbind request for a coscheduled job step applies to all of the coscheduled job steps. If any of the coscheduled job steps meet one of these conditions, the requested operation will fail. The **llq -l** command can be used to check whether the request is successful (see "llq - Query job status" on page 449 for more information).

Bound job steps will be scheduled to run only on the reserved resources. The only exception is that the front end node (FEN) needed by a Blue Gene job step will be satisfied from outside of the reservation. A reservation for Blue Gene resource does not reserve resources for a FEN. The launch program **mpirun** needs to run on a front end node.

Only existing job steps queued in LoadLeveler can be bound to a reservation through this command. The **LL_RES_ID** environment variable can be used to bind an interactive POE job to a reservation or cause **llsubmit** to both submit and bind a batch job to a reservation. For additional information about setting the **LL_RES_ID** environment variable to bind an interactive POE job to a reservation, see Chapter 8, "Building and submitting jobs," on page 165. The **llqres** command can be used to view the list of job steps bound to the reservation.

This command is for the BACKFILL scheduler only.

Examples

1. To request to bind the job step c188f2n01.6.0 to reservation c188f1n03.2.r, issue:

```
llbind -R c188f1n03.2.r c188f2n01.6.0
```

You should receive a response similar to the following:

Request to bind job steps to reservation c188f1n03.2.r has been sent to LoadLeveler

2. To request to unbind the job step c188f2n01.6.0 from the reservation to which it is currently bound, issue:

```
llbind -r c188f2n01.6.0
```

You should receive a response similar to the following:

Request to unbind job steps from their respective reservations has been sent to LoadLeveler.

Security

LoadLeveler administrators and users can issue this command.

llcancel - Cancel a submitted job

Purpose

llcancel – Cancels one or more jobs from the LoadLeveler queue.

Syntax

```
llcancel      { -? | -H | -v | -f hostlist | [-q] [-X cluster_name]
               [-u userlist] [-h hostlist] [joblist] }
```

Flags

-? Provides a short usage message.

-H Provides extended help information.

-v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.

-f *hostlist*

Forces all jobs that are running on the machines in the *hostlist* to be vacated. Those machines in the *hostlist* are then marked as "Down" in the LoadLeveler cluster. The *hostlist* for the **-f** option should only specify machines that have gone down and should only be used for those machines that still have jobs displayed in the LoadLeveler queue.

The **-f** option is intended to be used by administrators for cleanup and recovery after a machine has permanently crashed or was inadvertently removed from the cluster before all activity has quiesced. If you need to return the machine to the cluster later, you must clear all files from the spool and execute directory of the machines in the *hostlist*.

-q Specifies quiet mode: print no messages other than error messages.

-X *cluster_name*

Specifies the name of a single cluster where the command is to run.

-u *userlist*

Is a blank-delimited list of users. When used with the **-h** option, only the user's jobs monitored on the machines in the *hostlist* are canceled. When used alone, only the user's jobs monitored by the machine issuing the command are canceled.

-h *hostlist*

Is a blank-delimited list of machine names. All jobs monitored on machines in this list are canceled. When issued with the **-u** option, the *userlist* is used to further select jobs for cancellation.

joblist

Is a blank-delimited list of job and step identifiers. When a job identifier is specified, the command action is taken for all steps of the job. At least one job or step identifier must be specified.

The format of a job identifier is *host.jobid*. The format of a step identifier is *host.jobid.stepid*.

where:

- *host* is the name of the machine that assigned the job and step identifiers.
- *jobid* is the job number assigned to the job when it was submitted.
- *stepid* is the job step number assigned to the job step when it was submitted.

The job or step identifier may be specified in an abbreviated form, *jobid* or *jobid.stepid*, when the command is invoked on the same machine that assigned the job and step identifiers. In this case, LoadLeveler will use the local machine's hostname to construct the full job or step identifier.

Note: For coscheduled jobs, even if all coscheduled job steps are not in the list of targeted job steps, the requested operation is performed on all coscheduled job steps.

The **-u** or **-h** flags override the *joblist* parameter.

When the **-h** flag is specified by a non-administrator, all jobs submitted from the machines in *hostlist* by the user issuing the command are canceled.

When the **-h** flag is specified by an administrator, all jobs submitted by the administrator are canceled, unless the **-u** is also specified, in which case all jobs both submitted by users in *userlist* and monitored on machines in *hostlist* are canceled.

Group administrators and class administrators are considered normal users unless they are also LoadLeveler administrators.

Description

When you issue **llcancel**, the command is sent to the negotiator. You should then use the **llq** command to verify your job was canceled. A job state of CA (Canceled) indicates the job was canceled. A job state of RP (Remove Pending) indicates the job is in the process of being canceled.

When cancelling a job from a submit-only machine, you must specify the machine name that scheduled the job. For example, if you submitted the job from machine A, a submit-only machine, and machine B, a scheduling machine, scheduled the job to run, you must specify machine B's name in the cancel command. If machine A and B are in different sub-domains, you must specify the fully qualified name of the job in the cancel command. You can use the **llq -l** command to determine the fully qualified name of the job.

Examples

1. This example cancels the job step 3 that is part of the job 18 that is scheduled by the machine named bronze:


```
llcancel bronze.18.3
```
2. This example cancels all the job steps that are a part of job 8 that are scheduled by the machine named gold:


```
llcancel gold.8
```
3. This example cancels the job steps that are a part of job 5 that is scheduled to run in cluster1:


```
llcancel -X cluster1 silver.5
```

Results

1. The following shows a sample system response for the **llcancel gold.8** command:


```
llcancel: Cancel command has been sent to the central manager.
```

llcancel

2. The following shows a sample system response for the **llcancel -X cluster1 silver.5** command. The remote command has been sent to the central manager in cluster1:

```
llcancel: Cancel command has been sent to the central manager.
```

Security

LoadLeveler administrators and users can issue this command.

llchres - Change attributes of a reservation

Purpose

llchres – Changes one or more of the attributes of a LoadLeveler reservation.

Syntax

```
llchres { -? | -H | -v | [-q] [-t start_time | -t {+|-} minutes]
        [-d [+|-] duration] [-n [+|-] number_of_nodes | -h free |
        -h [+|-] host_list | -j job_step | -f job_command_file |
        -c number_of_bg_cnodes] [-U [+|-] user_list] [-G [+|-] group_list]
        [-s {yes|no}] [-i {yes|no}] [-u user] [-g group] -R reservation }
```

Flags

-? Provides a short usage message.

-H Provides extended help information.

-q Specifies quiet mode: print no messages other than error messages.

-v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.

-t *start_time* Modifies the *start_time* for a reservation using a 24-hour clock. The format [mm/dd[/[cc]yy]] HH:MM must be used.

-t {+|-} *minutes* Modifies the start time for a reservation using *minutes*. When *minutes* is preceded by a plus (+) sign or minus (-) sign, the current start time is postponed or moved closer, respectively, by the number of minutes specified.

-d [+|-] *duration* Specifies a new duration for the reservation in minutes. If *duration* is preceded by a plus (+) sign or minus (-) sign, the current duration of the reservation is increased or decreased, respectively, by the value specified.

-n [+|-] *number_of_nodes* Specifies a new request for the number of nodes to reserve. If *number_of_nodes* is preceded by a plus (+) sign or minus (-) sign, the current number of nodes in the reservation is increased or decreased, respectively, by the value specified.

-h **free**

-h [+|-] *host_list* Specifies a change to the list of machines to be reserved. When a blank-delimited host list is specified, it indicates that a new list of hosts are to be reserved. Specifying a plus (+) sign before host list adds the listed machines to the reservation. Specifying a minus (-) sign removes the listed machines from the reservation. Note that when a host list is specified, the first character of any host name cannot be a plus (+) or minus (-) sign. Specifying the reserved word **free** reserves all machines available for this reservation, which currently have an active **LoadL_startd** daemon. The reservation change request will succeed if at least one node can be included in the reservation.

- | **-j** *job_step* Specifies a new request that a set of nodes or Blue Gene compute nodes (C-nodes) that the job step can run on be reserved. The job step must be in an idle-like state and takes the form *[host.]jobid.stepid*.
- | where:
- | • *host* is the name of the machine that assigned the step identifier.
 - | • *jobid* is the job number assigned to the job when it was submitted.
 - | • *stepid* is the job step number assigned to the job step when it was submitted.
- | The step identifier may be specified in an abbreviated form, *jobid.stepid*, when the command is invoked on the same machine that assigned the step identifier. In this case, LoadLeveler will use the local machine's hostname to construct the full step identifier.
- | You must be an administrator or the owner of both the reservation and job step to make this request. If the request to modify the reservation is successful, the job step will be bound to the reservation.
- | **-f** *job_command_file* Specifies the full path to a new *job_command_file* that will be submitted and the first job step used to determine what resources to reserve. The job identifier of the newly created job will be displayed. All job steps will be bound to the reservation, or if the modification request fails, will be placed in the **NotQueued** state. The job identifier of the newly created job will be displayed.
- | **-c** *number_of_bg_cnodes* Specifies the number of C-nodes to reserve in the Blue Gene system. The shape of the allocated resource for a given size cannot be guaranteed, but the size of the allocated shape will be no smaller than the requested size and will be as close to the requested size as possible.
- | **-U** [+|-] *user_list* Specifies a new blank-delimited list of users who can use the reservation. If the list of users is preceded by a plus (+) sign or minus (-) sign, add those users to or remove those users from the existing list of users that can use the reservation, respectively.
- | **-G** [+|-] *group_list* Specifies a new blank-delimited list of LoadLeveler groups that can use the reservation. If the list of groups is preceded by a plus (+) sign or minus (-) sign, those groups will be added to or removed from the existing list of groups, respectively. The first character of any group name cannot be plus (+) sign or minus (-) sign.
- | **-s** {**yes** | **no**} SHARED mode is enabled when the reserved word **yes** is specified. When the reserved word **no** is specified, SHARED mode is disabled.
- | **-i** {**yes** | **no**} REMOVE_ON_IDLE mode is enabled when the reserved word **yes** is specified. When the reserved word **no** is specified, REMOVE_ON_IDLE mode is disabled.
- | **-u** *user* Specifies a new user ID that will own the reservation.
- | **-g** *group* Specifies a new LoadLeveler group that will own the reservation.

-R reservation Specifies the reservation identifier to be modified. The format of a full LoadLeveler reservation identifier is `[host.]rid[.r]`.

where:

- *host* is the name of the machine that assigned the reservation identifier.
- *rid* is the number assigned to the reservation when it was created. An *rid* is required.
- *r* indicates that this is a reservation ID (*r* is optional).

The reservation identifier may be specified in an abbreviated form, `rid[.r]`, when the command is invoked on the same machine that assigned the reservation identifier. In this case, LoadLeveler will use the local machine's host name to construct the full reservation identifier.

Note: When a plus (+) sign or minus (-) sign is used to increase or decrease a value, there cannot be spaces between the plus (+) sign or minus (-) sign and the value.

Description

The **llchres** command is for LoadLeveler administrators and the owner of a reservation. Either all requested changes will be made and a message indicating the reservation request has been sent will be displayed, or none of the changes will be made and a message describing the reason for the failure will be displayed along with the message that the request was sent. If a connection error occurs and the request cannot be sent, a message will be displayed.

Note that it is possible for a time out to occur while this command is waiting for a response from the LoadLeveler central manager. Even if a time out occurs or the command process is killed, the command may still succeed. To determine if the request has been granted, issue the **llqres** command.

Modification requests are subject to resource availability checks and reservation policies.

Notes on changing a reservation:

- Administrators can change the attributes of any reservation, including the user ID that owns the reservation, while non-administrators can change attributes of only the reservations they own and cannot change the reservation owner.
- A reservation owner who is not a LoadLeveler administrator cannot change the start time, duration or reserved resources if the start time is not at least later than the current time by the number of minutes specified by the **RESERVATION_MIN_ADVANCE_TIME** keyword.
- The new reservation start time must be later than the current time by at least the amount of time specified by the **RESERVATION_SETUP_TIME** keyword.
- A reservation in the CANCEL or COMPLETE state cannot be changed.
- When a reservation is not in the WAITING state, the start time cannot be changed.
- When a reservation is not in the WAITING state, the only ways to change reserved nodes are to add a number of nodes, or to add or delete a list of nodes. Blue Gene resources cannot be changed for a reservation not in the WAITING state.

llchres

- A reserved node with a bound step running cannot be removed from the reservation.
- When changing the reservation duration, the end time of the reservation must be later than the current time.
- You cannot delete all reserved resources from a reservation; a reservation must have at least one reserved node or Blue Gene C-node.
- You cannot add a node that is already reserved to a reservation when using **llchres -h +host_list**.
- You cannot delete a node that is not reserved from a reservation when using **llchres -h -host_list**.
- If you want to change the owner of a reservation, the new owner must be able to own an additional reservation (**max_reservations** for the user is not specified or if specified, the quota is not used up yet). If you want to change the group that owns the reservation, the new group must be able to own an additional reservation (**max_reservations** for the group is not specified or if specified, the quota is not used up yet). If the change request is granted, the new owner and group must have the permission to own a new reservation (they cannot both have **max_reservations** unspecified).
- A coscheduled job step cannot be specified when using the **-j** or **-f** flags.
- You cannot change a reservation to reserve a number of nodes if the reservation already has some Blue Gene C-nodes reserved.
- You cannot change a reservation to reserve a number of Blue Gene C-nodes if the reservation already has some nodes reserved.
- The **llchres -s no** option will fail for a reservation in the Active_Shared (AS) state.
- Shared mode must be **yes** for a reservation to change from Active (A) state to the AS state. Once the reservation is in the AS state, the shared mode cannot be changed again.

This command is for the BACKFILL scheduler only.

Examples

1. To have reservation c94n16.20.r start an hour later than currently scheduled with four fewer nodes, issue:

```
llchres -t +60 -n -4 -R c94n16.20.r
```

You should receive a response similar to the following:

```
Request to change reservation c94n16.20.r has been sent to LoadLeveler
```

2. To change the duration from 20 to 50 minutes and enable only users chris, jay, and dave to use the reservation c94n16.31.r, issue:

```
llchres -U chris jay dave -d 50 -R c94n16.31.r
```

You should receive a response similar to the following:

```
Request to change reservation c94n16.31.r has been sent to LoadLeveler
```

3. To change the number of Blue Gene C-nodes in reservation c94n03.2.r, issue:

```
llchres -c 512 -R c94n03.2.r
```

You should receive a response similar to the following:

```
Request to change reservation c94n03.2.r has been sent to LoadLeveler
```

Security

LoadLeveler administrators and users can issue this command.

llckpt - Checkpoint a running job step

Purpose

llckpt – Checkpoints a single job step.

Syntax

```
llckpt { -? | -H | -v | [-k | -u] [-r] [-q] [-X cluster_name] jobstep }
```

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- k Specifies that the job step is to be terminated after a successful checkpoint. The default is for the job to continue. Note that you cannot use the **-k** and **-u** flags together. If you need to restart the job on the same node, do not use the **-k** flag.
- u Specifies that the job step is to be put on user hold after a successful checkpoint. The default is for the job to continue. Note that you cannot use the **-k** and **-u** flags together.
- r When this flag is issued, it specifies that the command is to return without waiting for the checkpoint to complete. When using this flag you should be aware that information relating to the success or failure of the checkpoint will not be available to the command. The default is for the checkpoint to complete before returning.
- q Specifies quiet mode: print no messages other than error messages.
- X *cluster_name*
Specifies the name of a multicluster where the command is to run.
- jobstep*
Is the name of a job step to be checkpointed.

The format of a full LoadLeveler step identifier is *host.jobid.stepid*.

where:

- *host* is the name of the machine that assigned the step identifier.
- *jobid* is the job number assigned to the job when it was submitted.
- *stepid* is the job step number assigned to the job step when it was submitted.

The step identifier may be specified in an abbreviated form, *jobid.stepid*, when the command is invoked on the same machine that assigned the step identifier. In this case, LoadLeveler will use the local machine's hostname to construct the full step identifier.

Description

The **llckpt** command should be used to save the state of the job in the event it does not complete. Use the command only with jobs that are marked as checkpointable. You can mark a job step for checkpoint by specifying **checkpoint=yes** or **checkpoint=interval** in the job command file. Use **checkpoint=yes** to set checkpointing for an interactive job. For more information, see "LoadLeveler support for checkpointing jobs" on page 129.

When a job is checkpointed it can later be restarted from the checkpoint file rather than the beginning of the job. To restart a job from a checkpoint file, the original job command file should be used with the value of the **restart_from_ckpt** keyword set to **yes**. The name and location of the checkpoint file should be specified by the **ckpt_dir** and **ckpt_file** keywords.

If you need to restart the job on the same nodes, do not use the **-k** flag. Instead, use the **-u** flag to place the job in a hold state. You can later release the job from the hold state by issuing the **llhold -r** command. Note that a coscheduled job step cannot be specified with the **-u** flag.

Examples

1. This example checkpoints the job step 1 that is part of job 12 which was scheduled by the machine named **iron**. Upon successful completion of checkpoint, the job step will return to the RUNNING state.
llckpt iron.12.1
2. This example checkpoints the job step 3 that is part of job 14 which was scheduled by the machine named **bronze**. Upon successful completion of checkpoint the job step will be put on user hold:
llckpt -u bronze.14.3

Results

When the **-r** option is not used, the **llckpt** command will wait for the checkpoint to complete. Immediately upon executing the command **llckpt iron.12.1** the following message is displayed:

```
llckpt: The llckpt command will wait for the results of the checkpoint on
job step iron.12.1 before returning
```

Once the checkpoint has successfully completed, the following message is displayed:

```
llckpt: Checkpoint of job step iron.12.1 completed successfully
```

If there was a problem taking the checkpoint of an AIX job, the second message would have this form:

```
llckpt: Checkpoint FAILED for job step iron.12.1 with the following
error:
```

```
primary error code = <numeric error number>,
secondary error code = <secondary numeric error/extended numeric error>,
error msg len = <length of message>,
error msg = <text describing the error>
```

where: primary error code is defined by **/usr/include/sys/errno.h** and secondary error code is defined by **/usr/include/sys/chkerror.h**.

The **-r** option is used to return without waiting for the result of a checkpoint. The following output is displayed for the command **llckpt -r bronze.14.3**:

```
llckpt: The llckpt command will not wait for the checkpoint of
job step bronze.14.3 to complete before returning.
```

Due to delays in communication between LoadLeveler daemons, status information may not be returned at the same time that checkpoint termination is received. This indicates that the checkpoint has completed but the success or failure status is not known. When this happens, the following message is displayed:

llckpt

llckpt: Checkpoint of job step iron.12.1 completed. No status information is available.

Security

LoadLeveler administrators and users can issue this command.

llclass - Query class information

Purpose

llclass – Returns information about classes.

Syntax

```
llclass [-?] [-H] [-v] [-W] [-l] [-X {cluster_list | all}] [-c classlist | classlist]
```

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- W Specifies that the width of columns in tabular output will be increased to fit the widest entry.
- l Specifies that a long listing be generated for each class for which status is requested. If **-l** is *not* specified, then the standard listing is generated.
- X {*cluster_list* | **all**}
Indicates that you can specify the **-X** flag with either:
 - cluster_list* Is a blank-delimited list of clusters where the command is to run.
 - all** Is the reserved word indicating that the command is to run in all accessible clusters.
- c *classlist* | *classlist*
Is a blank-delimited list of classes for which you are requesting status. The **-c classlist** flag is used to distinguish a classlist when specified in combination with the **-X** flag. If a *classlist* is not specified, all classes are queried.

If you have more than a few classes configured for LoadLeveler, consider redirecting the output to a file when you use the **-l** flag.

Description

The **llclass** command queries information about job classes. The output of this command displays the number of defined classes and usage information.

Examples

1. To generate a standard listing for class `Parallel`, issue:

```
llclass Parallel
```

This example generates the standard listing where there are 24 initiators of class `Parallel` configured in the cluster, with one job step of class `Parallel` using 6 initiators currently running. You should receive output similar to the following:

Name	MaxJobCPU d+hh:mm:ss	MaxProcCPU d+hh:mm:ss	Free Slots	Max Slots	Description

llclass

```
Parallel 2+02:45:00 05:30:00 18 24 Parallel job class
```

The **standard listing** includes the following fields:

Description

Lists the information provided in the **class_comment** keyword for the specified class. The **class_comment** keyword is defined in the class stanza of the LoadLeveler administration file.

Free Slots

The number of initiators (slots) available for the specified class in the LoadLeveler cluster. A serial job step uses one initiator at run time. A parallel job step with N tasks uses N initiators at run time.

MaxJobCPU

The hard job CPU limit of job steps for the specified class. For a description of the job CPU limit for serial and parallel job steps, see the `job_cpu_limit` keyword.

MaxProcCPU

The hard CPU limit for the processes of the job steps of the specified class.

Max Slots

The number of configured initiators (slots) for the specified class in the LoadLeveler cluster.

Name The name of the class.

2. To generate a long listing for classes named *silver* and *gold*, issue:

```
llclass -l silver gold
```

The **long listing** includes the following fields:

Admin

The list of administrators for the specified class.

Ckpt_limit

Hard and soft checkpoint limits of a job step of the specified class.

Class_ckpt_dir

The name of the directory containing the checkpointing files of job steps of the specified class.

Class_comment

Lists the information provided in the **class_comment** keyword for the specified class. The **class_comment** keyword is defined in the class stanza of the LoadLeveler administration file.

Core_limit

The hard and soft core size limits of processes of job steps of the specified class.

Cpu_limit

The hard and soft CPU limits of processes of job steps of the specified class.

Data_limit

The hard and soft data area limits of processes of job steps of the specified class.

Def_wall_clock_limit

The default wall clock limit to those jobs that have no wall clock limit specified in their job command files.

Exclude_Groups

Groups who are not allowed to submit jobs of the specified class.

Exclude_Users

Users who are not permitted to submit jobs of the specified class.

Free_slots

The number of available initiators (slots) for the specified class in the LoadLeveler cluster. A serial job step uses one initiator of the appropriate class at run time. A parallel job step with N tasks uses N initiators at run time.

File_limit

The hard and soft file size limits of processes of job steps of the specified class.

Include_Groups

Groups having permission to submit jobs of the specified class.

Include_Users

Users who are permitted to submit jobs of the specified class.

Job_cpu_limit

The hard and soft job CPU limits of job steps of the specified class. For a description of the job CPU limit for serial and parallel job steps, see the `job_cpu_limit` keyword.

Maximum_slots

The total number of configured initiators (slots) for the specified class in the LoadLeveler cluster.

Maxjobs

The maximum number of job steps of the specified class that can run at any time in the LoadLeveler cluster.

Max_processors

The maximum number of processors than can be used for a parallel job step of the specified class.

Max_total_tasks

Used for BACKFILL scheduling only. `Max_total_tasks` sets the maximum number of tasks allowed to run at any given time for job steps of the specified class in the LoadLeveler cluster.

Max_proto_instances

The maximum number of protocol instances allowed for a job step of the specified class.

Name The name of the class

Nice The *nice* value of jobs of the specified class.

Preempt_class

Used for BACKFILL scheduling only, `Preempt_class` sets the preemption rule for job steps of the specified class.

Priority

The system priority of the specified class relative to other classes.

llclass

Resource_requirement

The default consumable resource requirements for job steps of the specified class.

Rss_limit

The hard and soft rss size limits of processes of job steps of the specified class.

Stack_limit

The hard and soft stack size limits of processes of job steps of the specified class.

Start_class

Used for BACKFILL scheduling only, Start_class sets the starting rule for job steps of the specified class.

Wall_clock_limit

The hard and soft wall clock (elapsed time) limits of job steps of the specified class.

See Appendix B, "Sample command output," on page 673 for sample output of long listings.

3. This example generates the standard listing for all accessible clusters including the local cluster in a multicluster environment:

```
llclass -X all
```

The output representing a cluster is delineated with a cluster header similar to the following:

```
===== Cluster c556_Cluster1 =====
```

Name	MaxJobCPU d+hh:mm:ss	MaxProcCPU d+hh:mm:ss	Free Slots	Max Slots	Description
mpich	3+08:00:00	12:30:00	100	132	MPICH Jobs
parallel	23:59:00	01:00:00	32	256	POE Parallel Jobs
No_Class	01:00:00	00:30:00	120	512	Default Class
large	2+08:00:00	18:30:00	50	128	Large Serial Jobs
medium	12:00:00	02:30:00	60	128	Medium Serial Jobs
small	01:00:00	00:30:00	12	128	Small Serial Jobs

```
===== Cluster c556_Cluster2 =====
```

Name	MaxJobCPU d+hh:mm:ss	MaxProcCPU d+hh:mm:ss	Free Slots	Max Slots	Description
mpich	3+08:00:00	12:30:00	110	132	MPICH Jobs
parallel	23:59:00	01:00:00	48	256	POE Parallel Jobs
No_Class	01:00:00	00:30:00	128	512	Default Class
large	2+08:00:00	18:30:00	74	128	Large Serial Jobs
ESSL	23:00:00	12:30:00	55	128	ESSL Jobs
OSL	12:00:00	06:00:00	33	128	OSL Jobs

Related Information

Each machine periodically updates the central manager with a snapshot of its environment. Since the information returned by **llclass** is a collection of these snapshots, all taken at varying times, the total picture may not be completely consistent.

Security

LoadLeveler administrators and users can issue this command.

llclusterauth - Generates public and private keys

Purpose

llclusterauth – Generates public and private keys that are used to provide secure intercluster communications.

Syntax

```
llclusterauth [-?] | [-H] | [-v] | [-k]
```

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- k Creates a public key, a private key, a security certificate, and a directory for authorized keys. The keys and certificate are created in the `/var/LoadL/ssl` directory for AIX and in the `/var/opt/LoadL/ssl` directory for Linux.
 - The private key is stored in `id_rsa`
 - The public key is stored in `id_rsa.pub`
 - The security certificate is stored in `id_rsa.cert`
 - The authorized keys are stored in `authorized_keys`

This command must run with **root** authority when using the **-k** flag to create key files.

If any directory in the path for the security files does not exist, the command will create the directory and set the owner to **root** and set the permissions to '0700'. The key and certificate files will be owned by **root** with permissions of '0600'.

Description

The **llclusterauth** command generates public and private keys that are used to provide secure intercluster communications. When multicluster security is configured to use Secure Sockets Layer (SSL), a connection on a secure port will be accepted only if the public key for the node requesting the connection is stored in a file in the authorized keys directory on the node being connected to.

Standard Error

- An error message is issued and the command exits for the following error cases:
- The command process does not have **root** authority
 - A required directory for the security files cannot be created
 - A security file cannot be created

Security

LoadLeveler administrators can issue this command.

llctl - Control LoadLeveler daemons

Purpose

llctl – Controls LoadLeveler daemons on all members of the LoadLeveler cluster.

Syntax

llctl [-?] [-H] [-v] [-q] [-g | -h *hostname*] *keyword*

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- q Specifies quiet mode: print no messages other than error messages.
- g Indicates that the command applies globally to all machines, except submit-only machines, that are listed in the administration file.
- h *host*
Indicates that the command applies to only the *host* machine in the LoadLeveler cluster. If neither -h nor -g is specified, the default is the machine on which the llctl command is issued.

keyword

Must be specified after all flags and can be the following:

capture *eventname*

Captures accounting data for all jobs running on the designated machines. *eventname* is the name you associate with the data, and must be a character string containing no blanks. For more information, see “Collecting job resource data based on events” on page 59.

drain [**schedd** | **startd** [*classlist* | **allclasses**]]

When you issue **drain** with no options, the following happens: (1) no more LoadLeveler jobs can begin running on this machine, and (2) no more LoadLeveler jobs can be submitted through this machine. When you issue **drain schedd**, the following happens: (1) the Schedd machine accepts no more LoadLeveler jobs for submission, (2) job steps in the Starting or Running state in the Schedd queue are allowed to continue running, and (3) job steps in the Idle state in the Schedd queue are drained, meaning they will not get dispatched. When you issue **drain startd**, the following happens: (1) the startd machine accepts no more LoadLeveler jobs to be run, and (2) job steps already running on the startd machine are allowed to complete. When you issue **drain startd classlist**, the classes you specify which are available on the startd machine are drained (made unavailable). When you issue **drain startd allclasses**, all available classes on the startd machine are drained.

flush

Terminates running jobs on this machine and sends them back, in the Idle state, to the negotiator to await redispach (provided **restart=yes** in the job command file). No new jobs are sent to this machine until **resume** is issued. Forces a checkpoint if jobs are enabled for checkpointing. However, the checkpoint gets canceled if it does not complete within the time period specified in the **ckpt_time_limit** keyword in the job command file.

purgeschedd

Requests that all jobs scheduled by the specified *host* machine be purged (removed). To use this keyword, you must first specify **schedd_fenced=true** in the machine stanza for this *host*. The **-g** option cannot be specified with this keyword. For more information, see “How do I recover resources allocated by a Schedd machine?” on page 660.

reconfig

Forces all daemons to reread the administration and configuration files.

recycle

Stops all LoadLeveler daemons and restarts them.

resume [schedd | startd [*classlist* | allclasses]]

When you issue **resume** with no options, job submission and job execution on this machine is resumed. When you issue **resume schedd**, the Schedd machine resumes the submission of jobs. When you issue **resume startd**, the startd machine resumes the execution of jobs. When you issue **resume startd *classlist***, the startd machine resumes the execution of those job classes you specify which are also configured (defined on the machine). When you issue **resume startd allclasses**, the startd machine resumes the execution of all configured classes.

start [drained]

When you issue **start** with no options it starts the LoadLeveler daemons on the machine or machines designated, either explicitly or implicitly. When you issue **start** without the **-g** or **-h** flag the LoadLeveler daemons are started on the same machine that issued the command. When you issue **llctl start** with either the **-g** or **-h** flag, the command specified by the **LL_RSH_COMMAND** configuration file keyword is used to run the command on all machines specified in the administration file. If **LL_RSH_COMMAND** is not specified, remote shell (**rsh**) is used and you must have **rsh** privileges in order to use **llctl start** with either the **-g** or **-h** flag.

When you issue **start** with the **drained** option the LoadLeveler daemons are started, but the startd daemon is started in the drained state.

LoadLeveler commands that run **rshell** include **llctl version** and **llctl start**.

stop

Stops the LoadLeveler daemons on the specified machine.

suspend

Suspends all jobs on this machine. This is not supported for parallel jobs.

version

Displays release number, service level, service level date, and operating system information for every LoadLeveler executable.

When you issue **llctl version** with either the **-g** or **-h** flag, the command specified by the **LL_RSH_COMMAND** configuration file keyword is used to run the command on all machines specified in the administration file. If **LL_RSH_COMMAND** is not specified, remote shell (**rsh**) is used and you must have **rsh** privileges in order to use **llctl version** with either the **-g** or **-h** flag.

LoadLeveler commands that run **rshell** include **llctl version** and **llctl start**.

Description

This command sends a message to the master daemon on the target machine requesting that action be taken on the members of the LoadLeveler cluster. Note the following when using this command:

- To perform the control operations of the **llctl** command, you must be a LoadLeveler administrator. The only exception to this rule is the "start" operation.
- LoadLeveler will fail to start if any value has been set for the `MALLOCTYPE` environment variable.
- After you make changes to the administration and configuration files for a running cluster, be sure to issue **llctl reconfig**. This command causes the LoadLeveler daemons to reread these files, and prevents problems that can occur when the LoadLeveler commands are using a new configuration while the daemons are using an old configuration.

Note: Changes to `SCHEDULER_TYPE` will not take effect at reconfiguration. The administrator must stop and restart or recycle LoadLeveler when changing `SCHEDULER_TYPE`.

- The **llctl drain startd classlist** command drains classes on the startd machine, and the startd daemon remains operational. If you reconfigure the daemon, the draining of classes remains in effect. However, if the startd goes down and is brought up again (either by the master daemon or by a LoadLeveler administrator), the startd daemon is configured according to the global or local configuration file in effect, and therefore the draining of classes is lost.

Draining all the classes on a startd machine is *not* equivalent to draining the startd machine. When you drain all the classes, the startd enters the Idle state. When you drain the startd, the startd enters the Drained state. Similarly, resuming all the classes on a startd machine is *not* equivalent to resuming the startd machine.

- If a job step is running on a machine that receives the **llctl recycle** command, or the **llctl stop** and **llctl start** commands, the running job step is terminated. If the restart option in the job command file was set to yes, then the job step will be restarted when LoadLeveler is restarted. If the job step is checkpointable, it will be restarted from the last valid checkpoint file when LoadLeveler is restarted.
- If you find that the **llctl -g** command (even if it is specified with additional options) is taking a long time to complete, you should consider using the AIX command **dsh** to send **llctl** commands (omitting the **-g** flag) to multiple nodes in a parallel fashion.
- When a node running a Schedd daemon fails, resources that have been allocated to any of the jobs scheduled by that Schedd are unavailable until the Schedd is restarted. Administrators can, however, recover these resources by using the **llctl** command's **purgeschedd** keyword to purge (remove) all of the jobs scheduled by the Schedd on the down node. The **purgeschedd** keyword can only work in conjunction with the **schedd_fenced** keyword, in the administration file, which causes the central manager to ignore (fence) the Schedd daemon running on the target node. You must reconfigure the central manager so it can recognize this fence. To use the **purgeschedd** keyword:
 1. Recognize that a node running a Schedd daemon is down, and that the node will be down long enough to necessitate that you recover the resources allocated to jobs scheduled by that Schedd.
 2. Add the statement "`schedd_fenced = true`" to the failed node's administration file machine stanza.

3. Reconfigure the central manager node so that the central manager recognizes the fenced Schedd daemon.
4. Invoke "llctl -h *host* purgeschedd" to purge all of the jobs scheduled by the Schedd on the failed node.
5. Once the failed node is working again, remove all of the files in the LoadLeveler spool directory. Remove the "schedd_fenced = true" statement from the administration file, then reconfigure the central manager node before starting Schedd on the machine.

Examples

1. This example stops LoadLeveler on the machine named *iron*:

```
llctl -h iron stop
```

2. This example starts the LoadLeveler daemons on all members of the LoadLeveler cluster (with the exception of the submit-only machines), starting with the central manager, as defined in the machine stanzas of the administration file:

```
llctl -g start
```

3. This example causes the LoadLeveler daemons on machine *iron* to re-read the administration and configuration files, which may contain new configuration information for the *iron* machine:

```
llctl -h iron reconfig
```

4. This example drains the classes *medium* and *large* on the machine named *iron*.

```
llctl -h iron drain startd medium large
```

5. This example drains the classes *medium* and *large* on all machines.

```
llctl -g drain startd medium large
```

6. This example stops all the jobs on the system, then allows only jobs of a certain class (*medium*) to run.

```
llctl -g drain startd allclasses
llctl -g flush
llctl -g resume
llctl -g resume startd medium
```

7. This example resumes the classes *medium* and *large* on the machine named *iron*.

```
llctl -h iron resume startd medium large
```

8. This example illustrates how to capture accounting information on a work shift called *day* on the machine *iron*:

```
llctl -h iron capture day
```

You can capture accounting information on all the machines in the LoadLeveler cluster by using the **-g** option, or you can collect accounting information on the local machine by simply issuing the following:

```
llctl capture day
```

Capturing information on the local machine is the default. For more information, see "Collecting job resource data based on events" on page 59.

Security

LoadLeveler administrators can issue this command.

lldbconvert - Job migration utility

Purpose

lldbconvert – Administrators can use the **lldbconvert** utility to convert jobs from LoadLeveler 3.3 format to LoadLeveler 3.4 format.

Syntax

lldbconvert [-d] [-D] [-H] [-?] [-v] [-o *OpSys*]

Flags

-d	Displays contents of input spool (no conversion).
-D	Converts LoadLeveler 3.3 spool to LoadLeveler 3.4 spool and displays contents of output spool after conversion.
-H	Provides extended help information.
-?	Provides a short usage message.
-v	Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
-o <i>OpSys</i>	Converts LoadLeveler 3.3 spool to LoadLeveler 3.4 spool, forcing OpSys test to: ' OpSys == "<i>OpSys</i>" if it occurs in Requirements or Preferences.

Description

On machines where jobs need to be migrated, log in as **loadl** (or the primary LoadLeveler user ID defined in **/etc/LoadL.cfg**) and run **lldbconvert** from the local LoadLeveler **spool** directory.

The **-o** flag is needed when the operating system specified in the requirement statement no longer matches the operating system after the migration, for example, if it changes from AIX52 to AIX53. This flag converts requirements statements in the form **OpSys=="*your_os*"**. Statements in other formats (such as those with *your_os* not enclosed in double quotes) are not converted.

When you create the **LoadL_admin** file for the version to which you are migrating, you should preserve all of the job class definitions used in the version from which you are migrating. For example, if a converted job needs to run as a large class job, and class large is not defined in the LoadLeveler 3.4 **LoadL_admin** file, the job will never be run.

lldbconvert only converts job steps that are in one of the following states: **Idle**, **Hold**, **Deferred**, or **NotQueued**. If the conversion is successful, you will receive further instructions from **lldbconvert** on what to do before starting LoadLeveler on the given machine.

Once you run **lldbconvert**, do not move the converted job queues to another **Schedd** machine.

Security

LoadLeveler administrators can issue this command.

llextrPD - Extract data from an RSCT peer domain

Purpose

llextrPD – Extracts the necessary data from a Reliable Scalable Cluster Technology (RSCT) peer domain (or local node if there is no active domain) to set up the administration file.

Syntax

```
llextrPD [ -? | -H | -v | [-m] [-a adapter_name] [-h hostlist] ]
```

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- m Specifies that only machine stanzas are to be generated. The adapter stanzas (and the corresponding **adapter_stanzas** statement of the machine stanza) will be suppressed in the final output. This option is for Dynamic Adapter Configuration support for peer domains with AIX RSCT.

LoadLeveler will dynamically detect and handle adapters and adapter changes for any machine in these domains which do not specify an adapter stanza in the administration file.

-a *adapter_name*

Specifies that the interface name of the given *adapter_name* on each node is used as the label (machine stanza name) of the generated machine stanza.

If you do not specify an adapter (or if an adapter is specified but does not exist on a particular node) then the label used for that machine is the Name field from the RSCT IBM.PeerNode class for the machine in the cluster.

Note: If an administrator wants to configure LoadLeveler to communicate using the Switch Network Interface for High Performance Switch (HPS) adapters in a peer domain they should use the **-a** flag with **m10** specified as the *adapter_name*. **m10** is guaranteed to be present on every node that contains an HPS adapter.

It is recommended that you do not specify **sn** as the adapter name. If you do, the machine will be named with the IP name of the **sn** adapter. If that IP name becomes unavailable because the adapter changes, LoadLeveler will not be able to contact any daemons on that machine.

-h *hostlist*

Specifies one or more host match terms. If multiple host match terms are used, they must be blank delimited and the list must be enclosed in quotes. Each host match term is either a regular expression that specifies a set of machine names, or the name of a file that contains one or more host match terms with each term on a separate line.

Each machine name that is returned by RSCT is compared to all of the regular expressions that were found by processing the host match terms. If the machine name matches at least one of the regular expressions, the machine

information is included in the output of the **llextrPD** command. If the **-h** flag is not used, all of the machines returned by RSCt are included in the **llextrPD** command output.

Description

This command extracts data for LoadLeveler to set up the administration file. If you plan to use the **llextrPD** command to construct machine and adapter stanzas for the LoadLeveler administration file, RSCt is required.

The **llextrPD** command must be run on one of the nodes in an active RSCt peer domain to obtain the RSCt peer nodes and network interface data from that cluster. If you are not running the command in an active RSCt peer domain you will just get information from the local machine. Adapter stanza names for HPS adapters are not included in the machine stanza alias. If you run an application which requires LoadLeveler to recognize a node by the interface name of a HPS adapter, you must manually add the adapter stanza name for the HPS adapter as an alias in the machine stanza.

Examples

1. The following example extracts the data from an RSCt peer domain:

```
llextrPD -a m10
```

Results:

```
#llextrPD: Cluster = "l1cluster" ID = "0Jt9zGF7nbDWwWjjTDrxjG" on
Mon Oct 16 15:24:13 2006
```

```
c121san10.ppd.pok.ibm.com: type = machine adapter_stanzas =
c121s0n10.ppd.pok.ibm.com c121s1n10.ppd.pok.ibm.com
c121san10.ppd.pok.ibm.com c121f2rp02.ppd.pok.ibm.com
alias = c121f2rp02.ppd.pok.ibm.com
```

```
c121s0n10.ppd.pok.ibm.com: type = adapter
adapter_name = sn0
network_type = switch
interface_address = 192.168.0.10
interface_name = c121s0n10.ppd.pok.ibm.com
multilink_address = 10.10.10.10
logical_id = 2
adapter_type = Switch_Network_Interface_For_HPS
device_driver_name = sni0
network_id = 1
```

```
c121s1n10.ppd.pok.ibm.com: type = adapter
adapter_name = sn1
network_type = switch
interface_address = 192.168.1.10
interface_name = c121s1n10.ppd.pok.ibm.com
multilink_address = 10.10.10.10
logical_id = 0
adapter_type = Switch_Network_Interface_For_HPS
device_driver_name = sn1
network_id = 1
```

```
c121san10.ppd.pok.ibm.com: type = adapter
adapter_name = m10
network_type = multilink
interface_address = 10.10.10.10
interface_name = c121san10.ppd.pok.ibm.com
multilink_list = sn0,sn1
```

```
c121f2rp02.ppd.pok.ibm.com: type = adapter
adapter_name = en0
```

```

network_type = ethernet
interface_address = 9.114.66.74
interface_name = c121f2rp02.ppd.pok.ibm.com
device_driver_name = ent0

```

```

c121san04.ppd.pok.ibm.com: type = machine adapter_stanzas =
c121s0n04.ppd.pok.ibm.com c121s1n04.ppd.pok.ibm.com
c121san04.ppd.pok.ibm.com c121f1rp04.ppd.pok.ibm.com
alias = c121f1rp04.ppd.pok.ibm.com

```

```

c121s0n04.ppd.pok.ibm.com: type = adapter
adapter_name = sn0
network_type = switch
interface_address = 192.168.0.4
interface_name = c121s0n04.ppd.pok.ibm.com
multilink_address = 10.10.10.4
logical_id = 11
adapter_type = Switch_Network_Interface_For_HPS
device_driver_name = sni0
network_id = 1

```

```

c121s1n04.ppd.pok.ibm.com: type = adapter
adapter_name = sn1
network_type = switch
interface_address = 192.168.1.4
interface_name = c121s1n04.ppd.pok.ibm.com
multilink_address = 10.10.10.4
logical_id = 9
adapter_type = Switch_Network_Interface_For_HPS
device_driver_name = sn1
network_id = 1

```

```

c121san04.ppd.pok.ibm.com: type = adapter
adapter_name = m10
network_type = multilink
interface_address = 10.10.10.4
interface_name = c121san04.ppd.pok.ibm.com
multilink_list = sn0,sn1

```

```

c121f1rp04.ppd.pok.ibm.com: type = adapter
adapter_name = en0
network_type = ethernet
interface_address = 9.114.66.68
interface_name = c121f1rp04.ppd.pok.ibm.com
device_driver_name = ent0

```

- The following example extracts the data from an RSCT peer domain for a dynamic adapter configuration:

```
llextrPD -m -a m10
```

Results:

```
#llextrPD: Cluster = "acc97" ID = "28jek7RdrHdGwr5C6zQwWm" on
Mon Oct 16 14:37:33 2006
```

```
c97m10n13.ppd.pok.ibm.com: type = machine
alias = c97n13.ppd.pok.ibm.com
```

```
c97m10n09.ppd.pok.ibm.com: type = machine
alias = c97n09.ppd.pok.ibm.com
```

```
c97m10n01.ppd.pok.ibm.com: type = machine
alias = c97n01.ppd.pok.ibm.com
```

```
c97m10n05.ppd.pok.ibm.com: type = machine
alias = c97n05.ppd.pok.ibm.com
```

- The following example shows an adapter stanza:

llextrPD

```
| .  
| .  
| .  
| c197blade2b11.ppd.pok.ibm.com: type = adapter  
|     adapter_name = eth0  
|     network_type = ethernet  
|     interface_address = 9.114.198.43  
|     interface_name = c197blade2b11.ppd.pok.ibm.com  
| .  
| .  
| .
```

4. The following example would only include information for all the nodes in the RSCD peer domain with names that included work1, work02, work03, and work05:

```
llextrPD -h "work1.* nodelist1.txt"
```

if nodelist1.txt is in the current directory and contains the two lines:

```
work02.*  
/home/steve/nodelist2.txt
```

and /home/steve/nodelist2.txt contains the single line:

```
work0[35].*
```

To further explain this example:

- The pattern work1.* is on the command line, which is the string work1 followed by any character (the dot) zero or more times (the asterisk).
- The pattern work02.* is in nodelist1.txt, which is the string work02 followed by any character (the dot) zero or more times (the asterisk).
- The pattern work0[35].* is in nodelist2.txt, which is the string work0 followed by a 3 or a 5 followed by any character (the dot) zero or more times (the asterisk).

Therefore, any pattern that has work1, work02, work03, or work05 in it will be displayed by the **llextrPD** command.

Security

LoadLeveler administrators and users can issue this command.

llfavorjob - Reorder system queue by job

Purpose

llfavorjob – Sets specified jobs to a higher system priority than all jobs that are not favored. This command also *unfavors* previously favored jobs, restoring the original priority, when you specify the **-u** flag.

Syntax

```
llfavorjob [-?] [-H] [-v] [-q] [-u] joblist
```

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- q Specifies quiet mode: print no messages other than error messages.
- u Unfavors previously favored jobs, requeuing them according to their original priority levels.

joblist

Is a blank-delimited list of job and step identifiers. When a job identifier is specified, the command action is taken for all steps of the job. At least one job or step identifier must be specified.

The format of a job identifier is *host.jobid*. The format of a step identifier is *host.jobid.stepid*.

where:

- *host* is the name of the machine that assigned the job and step identifiers.
- *jobid* is the job number assigned to the job when it was submitted.
- *stepid* is the job step number assigned to the job step when it was submitted.

The job or step identifier may be specified in an abbreviated form, *jobid* or *jobid.stepid*, when the command is invoked on the same machine that assigned the job and step identifiers. In this case, LoadLeveler will use the local machine's hostname to construct the full job or step identifier.

Description

If this command is issued against jobs that are already running, it has no effect. If the job vacates, however, and returns to the queue, the job gets re-ordered with the new priority.

If more than one job is affected by this command, then the jobs are ordered by the **sysprio** expression and are scanned before the not favored jobs. However, favored jobs which do not match the job requirements with available machines may run after not favored jobs. This command remains in effect until reversed with the **-u** option.

Examples

1. This example assigns job steps 12.4 on the machine *iron* and 8.2 on *zinc* the highest priorities in the system, with the job steps ordered by the **sysprio** expression:

llfavorjob

```
llfavorjob iron.12.4 zinc.8.2
```

2. This example unfavors job steps 12.4 on the machine *iron* and 8.2 on the machine *zinc*:

```
llfavorjob -u iron.12.4 zinc.8.2
```

Security

LoadLeveler administrators can issue this command.

llfavoruser - Reorder system queue by user

Purpose

llfavoruser – Sets a user’s jobs to the highest priority in the system, regardless of the current setting of the job priority. Jobs already running are not affected. This command also *unfavors* the user’s jobs, restoring the original priority, when you specify the **-u** flag.

Syntax

```
llfavoruser [-?] [-H] [-v] [-q] [-u] userlist
```

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- q Specifies quiet mode: print no messages other than error messages.
- u Unfavors previously favored users, reordering their jobs according to their original priority levels. If **-u** is **not** specified, the user’s jobs are favored.

userlist

Is a blank-delimited list of users whose jobs are given the highest priority. If **-u** is specified, *userlist* jobs are *unfavored*.

Description

This command affects your current and future jobs until you remove the favor.

When the central manager daemon is restarted, any favor applied to users is revoked.

The user’s jobs still remain ordered by user priority (which may cause jobs for the user to swap **sysprio**). If more than one user is affected by this command, the jobs of favored users are ordered by **sysprio** and are scanned before the jobs of not favored users. However, jobs of favored users which do not match job requirements with available machines may run after jobs of not favored users.

Examples

1. This example grants highest priority to all queued jobs submitted by users ellen and fred according to the **sysprio** expression:


```
llfavoruser ellen fred
```
2. This example unfavors all queued jobs submitted by users ellen and fred:


```
llfavoruser -u ellen fred
```

Security

LoadLeveler administrators can issue this command.

llfs - Fair share scheduling queries and operations

Purpose

llfs – Returns information about fair share scheduling in LoadLeveler or operates on fair share scheduling.

Syntax

```
llfs [ -? | -H | -v | -s savendir | -r [saved_file] | [-u user_list] [-g group_list] ]
```

Flags

-?	Provides a short usage message.
-H	Provides extended help information.
-v	Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
-s <i>savendir</i>	Saves a snapshot of the historic fair share data in the specified directory. The central manager must be able to write to the <i>savendir</i> directory. The directory does not need to be accessible on the node where the command is issued.
-r [<i>saved_file</i>]	Resets fair share scheduling by either clearing all historic fair share data, or restoring fair share scheduling to a state corresponding to a file previously saved by the llfs -s command. The <i>saved_file</i> file must be readable by the central manager. A full path name should be specified for the file. The file does not need to be accessible on the node where the command is issued.
-u <i>user_list</i>	Is a blank-delimited list of users. Fair share allocation and utilization information for these users will be displayed.
-g <i>group_list</i>	Is a blank-delimited list of LoadLeveler groups. Fair share allocation and utilization information for these groups will be displayed.

Description

The **llfs** command can be used either as a query command or as a control command.

As a query command, **llfs** with no options displays current fair share scheduling information for all users and groups who have jobs completed in the LoadLeveler cluster. With the **-u** or **-g** options, information for the users or groups requested is displayed.

The **llfs** command displays the following information as a query command:

- The time the information is obtained
- The total number of shares that both the cluster CPU and Blue Gene resources are divided into (for example, if there are 100 shares, then there are 100 CPU shares and 100 Blue Gene shares).
- The time interval in which most of the used shares are consumed
- The number of shares allocated and used for each user and group

Note: Used shares are for the cluster CPU resources and the Blue Gene used shares are for the Blue Gene resources. Blue Gene shares are only displayed when Blue Gene is enabled.

The share usage affects the job scheduling priority according to the **SYSPRIO** expression used in the global configuration file.

As a control command, you can use the **lfs** to reset fair share scheduling or save historic fair share data to a file. The **-s** option saves a snapshot of all historic fair share data in the LoadLeveler cluster. If the value of **FAIR_SHARE_INTERVAL** is large (for example, an entire year) and if there are concerns about losing a lot of historic data, this option can be used on a regular basis to save a snapshot of the historic fair share data to a file. The saved file can be used by the **lfs -r** option to restore fair share scheduling to the state corresponding to the saved file. Without a saved file, the **lfs -r** option will discard all previous historic fair share data and restarts fair share scheduling again.

This command is for LoadLeveler administrators and the BACKFILL scheduler only.

Examples

When **BG_ENABLED** is set to **false** in the configuration file, the following examples apply.

1. To display current fair share information without any options, issue:

```
lfs
```

You should receive output similar to the following:

```
Current Time: Fri Jan 20 13:45:55 EST 2006
FAIR_SHARE_TOTAL_SHARES = 100
FAIR_SHARE_INTERVAL = 180 hours
```

Name	Type	Allocated Shares	Used Shares
uno	user	50	20
zhong	user	10	10
load1	user	0	4
systemst	group	0	30
load1	group	0	4

2. To sort fair share information according to used shares in descending order, issue:

```
lfs > /tmp/out; head -6 /tmp/out; tail +7 /tmp/out |sort -rn +3
```

You should receive output similar to the following:

```
Current Time: Fri Jan 20 13:45:55 EST 2006
FAIR_SHARE_TOTAL_SHARES = 100
FAIR_SHARE_INTERVAL = 180 hours
```

Name	Type	Allocated Shares	Used Shares
systemst	group	0	30
uno	user	50	20
zhong	user	10	10
load1	user	0	4
load1	group	0	4

3. To display current fair share information for user **uno** and group **systemst**, issue:

```
lfs -u uno -g systemst
```

You should receive output similar to the following:

Name	Type	Allocated Shares	Used Shares
uno	user	50	20
systest	group	0	30

- To reset fair share scheduling by discarding all previous historic data, issue:
`llfs -r`
 You should receive a response similar to the following:
`llfs: request has been sent to LoadLeveler.`
- To save a snapshot of the historic data to the `/shared/save` directory, issue:
`llfs -s /shared/save`
 You should receive a response similar to the following:
`llfs: /shared/save/fair_share_data.200601201345 has been created.`
- To restart fair share scheduling from previously saved data, issue:
`llfs -r /shared/save/fair_share_data.200601201345`
 You should receive a response similar to the following:
`llfs: request has been sent to LoadLeveler.`

When **BG_ENABLED** is set to **true** in the configuration file, the following examples apply.

- To display current fair share information without any options, issue:
`llfs`
 You should receive output similar to the following:

```
Current Time: Tue 13 Jun 2006 06:35:24 PM CDT
FAIR_SHARE_TOTAL_SHARES = 100000
FAIR_SHARE_INTERVAL = 1 hours
```

Name	Type	Allocated Shares	Used Shares	BG	Used Shares
varella	user	0	0		0
No_Group	group	0	16		595
ezhong	user	0	16		595
- To sort fair share information according to the used shares of the cluster CPU resources in descending order, issue:
`llfs>/tmp/out; head -6 /tmp/out;tail +7 /tmp/out|sort -rn +4`
 You should receive output similar to the following:

```
Current Time: Tue 13 Jun 2006 06:38:03 PM CDT
FAIR_SHARE_TOTAL_SHARES = 100000
FAIR_SHARE_INTERVAL = 1 hours
```

Name	Type	Allocated Shares	Used Shares	BG	Used Shares
No_Group	group	0	14		521
ezhong	user	0	14		521
varella	user	0	0		0
- To display current fair share information for user **ezhong** and group **group**, issue:
`llfs -u ezhong -g group`
 You should receive output similar to the following:

```
Current Time: Tue 13 Jun 2006 06:39:34 PM CDT
FAIR_SHARE_TOTAL_SHARES = 100000
FAIR_SHARE_INTERVAL = 1 hours
```

Name	Type	Allocated Shares	Used Shares	BG	Used Shares
ezhong	user	0	13		483
group	group	0	0		0

Security

LoadLeveler administrators can issue this command.

llhold - Hold or release a submitted job

Purpose

llhold – Places jobs in user hold or system hold and releases jobs from both types of hold. Users can only move their own jobs into and out of user hold. Only LoadLeveler administrators can move jobs into and release them from system hold.

Syntax

```
llhold      [-?] [-H] [-v] [-q] [-s] [-r] [-X cluster_name]
           [-u userlist] [-h hostlist] [joblist]
```

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- q Specifies quiet mode: print no messages other than error messages.
- s Puts jobs in system hold. Only a LoadLeveler administrator can use this option.

If neither **-s** nor **-r** is specified, LoadLeveler puts the jobs in user hold.

- r Releases a job from hold. A job step in user hold can be released by the owner or a LoadLeveler administrator. A job step in system hold can only be released by a LoadLeveler administrator. If a job step that is in both system hold and user hold is released by a LoadLeveler administrator, the job step will be released from system hold but remains in user hold. If the owner releases a job step that is in both system hold and user hold, the job step is released from user hold but remains in system hold.

The **-r** flag can be used to restart jobs that were put on user hold by the **llckpt -u** command.

If neither **-s** nor **-r** is specified, LoadLeveler puts the jobs in user hold.

- X *cluster_name*
Specifies the name of a single cluster where the command is to run.

- u *userlist*
Is a blank-delimited list of users. When used with the **-h** option, only the user's jobs monitored on the machines in the *hostlist* are held or released. When used alone, only the user's jobs monitored on the Schedd machine are held or released.

- h *hostlist*
Is a blank-delimited list of machine names. All jobs monitored on machines in this list are held or released. When issued with the **-u** option, the *userlist* is used to further select jobs for holding or releasing.

When issued by a non-administrator, this option only acts upon jobs that user has submitted to the machines in *hostlist*.

When issued by an administrator, all jobs monitored on the machines are acted upon unless the **-u** option is also used. In that case, the *userlist* is also part of the selection process, and only jobs both submitted by users in *userlist* and monitored on the machines in the *hostlist* are acted upon.

joblist

Is a blank-delimited list of job and step identifiers. When a job identifier is specified, the command action is taken for all steps of the job. At least one job or step identifier must be specified.

The format of a job identifier is *host.jobid*. The format of a step identifier is *host.jobid.stepid*.

where:

- *host* is the name of the machine that assigned the job and step identifiers.
- *jobid* is the job number assigned to the job when it was submitted.
- *stepid* is the job step number assigned to the job step when it was submitted.

The job or step identifier may be specified in an abbreviated form, *jobid* or *jobid.stepid*, when the command is invoked on the same machine that assigned the job and step identifiers. In this case, LoadLeveler will use the local machine's hostname to construct the full job or step identifier.

Note: For coscheduled jobs, even if all coscheduled job steps are not in the list of targeted job steps, the requested operation is performed on all coscheduled job steps.

Description

This command does not affect a job step that is running unless the job step attempts to enter the Idle state. At this point, the job step is placed in the Hold state.

To ensure a job is released from both system hold and user hold, the administrator must issue the command with `-r` specified to release it from system hold. The administrator or the submitting user can reissue the command to release the job from user hold.

This command will fail if:

- A nonadministrator attempts to move a job into or out of system hold.
- A nonadministrator attempts to move a job submitted by someone else into or out of user hold.

Examples

1. This example places job 23, job step 0 and job 19, job step 1 on hold:
`llhold 23.0 19.1`
2. This example releases job 23, job step 0, job 19, job step 1, and job 20, job step 3 from a hold state:
`llhold -r 23.0 19.1 20.3`
3. This example places all jobs from users abe, barbara, and carol2 in system hold:
`llhold -s -u abe barbara carol2`
4. This example releases from a hold state all jobs on machines bronze, iron, and steel:
`llhold -r -h bronze iron steel`
5. This example releases from a hold state all jobs on machines bronze, iron, and steel that smith submitted:
`llhold -r -u smith -h bronze iron steel`

llhold

Results

The following shows a sample system response for the **llhold -r -h bronze** command:

```
llhold: Hold command has been sent to the central manager.
```

Security

LoadLeveler administrators and users can issue this command.

llinit - Initialize machines in the LoadLeveler cluster

Purpose

llinit – Initializes a new machine as a member of the LoadLeveler cluster

Syntax

```
llinit      [-?] [-H] [-q] [-prompt] [-local pathname] [-release pathname]
           [-cm machine] [-debug]
```

Flags

-? Provides a short usage message.

-H Provides extended help information.

-q Specifies quiet mode: print no messages other than error messages.

-prompt

Prompts or leads you through a set of questions that help you to complete the **llinit** command.

-local *pathname*

pathname is the local directory in which the spool, execute, and log subdirectories will be created. The default, if this flag is not used, is the home directory.

There must be a unique local directory for each LoadLeveler cluster member.

-release *pathname*

pathname is the release directory, where the LoadLeveler bin, lib, man, include, and samples subdirectories are located. The default, if this flag is not used, is the **/usr/lpp/LoadL/full** directory on AIX or the **/opt/ibmll/LoadL/full** directory on Linux.

-cm *machine*

machine is the central manager machine, where the negotiator daemon runs.

-debug

Displays debug messages during the execution of **llinit**.

Description

This command runs once on each machine during the installation process. It must be run by the user ID you have defined as the LoadLeveler user ID. The log, spool, and execute directories are created with the correct modes and ownerships. The LoadLeveler configuration and administration files, **LoadL_config** and **LoadL_admin**, respectively, are copied from LoadLeveler's release directory to LoadLeveler's home directory. The local configuration file, **LoadL_config.local**, is copied from LoadLeveler's release directory to LoadLeveler's local directory.

llinit initializes a new machine as a member of the LoadLeveler cluster by doing the following:

- Creates the following LoadLeveler subdirectories with the given permissions:
 - **spool** subdirectory, with permissions set to 700.
 - **execute** subdirectory, with permissions set to 1777.
 - **log** subdirectory, with permissions set to 775.
- Copies the **LoadL_config** and **LoadL_admin** files from the release directory samples subdirectory into the home directory of the LoadLeveler user ID.

llinit

- Copies the **LoadL_config.local** file from the release directory samples subdirectory into the local directory.
- Creates symbolic links from the loadl home directory to the spool, execute, and log subdirectories and the **LoadL_config.local** file in the local directory (if home and local directories are not identical).
- Creates symbolic links from the home directory to the bin, lib, man, samples, and include subdirectories in the release directory.
- Updates the **LoadL_config** with the release directory name.
- Updates the **LoadL_admin** with the central manager machine name.

Before running **llinit** ensure that your HOME environment variable is set to LoadLeveler's home directory. To run **llinit**, you must have:

- Write privileges in the LoadLeveler home directory
- Write privileges in the LoadLeveler release directory
- Write privileges in the LoadLeveler local directory.

Examples

The following example initializes a machine, assigning **/var/loadl** as the local directory, **/usr/lpp/LoadL/full** as the release directory, and the machine named **bronze** as the central manager.

```
llinit -local /var/loadl -release /usr/lpp/LoadL/full -cm bronze
```

Ensure that the local directory exists before running the preceding command.

Results

The command:

```
llinit -local /home/ll_admin -release /usr/lpp/LoadL/full -cm mars
```

will yield the following output:

```
llinit: creating directory "/home/ll_admin/spool"
llinit: creating directory "/home/ll_admin/log"
llinit: creating directory "/home/ll_admin/execute"
llinit: set permission "700" on "/home/ll_admin/spool"
llinit: set permission "775" on "/home/ll_admin/log"
llinit: set permission "1777" on "/home/ll_admin/execute"
llinit: creating file "/home/ll_admin/LoadL_admin"
llinit: creating file "/home/ll_admin/LoadL_config"
llinit: creating file "/home/ll_admin/LoadL_config.local"
llinit: editing file /home/ll_admin/LoadL_config
llinit: editing file /home/ll_admin/LoadL_admin
llinit: creating symbolic link "/home/ll_admin/bin -> \
/usr/lpp/LoadL/full/bin"
llinit: creating symbolic link "/home/ll_admin/lib -> \
/usr/lpp/LoadL/full/lib"
llinit: creating symbolic link "/home/ll_admin/man -> \
/usr/lpp/LoadL/full/man"
llinit: creating symbolic link "/home/ll_admin/samples -> \
/usr/lpp/LoadL/full/samples"
llinit: creating symbolic link "/home/ll_admin/include -> \
/usr/lpp/LoadL/full/include"
llinit: program complete.
```

Security

LoadLeveler administrators can issue this command.

llmkres - Make a reservation

Purpose

llmkres – Creates a LoadLeveler reservation. A set of nodes can be reserved in advance for a period of time to run both interactive and batch jobs. For additional information on running interactive jobs with reservations, see Chapter 8, “Building and submitting jobs,” on page 165.

Syntax

```
llmkres { -? | -H | -v | [-q] -t start_time -d duration { -n number_of_nodes |
-h host_list | -h free | -j job_step | -f job_command_file |
-c number_of_bg_cnodes} [-U user_list] [-G group_list] [-s {yes | no}]
[-i {yes | no}] [-g group] }
```

Flags

-?	Provides a short usage message.
-H	Provides extended help information.
-v	Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
-q	Specifies quiet mode: print no messages other than error messages.
-t <i>start_time</i>	Specifies the start time of the reservation using the format [mm/dd[/[cc]yy]] HH:MM. Hours must be specified using a 24-hour clock.
-d <i>duration</i>	Specifies the duration of the reservation in minutes.
-n <i>number_of_nodes</i>	Specifies the number of nodes to reserve. To reserve a number of Blue Gene C-nodes, use the -c flag.
-h free	
-h <i>host_list</i>	Specifies a blank-delimited list of machines to reserve. To reserve every node in the LoadLeveler cluster, use the -h <i>host_list</i> flag. If any one of the nodes cannot be reserved, the request will fail. Specifying the reserved word free reserves all machines available for this reservation, which currently have an active LoadL_startd daemon. The reservation change request will succeed if at least one node can be included in the reservation.
-j <i>job_step</i>	Specifies a job step whose requirements will be used to determine what resources to reserve. The job step must be in an idle-like state and takes the form [<i>host.</i>] <i>jobid.stepid</i> .
	where:
	<ul style="list-style-type: none"> • <i>host</i> is the name of the machine that assigned the job and step identifiers. • <i>jobid</i> is the job number assigned to the job when it was submitted. • <i>stepid</i> is the job step number assigned to the job step when it was submitted.

llmkres

The step identifier may be specified in an abbreviated form, *jobid.stepid*, when the command is invoked on the same machine that assigned the step identifier. In this case, LoadLeveler will use the local machine's host name to construct the full step identifier.

You must be an administrator or the job step owner to make this request. If the request to make the reservation is successful, the job step will be bound to the reservation. If the request is not successful, there is no change to the status of the job step.

Note: If the job step is of type **blue_gene**, then only Blue Gene compute nodes (C-nodes) will be reserved to satisfy the Blue Gene resource requirements of the job step.

-f *job_command_file*

Specifies the path to a *job_command_file* that will be submitted and the first job step used to determine what resources to reserve. All job steps will be bound to the reservation, or if the reservation request fails, be placed in the **NotQueued** state. The job ID of the newly created job will be displayed.

-c *number_of_bg_cnodes*

Specifies the number of C-nodes to reserve in the Blue Gene system. The shape of the allocated resource for a given size cannot be guaranteed, but the size of the allocated shape will be no smaller than the requested size and will be as close to the requested size as possible.

-U *user_list*

Specifies a blank-delimited list of users who can use the reservation.

-G *group_list*

Specifies a blank-delimited list of LoadLeveler groups whose users can use the reservation.

-s {**yes** | **no**}

Specifies if the SHARED option is selected for the reservation. For a SHARED reservation, after all bound job steps that can run on the reserved nodes are scheduled to run, the remaining resources can be used to run job steps not bound to the reservation. Only bound job steps can be scheduled to run on a reservation that is not shared. The default is not to share the reservation.

-i {**yes** | **no**}

Specifies if the REMOVE_ON_IDLE option is selected for the reservation. For a REMOVE_ON_IDLE reservation, if all bound job steps are finished or if all bound job steps are Idle and none can run on the reserved nodes, the reservation will be removed (canceled) automatically by LoadLeveler. If this option is not set, the reservation will remain, regardless of being used or not. The default is not to remove the reservation automatically.

-g *group*

Specifies a LoadLeveler group that will own the reservation. The default is what is specified as the *default_group* in the user stanza or No_Group. Ownership of a reservation by a group does not imply that all of the members of the group can use the reservation, but rather is used as a count toward the maximum number of reservations that a group can own.

Description

The **llmkres** command is for users authorized by LoadLeveler administrators. The user ID requesting the creation of a reservation becomes the owner of the

reservation and can use the reservation. A unique reservation ID will be displayed upon successful creation of the reservation, otherwise a message will be printed out to indicate a failure.

Note that it is possible for a time out to occur while this command is waiting for a response from the LoadLeveler central manager. Even if a time out occurs or the command process is killed, the command may still succeed. To determine if the request has been granted, issue the **llqres** command.

The owner of a reservation maintains certain privileges beyond those allowed for users of the reservation. The owner of a reservation and LoadLeveler administrators can always use the reservation. The owner of a reservation (and the LoadLeveler administrator) can cancel or change a reservation. Users of a reservation are allowed to bind their jobs to a reservation. When a reservation is created, the **-U** and **-G** flag can be used to specify who can use the reservation.

This command is for the BACKFILL scheduler only.

A coscheduled job step cannot be specified when using the **-j** or **-f** flags.

A job step that requests a size that fits into the size of a Blue Gene reservation might not always be able to run inside the reservation because of the internal topology of the Blue Gene system. For example, if a Blue Gene reservation was created using a Blue Gene job that requested 1048 compute nodes with a connection type of **MESH**, then a job requesting 1048 compute nodes with a connection type of **TORUS** may not be able to run inside that reservation.

Examples

1. To reserve 3 nodes for 2 hours starting at 2 p.m. of the current year allowing members of the LoadLeveler group **loadlusr** to use the reservation, issue:

```
llmkres -t 01/16 14:00 -d 120 -n 3 -G loadlusr
```

Note that if you specify a date that has already passed in the current year, you must include the year or an error will occur.

You should receive a response similar to the following:

```
The reservation c94n16.pok.ibm.com.20.r has been successfully made.
```

2. To reserve nodes based on the requirements of one user job, issue:

```
llmkres -t 10/16/2006 02:00 -d 420 -f weather.cmd -i yes
```

You should receive a response similar to the following:

```
The job "c94n16.pok.ibm.com.25" has been submitted.
```

```
The reservation c94n16.pok.ibm.com.31.r has been successfully made.
```

3. To reserve two nodes for use by the reservation owner and two additional users issue:

```
llmkres -t 01/17 13:30 -d 240 -h c94n01 c94n16 -U jay chris
```

You should receive a response similar to the following:

```
The reservation c94n16.pok.ibm.com.55.r has been successfully made.
```

4. To reserve 1024 Blue Gene C-nodes, issue:

```
llmkres -t 18:00 -d 60 -c 1024
```

You should receive a response similar to the following:

llmodify - Change attributes of a submitted job step

Purpose

llmodify – Changes the attributes or characteristics of a submitted job.

Syntax

```
llmodify      { -? | -H | -v | [-q] [-X cluster_name]
                | -c consumable_cpus | -m consumable_real_memory |
                -W wclimit_add_min | -C job_class | -a account_no |
                -s q_sysprio | -p {preempt | nopreempt} |
                -k keyword=value } jobstep }
```

Flags

- ?** Provides a short usage message.
- H** Provides extended help information.
- v** Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- q** Specifies quiet mode: print no messages other than error messages.
- X** *cluster_name*
Specifies the name of a single cluster where the command is to run. This flag cannot be specified with the **-p**, **-s**, **-W**, or **-x** flags.
- c** *consumable_cpus*
Specifies the consumable CPU value for an idle-like job step.

Allows the ConsumableCpus resource requirement to be reset to the specified value. This value can be any integer equal to or greater than zero (0) and should follow the rules for the **resources** keyword in the job command file.
- m** *consumable_real_memory*
Specifies the consumable real memory value for an idle-like job step.

Allows the ConsumableMemory resource requirement to be reset to the specified value. No units should be specified, as megabytes (MB) is assumed. This value can be any integer greater than zero (0) and should follow the rules for the **resources** keyword in the job command file.
- W** *wclimit_add_min*
Specifies additional time in minutes to add to the wall clock limits of a running-like job step. This option is for preventing a job step from being killed due to the wall clock limits. This is a LoadLeveler administrator only option.

Both the hard limit and soft limit are increased by the specified value. This value can be any integer greater than 0.

The increase will only be effective if a limit was originally set and not already exceeded. If you attempt to modify the wall clock limit for a job step that is approaching its current wall clock limit, it is possible for the current wall clock limit to expire before it can be changed.

Specifying **llmodify -W** will fail if the wall clock time of the job step is extended beyond the start time of any reservation that reserves resources currently used by the job step. The reservations in conflict must be canceled before the request to increase the job step's wall clock limit can be granted.

lmodify

-C *job_class*

Specifies the job class name.

Allows the job class name to be reset to the specified value for an idle-like job step. This value can be any string without white spaces.

-a *account_no*

Specifies the account number.

Allows the account number to be reset to the specified value for an idle-like job step.

-s *q_sysprio*

Specifies the job step priority.

This option allows the *q_sysprio* for a job step to be reset to the specified integer value. The new job step priority will be fixed. Once the priority has been modified, it will no longer be changed if the central manager recalculates priorities. This is a LoadLeveler administrator only option.

-p {**preempt**|**nopreempt**}

Specifies whether a job is preemptable or nonpreemptable.

-k *keyword=value*

Modifies *keyword* to the new *value* provided.

where *keyword* is one of the following:

account_no

Changes the account number to the specified value for an idle-like job step.

bg_connection

Changes the connection option of an idle-like Blue Gene job. The subsequent *value* argument must be a string that is either **TORUS**, **MESH**, or **PREFER_TORUS**. **bg_connection** cannot be modified if **bg_partition** is already specified.

bg_partition

Changes the requested partition ID of an idle-like Blue Gene job. If this value is specified, **bg_requirements**, **bg_connection**, **bg_shape**, **bg_size**, and **bg_rotate** will be reset to their default values.

bg_requirements

Changes the memory requirement that a Blue Gene base partition in the LoadLeveler cluster must meet to run an idle-like Blue Gene job. The subsequent *value* option must be an expression. Memory is the only variable that is supported. **bg_requirements** cannot be modified if **bg_partition** is already specified.

bg_rotate

Changes the rotate option of an idle-like Blue Gene job. The subsequent *value* argument must be a string that is either **True** or **False**. **bg_rotate** cannot be modified if **bg_partition** is already specified.

bg_shape

Changes the shape of an idle-like Blue Gene job. The subsequent *value* argument must be of the form "*XxYxZ*", where *X*, *Y*, and *Z* are integers in units of the number of base partitions. If this value is specified, any value previously specified for **bg_size** or **bg_partition** will be reset to its default value.

bg_size

Changes the size of an idle-like Blue Gene job. The subsequent *value* argument must be an integer in units of compute nodes. If this value is specified, any value previously specified for **bg_shape** or **bg_partition** will be reset to its default value.

class Changes the job class name to the specified value for an idle-like job step. The value can be any string without white spaces.

consumableCpus

Changes the ConsumableCpus resource requirement to the specified value. The value can be any integer equal to or greater than zero (0).

consumableMemory

Changes the ConsumableMemory resource requirement to the specified value. No units should be specified, as megabytes (MB) is assumed. This value can be any integer greater than zero (0).

preemptable

Specifies whether a job is preemptable or nonpreemptable. The value must be either **yes** or **no**.

sysprio

Changes the *q_sysprio* for a job step to the specified integer value. The new job step priority will be fixed. This is a LoadLeveler administrator only option.

wlimit_add

Increases the wall clock limit of a running-like job step by the number of specified minutes. This is a LoadLeveler administrator only option.

jobstep

Is the name of a job step to be modified.

The format of a full LoadLeveler step identifier is *host.jobid.stepid*.

where:

- *host* is the name of the machine that assigned the job and step identifiers.
- *jobid* is the job number assigned to the job when it was submitted.
- *stepid* is the job step number assigned to the job step when it was submitted.

The step identifier may be specified in an abbreviated form, *jobid.stepid*, when the command is invoked on the same machine that assigned the step identifier. In this case, LoadLeveler will use the local machine's hostname to construct the full step identifier.

Description

All options are for the job step owner or a LoadLeveler administrator on an idle-like job step with the following exceptions:

- **-p**, **-s**, and **-W** are LoadLeveler administrator only options
- **-W** is valid only for a job step in a running-like state

A request to mark a job step nonpreemptable will fail if the job step's expected end time extends into an existing reservation.

At the time a job step is modified, LoadLeveler does not check to make certain that the job step with the modified values can be scheduled to run.

To determine if a modification request is successful, issue the **llq -x -l** command and check the fields shown (see Table 85 on page 438) in the output.

llmodify

Table 85. llmodify options and keywords

Options and keywords	Field to check
-a or -k <i>account_no</i>	Account
-C or -k <i>class</i>	Class
-c or -k <i>consumableCpus</i>	Resources
-m or -k <i>consumableMemory</i>	Resources
-p or -k <i>preemptable</i>	Preemptable
-k <i>bg_connection</i>	Wiring Requested
-k <i>bg_partition</i>	Partition Requested
-k <i>bg_requirements</i>	BG Requirements
-k <i>bg_rotate</i>	Rotate
-k <i>bg_shape</i>	Shape Requested
-k <i>bg_size</i>	Size Requested
-s or -k <i>priority</i>	q_sysprio
-W or -k <i>wclimit_add</i>	Wall Clk Hard Limit/Wall Clk Soft Limit

An idle-like state is one of the following job states:

- Idle
- Deferred
- User Hold
- System Hold
- User & System Hold
- Not Queued
- Vacated
- Rejected

A running-like state is one of the following job states:

- Checkpointing
- Pending
- Preempted
- Preempt Pending
- Resume Pending
- Running
- Starting

Examples

1. This example puts the job step c163n07.12.0 in a non-preemptable state:
`llmodify -p nopreempt c163n07.12.0`
2. To extend the wall clock limits of job step c163n07.12.0 by 30 minutes:
`llmodify -W 30 c163n07.12.0`
3. To change the shape of job step c193n04.11.0 to 4x5x4 base partitions:
`llmodify -k bg_shape=4x5x4 c193n04.11.0`
4. To change the connection type of job step c193n04.11.0 to **MESH**:
`llmodify -k bg_connection=MESH c193n04.11.0`
5. This example changes the ConsumableCpus resource requirement of job step c188f2n08.15.0 in cluster1 to the value 3:
`llmodify -X cluster1 -c 3 c188f2n08.15.0`

Results

The following shows a sample system response for **llmodify -s 109 c163n07.12.0**:
llmodify: request has been sent to LoadLeveler.

llmodify returns the following exit values:

- 0 The command ran successfully.
- 1 An error occurred.

Security

LoadLeveler administrators and users can issue this command.

llmovejob - Move a single idle job from the local cluster to another cluster

Purpose

llmovejob – Moves a single idle-like job from the local cluster to a remote cluster.

Syntax

```
llmovejob { -? | -H | -v | -C cluster_name -j job_ID }
```

Flags

- | | |
|------------------------|--|
| -? | Provides a short usage message. |
| -H | Provides extended help information. |
| -v | Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release. |
| -C <i>cluster_name</i> | Indicates the remote cluster the job specified by <i>job_ID</i> should be transferred to. The -C flag must be specified in combination with the -j flag. |
| -j <i>job_ID</i> | Indicates the <i>job_ID</i> to be transferred to the cluster specified by <i>cluster_name</i> . The -j flag must be specified in combination with the -C flag. |

Description

The **llmovejob** command moves a single idle-like job from one cluster to another. Upon successful transfer, the job's submitting owner is notified of the move by mail. If any steps within the job are not idle, the transfer request is rejected and the command exits with an error message. The remote job retains the original *job_ID* from the local cluster. Upon transfer, the remote cluster performs any user mapping and remote job filtering necessary for the job.

Any changes made to the idle job in the local cluster by the **llmodify** command will not be carried forward to the remote cluster. Any jobs submitted when the local cluster was not configured as a part of a multicluster cannot be moved once the cluster converts to a multicluster environment.

Prior to moving the job, the administrator can examine the LoadLeveler statements in the job command file using the **llq -x -d** command. This will show what values the moved job will use during submission to the remote cluster.

Only administrators can issue the **llmovejob** command. In a mixed operating system multicluster environment, administrators must ensure the binary compatibility of the job being transferred.

Standard Error

An error message is issued and the command exits for the following error cases:

- The cluster name is unknown
- The requested cluster cannot be accessed
- The job is not in the Idle state
- The job is unknown

- The **-C** and **-j** flags were not specified together

Examples

This example moves the idle job **silver.11** from the local cluster to the remote cluster **cluster1**:

```
llmovejob -C cluster1 -j silver.11
```

You should receive output similar to the following:

```
Job silver.11 has been submitted to cluster cluster1
```

Security

LoadLeveler administrators can issue this command.

llmovespool - Move job records

Purpose

llmovespool – Moves the job records from the spool of one managing Schedd to another managing Schedd in the local cluster.

Syntax

```
llmovespool { -? | -H | -v | [-d directory] -h hostname }
```

Flags

-?	Provides a short usage message.
-H	Provides extended help information.
-v	Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
-d <i>directory</i>	Indicates the directory containing the job queue whose job records are to be moved. If not specified, the job queue in the current working directory will be moved. The specified directory must be accessible from the machine upon which the command is issued.
-h <i>hostname</i>	Indicates the hostname of the machine running the Schedd daemon, which manages the job queue database that will receive the job records being moved. This flag has no default and is required.

Description

The **llmovespool** command is intended for recovery purposes only. The command moves the job records from the spool of one managing Schedd to another managing Schedd in the local cluster. The command must be run from a machine that has read and write access to the specified **spool** directory containing the job records being moved. This machine must also have network connectivity to the machine running the Schedd daemon that will receive the job records. Jobs to be moved can be in any state.

The Schedd that created the job records to be moved must not be running during the move operation. Jobs within the job queue database will be unrecoverable if the job queue is updated during the move by any process other than the **llmovespool** command. The Schedd that created the job records to be moved must have the **schedd_fenced** machine stanza keyword set to **true** prior to the **llmovespool** command being issued.

All moved jobs retain their original job identifiers. The **llmovespool** command reports the status of each job as it is processed. When the job records for a job are successfully transferred, the **schedd_host** of the job is updated to represent the new managing Schedd and the job records in the specified **spool** directory are deleted. The successful status is reported to standard output. If the transfer for any step within a job fails, the job records for that step remain in the specified **spool** directory and the error status is reported to standard error. If for some reason a job fails, the **llmovespool** command should be reissued against the specified **spool** directory to reprocess the job.

The **l1movespool** command does not move the reservation queue or fair share scheduling data found within the specified **spool** directory.

The command can be issued by administrators only.

Standard Error

An error message is issued and the command exits for the following error cases:

- The command was not issued with the required **-h** flag.
- The machine stanza for the machine running the Schedd daemon, which manages the job queue database that is receiving the job records, has the **schedd_fenced** keyword set to **true**.
- The machine stanza for the machine running the Schedd daemon, which manages the job queue database being moved, does not have the **schedd_fenced** keyword set to **true**.
- The specified hostname is not a valid machine.
- The command cannot make a connection to the specified hostname.
- The specified directory does not exist.
- The job records within the specified directory cannot be accessed.
- There are no job records within the specified **spool** directory.
- The Schedd on the specified hostname cannot accept the transferred job because a job with the same job identifier already exists.

Examples

This example moves the job records found in **/tmp/tmp_spool** to the Schedd running on the **c188f2n08** machine:

```
l1movespool -d /tmp/tmp_spool -h c188f2n08
```

You should receive output similar to the following:

```
The job spool records in /tmp/tmp_spool are being moved to c188f2n08.  
The records for job c188f2n02.ppd.pok.ibm.com.1 were successfully transferred.
```

```
The transfer is complete.
```

Security

LoadLeveler administrators can issue this command.

llpreempt - Preempt a submitted job step

Purpose

llpreempt – Preempts the job steps specified in the *joblist* argument using the preempt method specified in the *preempt_method* argument or resumes the jobs steps specified in the *joblist* argument. Only jobs that have been preempted with the preempt method of suspend through the **llpreempt** command or the **ll_preempt** subroutine can be resumed with this command. The **llpreempt** command cannot resume a job step that was preempted through the PREEMPT_CLASS rules or a job step that was preempted with a preempt method other than suspend.

Syntax

```
llpreempt    -? | -H | -v | [-q] [ -r | -m method] { [-u userlist]
              [-h hostlist] | [joblist] }
```

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- q Specifies quiet mode: print no messages other than error messages.
- r Resumes the specified jobs. This option is valid only for jobs that were preempted by the suspend method.
- m *preempt_method*
 - su Indicates preempted jobs that are to be suspended. Suspended jobs will stay in the preempted state until the action is undone with the **-r** flag. This is the default.

Preemption using the suspend method is not supported by the LoadLeveler for Linux platforms. On these platforms, the **llpreempt** command will have no effect if the suspend method is specified either explicitly as a command line option (**-m su**), or implicitly through the **default_preempt_method = su** configuration keyword. Note that **su** is the default value of the **default_preempt_method** keyword.

 - vc** Indicates that preempted jobs are to be vacated. The preempted jobs will be terminated and remain in the job queue. The job will be rescheduled to run as soon as resources for the job are available.
 - rm** Indicates that preempted jobs are to be removed. The preempted jobs will be terminated and removed from the job queue. In order to rerun the job, you must resubmit the job to LoadLeveler.
 - sh** Indicates that preempted jobs are to be put into system hold. The preempted jobs will be terminated and remain in the job queue in system hold state. The jobs will remain in system hold until released by a LoadLeveler administrator using the **llhold** command. After being released, the job will go into the idle state where it will be rescheduled to run as soon as resources for the job are available.
 - uh** Indicates that preempted jobs are to be put into user hold. The preempted jobs will be terminated and remain in the job queue in user

hold state. The jobs will remain in the user hold until released by the owner of the job step or by a LoadLeveler administrator using the **llhold** command. After being released, the job will go into the idle state where it will be rescheduled to run as soon as resources for the job are available.

-u *userlist*

Specifies a blank-delimited list of user names. When used with the **-h** option, only the user's job steps monitored on the machines in the *hostlist* are preempted. When used alone, only the user's jobs monitored by the machine issuing the command are preempted.

-h *hostlist*

Specifies a blank-delimited list of host names. All job steps monitored by these hosts are preempted. When used with the **-u** option, only the specified user's job steps monitored by these hosts are preempted.

joblist

Is a blank-delimited list of job and step identifiers to be preempted. When a job identifier is specified, the command action is taken for all steps of the job. At least one job or step identifier must be specified.

The format of a job identifier is *host.jobid*. The format of a step identifier is *host.jobid.stepid*.

where:

- *host* is the name of the machine that assigned the job and step identifiers.
- *jobid* is the job number assigned to the job when it was submitted.
- *stepid* is the job step number assigned to the job step when it was submitted.

The job or step identifier may be specified in an abbreviated form, *jobid* or *jobid.stepid*, when the command is invoked on the same machine that assigned the job and step identifiers. In this case, LoadLeveler will use the local machine's hostname to construct the full job or step identifier.

If the **-u** or **-h** option is specified, the *joblist* is ignored.

Note: For coscheduled jobs, even if all coscheduled job steps are not in the list of targeted job steps, the requested operation is performed on all coscheduled job steps.

Description

This is a LoadLeveler administrator command used for BACKFILL and external schedulers only. Regular users do not have the authority to run this command. This command can only be used when the preemption function is enabled (BACKFILL or external schedulers with preemption enabled).

LoadLeveler for Linux platforms support all preempt methods except suspend (**su**).

Job steps with a **job_type** of **bluegene** cannot be made preemptable.

Examples

1. This example requests that job step c163n07.12.0 be preempted by the default preempt method:
llpreempt c163n07.12.0
2. This example requests that job step c163n07.12.0 be resumed:
llpreempt -r c163n07.12.0

llpreempt

3. This example requests that all job steps owned by user frank and monitored by host c52n01 be preempted by the system hold method:

```
llpreempt -m sh -u frank -h c52n01
```

Results

The following shows a sample system response for the **llpreempt** command:

```
llpreempt: request has been sent to LoadLeveler.
```

Security

LoadLeveler administrators can issue this command.

llprio - Change the user priority of submitted job steps

Purpose

llprio – Changes the user priority of one or more job steps in the LoadLeveler queues. You can adjust the priority by supplying a + (plus) or – (minus) immediately followed by an *integer* value. **llprio** does not affect a job step that is running, even if its priority is lower than other jobs steps, unless the job step goes into the Idle state.

Syntax

llprio [-?] [-H] [-v] [-q] [-X *cluster_name*] [+*integer* | -*integer* | -p *priority*] *joblist*

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- q Specifies quiet mode: print no messages other than error messages.
- X *cluster_name*
Specifies the name of a single cluster where the command is to run.
- + | - *integer*
Operates on the current priority of the job step, making it higher (closer to execution) or lower (further from execution) by adding or subtracting the value of *integer*.
- p *priority*
Is the new absolute value for priority. The valid range is 0–100 (inclusive) where 0 is the lowest possible priority and 100 is highest.

joblist

Is a blank-delimited list of jobs. When a job identifier is specified, the command action is taken for all steps of the job. At least one job or step identifier must be specified.

The format of a job identifier is *host.jobid*. The format of a step identifier is *host.jobid.stepid*.

where:

- *host* is the name of the machine that assigned the job and step identifiers.
- *jobid* is the job number assigned to the job when it was submitted.
- *stepid* is the job step number assigned to the job step when it was submitted.

The job or step identifier may be specified in an abbreviated form, *jobid* or *jobid.stepid*, when the command is invoked on the same machine that assigned the job and step identifiers. In this case, LoadLeveler will use the local machine's hostname to construct the full job or step identifier.

Description

The user priority of a job step ranges from 0 to 100 inclusively, with higher numbers corresponding to greater priority. The default priority is 50. Only the owner of a job step or the LoadLeveler administrator can change the priority of that job step. Note that the priority is not the UNIX *nice* priority.

llprio

Priority changes resulting in a value less than 0 become 0.

Priority changes resulting in a value greater than 100 become 100.

Any change to a job step's priority applied by a user is relative only to *that user's other job steps* in the same class. If you have three job steps enqueued, you can reorder those three job steps with **llprio** but the result does not affect job steps submitted by other users, regardless of their priority and position in the queue.

For more information, see "Setting and changing the priority of a job" on page 212.

Examples

1. This example raises the priority of job 4, job step 1 submitted to machine bronze by a value of 25:

```
llprio +25 bronze.4.1
```
2. This example sets the priority of job 18, job step 4 submitted to machine silver to 100, the highest possible value:

```
llprio -p 100 silver.18.4
```

You should receive a response similar to the following:
llprio: Priority command has been sent to the central manager.

Security

LoadLeveler administrators and users can issue this command.

llq - Query job status

Purpose

llq – Queries information about jobs in the LoadLeveler queues.

Syntax

```
llq [-?] [-H] [-v] [-W] [-x [-d]] [-s] [-l] [-b] [-w] [-X {cluster_list | all} ]
    [-j joblist | joblist] [-u userlist] [-h hostlist] [-c classlist]
    [-R reservation_list] [-f category_list] [-r category_list]
```

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- W Specifies that the width of columns in tabular output will be increased to fit the widest entry.
- x Provides extended information about the selected job. If the -x flag is used with the -r, -s, or -f flag, an error message is generated.

CPU usage and other resource consumption information on active jobs can only be reported using the -x flag if the LoadLeveler administrator has enabled it by specifying A_ON and A_DETAIL for the ACCT keyword in the LoadLeveler configuration file.

Normally, llq connects with the central manager to obtain job information. When you specify -x, llq connects to the Schedd machine that received the specified job to get extended job information. However, some statistics, including those corresponding to System Priority and q_sysprio, are available only from the central manager. Do not use the -x option if you need these statistics.

When specified without -l, CPU usage for active jobs is reported in the short format.

Note: Using both the -l and -x options without a *joblist* specification can produce a very long report and excessive network traffic.

- d Displays the user-specified unfiltered job command file keyword statements. Information is available only on jobs submitted in a multicluster environment. You must specify the -d flag in combination with the -x flag.
- s Provides information on why a selected list of jobs remain in the NotQueued, Idle, or Deferred state. Along with this flag, users must specify a list of jobs. The user can also optionally supply a list of machines to be considered when determining why the jobs cannot run. If a list of machines is not provided, the default is the list of machines in the LoadLeveler cluster. For each job, llq determines why the job remains in one of the given states instead of Running.
- l Specifies that a long listing be generated for each job for which status is requested.

- b** Shows Blue Gene jobs in short form. This is the Blue Gene equivalent of the **llq** standard listing. Using this flag will display the following fields:
- BG** The state of the job on the Blue Gene system.
- Id** The LoadLeveler job step ID.
- LL** The LoadLeveler state of the job step.
- Owner**
The user ID of the job's owner.
- Partition**
The name of the Blue Gene partition assigned to the job.
- PT** The state of the Blue Gene partition assigned to the job.
- Size** The number of Blue Gene compute nodes allocated for the job.
- Submitted**
The time the job step was submitted to LoadLeveler.
- w** Provides AIX Workload Manager (WLM) CPU and real memory statistics for jobs in the running state. This flag can be used with a joblist, steplist, or a single stepid. All other flags except **-h** will result in an error message.
- When the **-w** flag is augmented with a single stepid, the **-h** flag can be used in conjunction with **-w** to specify a single hostname.
- This flag can only be used when `ENFORCE_RESOURCE_USAGE` is enabled in the configuration file. Otherwise, an error message is produced.
- The following statistics are displayed for every node the job is running on:
- Current CPU resource consumption as a percentage of the total resources available
 - Total CPU time consumed in milliseconds
 - Current real memory consumption as a percentage of the total resources available
 - The highest number of resident memory pages used
- X {*cluster_list* | all}**
Indicates that you can specify the **-X** flag with either:
- cluster_list* Is a blank-delimited list of clusters where the command is to run.
- all** Is the reserved word indicating that the command is to run in all accessible clusters.
- j *joblist* | *joblist***
Is a blank-delimited list of job and step identifiers. When a job identifier is specified, the command action is taken for all steps of the job. At least one job or step identifier must be specified.
- The format of a job identifier is *host.jobid*. The format of a step identifier is *host.jobid.stepid*.
- where:
- *host* is the name of the machine that assigned the job and step identifiers.
 - *jobid* is the job number assigned to the job when it was submitted.
 - *stepid* is the job step number assigned to the job step when it was submitted.

The job or step identifier may be specified in an abbreviated form, *jobid* or *jobid.stepid*, when the command is invoked on the same machine that assigned the job and step identifiers. In this case, LoadLeveler will use the local machine's hostname to construct the full job or step identifier.

The **-j** *joblist* flag is used to distinguish a *joblist* when specified in combination with the any flag that supports a list.

If the **-X** flag is specified in combination with a *joblist*, the **-j** flag must be specified. For example:

```
llq -X my_cluster1 my_cluster2 -j c94n13.2.1 c94n13.25.0
```

-u *userlist*

Is a blank-delimited list of users. Only job steps belonging to users in this list are queried.

-h *hostlist*

Is a blank-delimited list of machines. If the **-s** flag is not specified, only job steps managed by the Schedd on machines in this list are queried. If the **-s** flag is specified, the list of machines is considered when determining why a job remains in the Idle state.

When the **-h** flag is used with the **-w** flag, only a single machine name can be specified to obtain the WLM statistics for that machine.

-c *classlist*

Is a blank-delimited list of classes. Only job steps belonging to classes in this list are queried.

-f *category_list*

Is a blank-delimited list of categories you want to query. Each category you specify must be preceded by a percent sign. The *category_list* cannot contain duplicate entries. This flag allows you to create a customized version of the standard **llq** listing. You cannot use this flag with the **-l** flag. The output fields produced by this flag all have a fixed length. The output is displayed in the order in which you specify the categories. *category_list* can be one or more of the following:

```
%a    Account number
%c    Class
%cc   Completion code
%dc   Completion date
%dd   Dispatch Date
%dh   Hold date
%dq   Queue date ("Submitted" date of "standard" llq output)
%fj   Favored Job
%gl   LoadLeveler group
%gu   UNIX group
%h    Hostname (first hostname if more than one machine is allocated to
      the job step)
%id   Step ID
%is   Virtual image size
%jn   Job name
%jt   Job type
%nh   Number of hosts allocated to the job step
%o    Job owner
%p    User priority
%R    Reservation ID
%sn   Step name
%st   Status
```

%X Cluster name where the job is to be scheduled
%Xf Cluster name from where the job was sent
%Xk Cluster name the user requested
%Xs Cluster name from where the job was submitted
%Xu User name of the original submission

-r *category_list*

Is a blank-delimited list of formats (categories) you want to query. Each category you specify must be preceded by a percent sign. The *category_list* cannot contain duplicate entries. This flag allows you to create a customized version of the standard **llq** listing. You cannot use this flag with the **-l** flag. The output produced by this flag is considered raw, in that the fields can be variable in length. Output fields are separated by an exclamation point (!). The output is displayed in the order in which you specify the formats. *category_list* can be one or more of the formats listed under the **-f** flag.

-R *reservation_list*

Is a blank-delimited list of reservation identifiers. Only job steps bound to reservations in this list are queried. The format of a full LoadLeveler reservation identifier is [*host.*]rid[.r].

where:

- *host* is the name of the machine that assigned the reservation identifier.
- *rid* is the number assigned to the reservation when it was created. An *rid* is required.
- **r** indicates that this is a reservation ID (**r** is optional).

The reservation identifier may be specified in an abbreviated form, *rid[.r]*, when the command is invoked on the same machine that assigned the reservation identifier. In this case, LoadLeveler will use the local machine's host name to construct the full reservation identifier.

If a job step is not specified and if **-u**, **-h**, **-c**, or **-R** is not specified, all jobs are queried.

If a job step is specified, you cannot specify **-u**, **-h**, **-c**, or **-R**, except in the cases of **-w** and **-s**, for which the **-h** flag has special meaning.

When **-u**, **-h**, **-c**, or **-R** are used in combination, the result is the intersection of the job steps selected by each flag.

The **-b** flag can be used alone or with the **-u** flag and the *joblist* argument. If used in conjunction with any other flag, an error will occur.

You cannot specify **-d**, **-x**, or **-w** in combination with the **-X** flag.

Description

The **llq** command queries information about jobs in the LoadLeveler queues.

Examples

1. This example generates the standard listing where the machine mars has two jobs running and one job waiting:

Id	Owner	Submitted	ST	PRI	Class	Running On
mars.498.0	brownap	5/20 11:31	R	100	silver	mars
mars.499.0	brownap	5/20 11:31	R	50	No_Class	mars
mars.501.0	brownap	5/20 11:31	I	50	silver	

3 job step(s) in query, 1 waiting, 0 pending, 2 running, 0 held,
0 preempted

The **standard listing** includes the following fields:

Class Job class.

Id The format of a full LoadLeveler step identifier is *host.jobid.stepid*. If the **llq** command returns information about a job owned by a Schedd in the same domain, then the domain of the hostname will not appear in the output. However, when the **llq** command reports information about a job owned by a Schedd in a different domain, the fully qualified hostname is always included. Due to space limitations, the domain of the host may be truncated to fit in the space allocated to the Id field. If the domain is truncated, a dash (-) will appear at the end to indicate that characters have been left out. To see the full job ID, run **llq** with the **-l** flag.

Owner

User ID that the job will be run under.

PRI

User priority of the job step, where the values are defined with the **user_priority** keyword in the job command file or changed by the **llprio** command, which is described in “llprio - Change the user priority of submitted job steps” on page 447

Running On

If running, the name of the machine the job step is running on. This is blank when the job is not running. For a parallel job step, only the first machine is shown.

ST

For more information, see “LoadLeveler job states” on page 18.

Submitted

Date and time of job submission.

- This example generates the long listing. The long listing is generated when you specify the **-x -l** flags with the **llq** command:

```
llq -l -x c271f2rp01.ppd.pok.ibm.com.16.0
```

The **long listing** includes the following fields. See Appendix B, “Sample command output,” on page 673 for sample output of long listings.

Account

The account number specified in the job command file.

Adapter Requirement

Reflects the settings of the **network** keyword in the job command file.

For more information on the **network** keyword, see “Job command file keyword descriptions” on page 336.

Allocated Hosts

The machines that have been allocated for this job step.

Args

Arguments that were passed to the executable.

Blocking

Reflects the settings for the **blocking** keyword in the job command file.

Blue Gene Job ID

The ID of the Blue Gene job in the Blue Gene DB2[®] database. This field is displayed for Blue Gene jobs only.

Blue Gene Status

The state of the Blue Gene job in the Blue Gene DB2 database. This field is displayed for Blue Gene jobs only.

BG Requirements

The job step requirements on Blue Gene resources as specified at job submission. This field is displayed for Blue Gene jobs only.

Bulk Transfer

Indicates that the value will be Yes or No depending on whether the application requested that the communication subsystem use bulk transfer by specifying `bulkxfer=yes` in the job command file.

Checkpoint Directory

Value of the `ckpt_dir` keyword.

Checkpoint File

For AIX checkpointable job steps, the file name to be used for checkpoint data.

Checkpointable

Indicates if LoadLeveler considers the job step checkpointable (yes, no, or interval).

Ckpt Accum Time

Accumulated time, in seconds, the job step has spent checkpointing.

Ckpt Elapse Time

Amount of time taken to perform the last successful checkpoint.

Ckpt Execute Dir

The directory where the job step's executable will be saved for checkpointable jobs.

Ckpt Hard Limit

Checkpoint hard limit as specified at job step submission.

Ckpt Soft Limit

Checkpoint soft limit as specified at job step submission.

Ckpt Start Time

The start time of the current checkpoint in progress. Blank if no checkpoint running.

Class The class of the job step as specified at job submission.

class_sysprio

The class priority of the job step, where the value is defined in the administration file.

Cluster input file

The information format is *local_pathname, remote_pathname*.

where:

local_pathname

Is the full pathname of the file to be copied from the local cluster.

remote_pathname

Is the full pathname of the file that will be copied into the remote cluster.

Cluster output file

The information format is *local_pathname, remote_pathname*.

where:

local_pathname

Is the full pathname of the file that will be copied into the local cluster.

remote_pathname

Is the full pathname of the file to be copied from the remote cluster.

Cmd The name of the executable associated with the **executable** keyword (if specified) or the name of the job command file.

Comment

The comment specified by the **comment** keyword in the job command file.

Completion Code

The status returned by the wait3 UNIX system call.

Completion Date

Date and time job completed or exited.

Core Hard Limit

Core hard limit as specified at job submission.

Core Soft Limit

Core soft limit as specified at job submission.

Coschedule

Indicates whether the job step is required to be coscheduled (**yes** or **no**).

Cpu Hard Limit

CPU hard limit as specified at job submission.

Cpu Soft Limit

CPU soft limit as specified at job submission.

Data Hard Limit

Data hard limit as specified at job submission.

Data Soft Limit

Data soft limit as specified at job submission.

Dependency

Job step dependencies as specified at job submission.

Dispatch Time

The time that the job was dispatched.

Env Environment variables to be set before executable runs. Appears only when the **-x** option is specified.

Err The file to be used for stderr.

Error Text

The error text in the Blue Gene job record from the Blue Gene DB2 database. This field is displayed for Blue Gene jobs only.

Fail Ckpt Time/Date

Time and date stamp of the last failed checkpoint.

File Hard Limit

File hard limits as specified at job submission.

File Soft Limit

File soft limit as specified at job submission.

Favored Job

Indicates whether the job has been specified to have a higher system priority than all jobs that are not favored (**yes** or **no**).

Good Ckpt Time/Date

Time and date stamp of the last successful checkpoint.

group_sysprio

The group priority of the job step, where the value is defined in the administration file.

high water

The highest number of resident memory pages used. Real Memory resource only.

Hold Job Until

Job step is deferred until this date and time.

In The file to be used for stdin.

Initial Working Dir

The directory from which the job step is run. The relative directory from which the stdio files are accessed, if appropriate.

Job Accounting Key

The Job Accounting Key is a unique identifier for a LoadLeveler job step. The accounting key is stored in the AIX accounting record for each process associated with a LoadLeveler job step. This field can be used to correlate AIX accounting records with LoadLeveler accounting records. The Job Accounting Key is stored in the history file and can be displayed using the **llsummary -l** command.

This keyword is not applicable on LoadLeveler for Linux platforms.

For more information on the Job Accounting Key, see “Correlating AIX and LoadLeveler accounting records” on page 61.

Job Name

The name of the job.

Job Step ID

The job step identifier.

Large Page

Indicates whether Large Page memory should be used to run this job step. Can be Y (use Large Page memory if available), N (No), or M (Mandatory).

LoadLeveler Group

The LoadLeveler group associated with the job step.

Machine Speed

For a serial job step, the value associated with the **speed** keyword of the machine that is running this job step. For a parallel job step, the value associated with the **speed** keyword of the first machine that has been allocated for this job step.

Max Processors

The maximum number of processors that can be used for this job step.

McmAffinityOptions

The MCM affinity options for the job.

Min Processors

The minimum number of processors needed for this job step.

Negotiator Messages

Informational messages for the job step if it is in the Idle or NotQueued state.

(Node) Allocated Hosts

- The machines of this Node type that have been allocated for this job step. The format is:

```
hostname:task status:adapter usage, ... ,adapter usage, \
    cpu usage, ... ,cpu usage + ... +,
hostname:task status:adapter usage, ... ,adapter usage, \
    cpu usage, ... ,cpu usage
```

- The adapter usage information has the format:

```
adapter name(adapter window ID, network protocol, mode, \
    adapter window memory)
```

For information on the units used to report window memory, see the description of the “Adapter” field in “llstatus - Query machine status” on page 477.

- The CPU usage information has one of the following formats:

```
CPU <cpulist>
MCMnumber:CPU <cpulist>
```

The *cpulist* is a blank-delimited list of individual CPU IDs or CPU ranges, or a combination of both CPU IDs and CPU ranges. The CPU range is specified as the starting CPU ID and the ending CPU ID separated by a hyphen (-).

(Node) Name

Blank value. Reserved for future use.

(Node) Node actual

Actual number of machines of this Node type that are used in the running of this job step.

(Node) Node maximum

Maximum number of machines of this Node type that can be used to run this job step.

(Node) Node minimum

Minimum number of machines of this Node type required to run this job step.

(Node) Preferences

Job step preferences as specified at job submission.

(Node) Requirements

Job step requirements as specified at job submission.

(Node/Master Task) Exec Args

The arguments passed to the master task executable.

(Node/Master Task) Executable

The executable associated with the master task.

(Node/Master Task) Num Task Inst

The number of task instances of the master task.

(Node/Master Task) Task Instance

- Task instance information has the format:
hostname:task ID:adapter usage, ... ,adapter usage
- Adapter usage information has the format:
adapter name(adapter window ID, network protocol, mode, \
adapter window memory)
For information on the units used to report window memory, see the description of the “Adapter” field in “llstatus - Query machine status” on page 477.

(Node/Task) Num Task Inst

The number of task instances.

(Node/Task) Task Instance

- Task instance information has the format:
hostname:task ID:adapter usage, ... ,adapter usage, cpu usage
- Adapter usage information has the format:
adapter name(adapter window ID, network protocol, mode, \
adapter window memory)
For information on the units used to report window memory, see the description of the “Adapter” field in “llstatus - Query machine status” on page 477.
- The CPU usage information has one of the following formats:

```
CPU <cpulist>
MCMnumber:CPU <cpulist>
```

The *cpulist* is a blank-delimited list of individual CPU IDs or CPU ranges, or a combination of both CPU IDs and CPU ranges. The CPU range is specified as the starting CPU ID and the ending CPU ID separated by a hyphen (-).

Node Usage

A request that a node be shared or not shared or that a time-slice is not shared. The user specifies this request while submitting the job.

Notifications

The notification status for the job step, where:

always

Indicates notification is sent through the mail for the complete, error, never, and start notification categories.

complete

Indicates notification is sent through the mail only when the job step completes.

error

Indicates notification is sent through the mail only when the job step terminates abnormally.

never

Indicates notification is never sent.

start

Indicates notification is sent through the mail only when starting or restarting the job step.

Notify User

The user to be notified by mail of a job's status.

Out The file to be used for stdout.

Outbound Schedds

The list of Schedds that have acted as the outbound Schedd for the job. The last Schedd in the list is the current outbound Schedd. This field only displays multicluster-specific information that was submitted or moved to a remote cluster.

Owner

The user ID that the job will be run under.

Partition ID

The ID of the Blue Gene partition allocated for the job. This field is displayed for Blue Gene jobs only.

Partition State

The state of the Blue Gene partition allocated for the job. This field is displayed for Blue Gene jobs only.

Port Number

The port number for InfiniBand resources used by the running job.

Preempt Wait Count

Specifies the number of job steps that an idle job step must preempt before it can be started.

Preemptable

Indicates whether a job step is preemptable (yes or no).

Preferences

Job step preferences as specified at job submission.

previous q_sysprio

The previous adjusted system priority of the job step. For more information, see "Example: How does a job's priority affect dispatching order?" on page 213.

q_sysprio

The adjusted system priority of the job step. For more information, see "Example: How does a job's priority affect dispatching order?" on page 213.

Queue Date

The date and time that LoadLeveler received the job.

Requested Cluster

The cluster the user specified at job submission.

Requested Res. ID

The reservation identifier that a job step is requested to be bound to, but has not yet been bound to. This field will be set when a job submitted with a request to bind has been successfully submitted, but

the bind has not yet occurred. The bind may never occur if either the owner of the job step is not allowed to use the reservation, or if the reservation does not exist.

If a job command file is used to select nodes to reserve in a make or change reservation request and the request fails, all steps of the job, if submitted successfully, will have MAKERES as their Requested Res. ID and the steps will be in the **NQ** state.

Requirements

Job step requirements as specified at job submission.

Reservation ID

The reservation identifier that a job step is bound to. If a job step is not bound to any reservation, this field will be blank.

Resource

The resource being enforced by WLM. This is either **CPU** or **Real Memory**.

Resources

Reflects the settings for the **resources** keyword in the job command file.

Restart

Restart status (**yes** or **no**).

Restart From Ckpt

Indicates if a job has been restarted from an existing checkpoint (**yes** or **no**).

Restart Same Nodes

Indicates if a job step should be restarted on the same nodes after vacate (**yes** or **no**).

Rotate Indicates whether the scheduler is free to rotate the requested Blue Gene partition shape in order to match an available partition (**TRUE** or **FALSE**). This field is displayed for Blue Gene jobs only.

RSet The RSet requirement of the job.

Rss Hard Limit

RSS hard limit as specified at job step submission.

Rss Soft Limit

RSS soft limit as specified at job step submission.

Running Host

For a serial job step, the machine that is running this job step. For a parallel job step, the first machine that has been allocated for this job step.

Schedd History

The list of Schedds that have acted as the Schedd host for the job. The last one in the list is the current Schedd host. This field only displays multicluster-specific information.

Scheduling Cluster

The cluster name where the job is to be scheduled. This field only displays multicluster-specific information.

Sending Cluster

The cluster name that the job was sent from when moved. This field only displays multicluster-specific information.

Shape Allocated

The allocated shape of the Blue Gene partition for the job. This field is defined only for running-like jobs. This field is displayed for Blue Gene jobs only.

Shape Requested

The requested shape of the Blue Gene partition for the job, if defined, in units of base partitions. This field is displayed for Blue Gene jobs only.

Shell The shell to be used when the job step runs.

Size Allocated

The size of the Blue Gene partition for the job in units of compute nodes. The size allocated is not always identical to the requested size. This field is displayed for Blue Gene jobs only.

Size Requested

The requested size of the Blue Gene partition for the job in units of compute nodes. The size must be equivalent to the size of the shape, if such is defined. This field is displayed for Blue Gene jobs only.

SMT requested

Indicates the required Simultaneous Multi-Threading (SMT) state, which is defined in the job command file, if the job step requires SMT to be turned on or off. Valid values are **yes** or **no**.

snapshot

Current CPU or Real Memory consumption as a percentage of the total resources available.

Stack Hard Limit

Stack hard limit as specified at job submission.

Stack Soft Limit

Stack soft limit as specified at job submission.

Starter idrss/Step Starter idrss

An integral value of the amount of unshared memory in the data segment of a process (expressed in units of kilobytes * seconds-of-execution).

Starter inblock/Step inblock

Number of times file system performed input. Cumulative total.

Starter isrss/Step isrss

Depending on the Operating System, this field may contain the integral value of unshared stack size.

Starter ixrss/Step ixrss

An integral value indicating the amount of memory used by the text segment that was also shared among other processes (expressed in units of kilobytes * seconds-of-execution).

Starter majflt/Step majflt

Number of page faults (I/O required). Cumulative total.

Starter maxrss/Step maxrss

Maximum resident set size utilized. Maximum value.

Starter minflt/Step minflt

Number of page faults (reclaimed). Cumulative total.

- Starter msgrcv/Step msgrcv**
Number of IPC messages received. Cumulative total.
- Starter msgsnd/Step msgsnd**
Number of IPC messages sent. Cumulative total.
- Starter nivcsw/Step nivcsw**
Number of involuntary context switches. Cumulative total.
- Starter nsignals/Step nsignals**
Number of signals delivered. Cumulative total.
- Starter nswap/Step nswap**
Number of times swapped out. Cumulative total.
- Starter nvcswh/Step nvcswh**
Number of context switches due to voluntarily giving up processor. Cumulative total.
- Starter oublock/Step oublock**
Number of times file system performed output. Cumulative total.
- Starter System Time/Step System Time**
CPU system time of Starter/Step processes. Cumulative total.
- Starter Total Time/Step Total Time**
CPU total time of Starter/Step processes. Cumulative total.
- Starter User Time/Step User Time**
CPU user time of Starter/Step processes. Cumulative total.
- Status** The status (state) of the job. For more information, see “LoadLeveler job states” on page 18.
- Step Adapter Memory**
The total adapter pinned memory for the job step.
- Step Cpu Hard Limit**
Job step CPU hard limit as specified at job submission.
- Step Cpu Soft Limit**
Job step CPU soft limit as specified at job submission.
- Step Cpus**
The total ConsumableCpus for the job step.
- Step Name**
The name of the job step.
- Step rCxt Blocks**
The number of rCxt blocks for High Performance Switch adapters.
- Step Real Memory**
The total ConsumableMemory for the job step.
- Step Type**
Type of job step:
- Serial
 - General parallel
 - Blue Gene
 - MPICH parallel
- Step Virtual Memory**
The total ConsumableVirtualMemory for the job step.

Structure Version

An internal version identifier.

Submitting Cluster

The cluster name where the job was submitted from. This field only displays multicluster-specific information.

Submitting Host

The name of the machine to which the job is submitted.

Submitting User

The user name that the job was submitted under. This field only displays multicluster-specific information.

System Priority

The overall system priority of the job step, where the value is defined by the SYSPRIO expression in the configuration file.

Task_geometry

Reflects the settings for the **task_geometry** keyword in the job command file.

total Total CPU time consumed in milliseconds. CPU resource only.

Unix Group

The effective UNIX group name.

User Priority

The priority of the job step, as specified by the user in the job command, or changed by the **llprio** command.

User Space Windows

The number of switch adapter windows assigned to the job step.

user_sysprio

The user system priority of the job step, where the value is defined in the administration file.

Virtual Image Size

The value of the **image_size** keyword (if specified) or the size of the executable associated with the **executable** keyword (if specified) or the size of the job command file.

Wall Clk Hard Limit

Wall clock hard limit as specified at job submission.

Wall Clk Soft Limit

Wall clock soft limit as specified at job submission.

Wiring Allocated

Allocated type of wiring for the Blue Gene partition. It is either TORUS or MESH. This field is displayed for Blue Gene jobs only.

Wiring Requested

Requested type of wiring for the Blue Gene partition. It is TORUS, MESH, or PREFER_TORUS. This field is displayed for Blue Gene jobs only.

- Using the abbreviated form of *jobid*, this example generates a standard listing for all job steps with *jobid* 12 assigned by the local machine:

```
llq 12
```

- This example generates a standard listing for all job steps owned by either rich or nathan and bound to reservation 6:

```
llq -u rich nathan -R 6
```

5. This example generates an extended listing for all job steps of class batch or class highprio, managed by the Schedd daemon on either c94n07 or c94n09:
llq -x -c batch highprio -h c94n07 c94n09
6. The following example generates a standard listing for all job steps bound to reservation c94n04.2.r:

```
llq -R c94n04.2.r
```

You should receive a response similar to the following:

```

Id              Owner      Submitted  ST PRI Class      Running On
-----
c94n04.5.0      zhong      2/8  08:17 I  50  classA

```

1 job step(s) in query, 1 waiting, 0 pending, 0 running, 0 held,
0 preempted

7. The following example generates a customized listing for all job steps:

```
llq -f %id %o %R
```

You should receive a response similar to the following:

```

Step Id          Owner      Reservation ID
-----
c94n04.5.0      zhong      c94n04.2.r
c94n04.4.0      zhong

```

2 job step(s) in queue, 1 waiting, 0 pending, 0 running, 1 held,
0 preempted

8. The following is sample output for **llq -X cluster2**. The output representing a cluster is delineated with a cluster header, in this example it is cluster2.

```

===== Cluster cluster2 =====

```

```

Id              Owner      Submitted  ST PRI Class      Running On
-----
c188f2n02.9.0  brownap    10/29 13:54 R  50  april      c188f2n08

```

1 job step(s) in query, 0 waiting, 0 pending, 1 running, 0 held, 0 preempted

9. The following is sample output for **llq -X all** command where there are two clusters:

```

===== Cluster cluster1 =====

```

```

Id              Owner      Submitted  ST PRI Class      Running On
-----
c193n13.283.0  rolf      8/4  10:58 R  50  No_Class  c193n13
c193n13.283.0  rolf      8/4  10:58 R  50  No_Class  c193n13
c193n13.289.0  rolf      8/4  10:58 R  50  No_Class  c193n13
c193n13.291.0  rolf      8/4  10:59 R  50  No_Class  c193n13
c193n13.293.0  rolf      8/4  10:59 R  50  No_Class  c193n13
c193n13.295.0  rolf      8/4  10:59 R  50  No_Class  c193n13
c193n13.297.0  rolf      8/4  11:01 R  50  No_Class  c193n13
c193n13.299.0  brownap    8/4  11:02 R  50  No_Class  c193n13

```

7 job step(s) in queue, 0 waiting, 0 pending, 7 running, 0 held, 0 preempted

```

===== Cluster cluster2 =====
Id              Owner      Submitted  ST PRI Class      Running On
-----
c193n13.265.0  11b1d     8/2 13:42  R  50  No_Class  c197blade3b14
c193n13.267.0  11b1d     8/2 13:43  R  50  No_Class  c197blade3b14
c193n13.271.0  11b1d     8/2 13:55  I  50  No_Class
c193n13.273.0  11b1d     8/4 10:53  I  50  No_Class
c193n13.275.0  11b1d     8/4 10:53  I  50  No_Class
c193n13.277.0  11b1d     8/4 10:53  I  50  No_Class
c193n13.279.0  11b1d     8/4 10:58  I  50  No_Class

```

7 job step(s) in queue, 5 waiting, 0 pending, 2 running, 0 held, 0 preempted

10. The following is sample output for **llq -b** for a standard Blue Gene listing:

```

Id              Owner      Submitted  LL BG PT Partition  Size
-----
fen01.193.0    jdoe       2/21 22:52  R      C  part031  65536
fen02.1305.0   jfoe       2/20 02:36  R      R  B  part031   1024
fen01.194.0    jane       2/21 14:12  I

```

11. This example generates a customized and formatted standard listing:

```
llq -f %id %c %dq %dd %gl %h
```

You should receive output similar to the following:

```

Step Id      Class      Queue Date  Disp. Date  LL Group  Running On
-----
116.2.0     No_Class  04/08 09:19  04/08 09:21  No_Group  116.pok.ibm.com
116.1.0     No_Class  04/08 09:19  04/08 09:21  No_Group  116.pok.ibm.com
116.3.0     No_Class  04/08 09:19  04/08 09:21  No_Group  115.pok.ibm.com

```

3 job step(s) in queue, 0 waiting, 0 pending, 3 running, 0 held, 0 preempted

12. This example generates a customized, unformatted (raw) standard listing. Output fields are separated by an exclamation point (!).

```
llq -r %id %c %dq %dd %gl %h
```

You should receive output similar to the following:

```

116.pok.ibm.com.2.0!No_Class!10/16/2006 09:19!10/16/2006 09:21! \
    No_Group!116.pok.ibm.com
116.pok.ibm.com.1.0!No_Class!10/16/2006 09:19!10/16/2006 09:21! \
    No_Group!116.pok.ibm.com
116.pok.ibm.com.3.0!No_Class!10/16/2006 09:19!10/16/2006 09:21! \
    No_Group!115.pok.ibm.com

```

13. This example generates a WLM CPU and real memory statistics listing where c209f1n05.13.0 is a CPU intensive parallel job step currently running on the 2 nodes c209f1n05 and c209f1n01: If the LoadLeveler interface to AIX Workload Manager (WLM) is enabled, the **-w** option can be used to obtain CPU and real memory statistics of job steps in running state. Note that Large Page memory information is not included in the statistics since WLM does not manage Large Page memory.

```
llq -w c209f1n05.13.0
```

You should receive output similar to the following:

```

===== Job Step c209f1n05.ppd.pok.ibm.com.13.0 =====
c209f1n05.ppd.pok.ibm.com:
  Resource: CPU
    snapshot: 99
    total: 80172
  Resource: Real Memory
    snapshot: 1
    high water: 2561

c209f1n01.ppd.pok.ibm.com:
  Resource: CPU
    snapshot: 100
    total: 79303
  Resource: Real Memory
    snapshot: 1
    high water: 1919

```

14. The following is sample llq -l output for task instances and allocated hosts if the job requested MCM affinity:

```

Allocated Hosts: e189f4rp04.ppd.pok.ibm.com:: \
    sn1(MPI,IP,-1,Shared,0 rCxt Blks), \
    sn1(MPI,IP,-1,Shared,0 rCxt Blks), \
    MCM0:CPU< 0-5 >,MCM0:CPU< 0-5 > \
+ e189f4rp03.ppd.pok.ibm.com:: \
    sn1(MPI,IP,-1,Shared,0 rCxt Blks), \
    sn1(MPI,IP,-1,Shared,0 rCxt Blks), \
    MCM1:CPU< 4-5 >,MCM1:CPU< 4-5 >

Num Task Inst: 4
  Task Instance: e189f4rp04:0:sn1(MPI,IP,-1,Shared,0 rCxt Blks), \
    MCM0:CPU< 0-5 >
  Task Instance: e189f4rp04:1:sn1(MPI,IP,-1,Shared,0 rCxt Blks), \
    MCM0:CPU< 0-5 >
  Task Instance: e189f4rp03:2:sn1(MPI,IP,-1,Shared,0 rCxt Blks), \
    MCM1:CPU< 4-5 >
  Task Instance: e189f4rp03:3:sn1(MPI,IP,-1,Shared,0 rCxt Blks), \
    MCM1:CPU< 4-5 >

```

Note that the < > notation will be used to list individual CPU IDs instead of the CPU count () notation when the **RSET_SUPPORT** configuration file keyword is set to **RSET_CONSUMABLE_CPUS** or **RSET_MCM_AFFINITY**.

15. The following example shows the Resource Set information in the llq -l listing when the consumable CPUs Resource Set requirement is requested:

```

Allocated Hosts : e189f4rp01.ppd.pok.ibm.com:: \
    sn0(MPI,IP,-1,Shared,0 rCxt Blks), \
    sn0(MPI,IP,-1,Shared,0 rCxt Blks), \
    CPU< 0-5 >,CPU< 0-5>
+ e189f4rp02.ppd.pok.ibm.com:: \
    sn0(MPI,IP,-1,Shared,0 rCxt Blks), \
    sn0(MPI,IP,-1,Shared,0 rCxt Blks), \
    CPU< 0-5 >,CPU< 0-5 >

Num Task Inst: 4
  Task Instance: e189f4rp01:0:sn0(MPI,IP,-1,Shared,0 rCxt Blks), \
    CPU< 0-5 >
  Task Instance: e189f4rp01:1:sn0(MPI,IP,-1,Shared,0 rCxt Blks), \
    CPU< 0-5 >

```

```

Task Instance: e189f4rp02:2:sn0(MPI,IP,-1,Shared,0 rCxt Blks), \
CPU< 0-5 >
Task Instance: e189f4rp02:3:sn0(MPI,IP,-1,Shared,0 rCxt Blks), \
CPU< 0-5 >

```

16. The following example shows output for the **llq -l** command when rCxt blocks are present:

```

.
.
.
Adapter Requirement: (sn_single,MPI,US,shared,AVERAGE,instances=1, \
rcxtblks=5)
.
.

```

```

Num Task Inst: 4
Task Instance: c60f1rp02:0:sn1(MPI,US,10,Shared,5 rCxt Blks),
Task Instance: c60f1rp02:1:sn0(MPI,US,10,Shared,5 rCxt Blks),
Task Instance: c60f1rp02:2:sn1(MPI,US,11,Shared,5 rCxt Blks),
Task Instance: c60f1rp02:3:sn0(MPI,US,11,Shared,5 rCxt Blks),

```

17. The following example shows output for the **llq -l** command when SMT is requested:

```

===== Job Step blablahome.clusters.com.22.0 =====
Job Step Id: blablahome.clusters.com.22.0
Job Name: blablahome.clusters.com.22

```

```

.
.
.
Status: Running
.
.

```

```

Large Page: N
Coschedule: no
SMT requested: yes
Checkpointable: no
.
.

```

18. The following example shows the port number for the InfiniBand resources used by the running job:

```

.
.
.
Task
----

Num Task Inst: 2
Task Instance: c171f6sq08:0:ib0(MPI,US,2,Shared,0 rCxt Blks,1), \
ib1(MPI,US,66,Shared,0 rCxt Blks,2),
Task Instance: c171f6sq08:1:ib0(MPI,US,3,Shared,0 rCxt Blks,1), \
ib1(MPI,US,67,Shared,0 rCxt Blks,2),
.
.
.

```

Security

LoadLeveler administrators and users can issue this command.

llqres - Query a reservation

Purpose

llqres – Returns information about reservations in LoadLeveler.

Syntax

```
llqres { -? | -H | -v | -W | [-l | -r] [-s] [[-u user_list] [-g group_list]
[-h host_list | -h all] [-B base_partition_list | -B all] [-b begin_time]
[-e end_time] | -R reservation_list }
```

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- W Specifies that the width of columns in tabular output will be increased to fit the widest entry.
- l Specifies that a long listing be generated for each reservation to be queried.
- r Specifies raw mode for output. Each reservation will occupy one line with the output fields separated by an exclamation point (!).
- s Specifies that short host names will be used in the output of machine names.
- u *user_list*
Specifies a blank-delimited list of users. Reservations to be queried are owned by one of these users.
- g *group_list*
Specifies a blank-delimited list of LoadLeveler groups. Reservations to be queried are owned by one of these groups.
- h all
- h *host_list*
Specifies a blank-delimited list of machines or all machines. Reservations to be queried use one or more of these machines.
- B all
- B *base_partition_list*
Specifies all or a blank-delimited list of base partitions in the Blue Gene system. Reservations to be queried use one or more of these base partitions.
- b *begin_time*
Reservations to be queried are active at or after the specified begin time. The -b flag can be used together with the -e flag to query reservations expected to be active between the begin and end times. The *begin_time* must be specified using the format [mm/dd[/[cc]yy]] HH:MM. Hours must be specified using a 24-hour clock.
- e *end_time*
Reservations to be queried are active at or before the specified end time. The -e flag can be used together with the -b flag to query reservations

expected to be active between the begin and end times. The *end_time* must be specified using the format [mm/dd[/[cc]yy]] HH:MM. Hours must be specified using a 24-hour clock.

-R *reservation_list*

Is a blank-delimited list of reservation identifiers to be queried. The format of a full LoadLeveler reservation identifier is [*host.*]*rid*[.*r*].

where:

- *host* is the name of the machine that assigned the reservation identifier.
- *rid* is the number assigned to the reservation when it was created. An *rid* is required.
- *r* indicates that this is a reservation ID (*r* is optional).

The reservation identifier may be specified in an abbreviated form, *rid*[.*r*], when the command is invoked on the same machine that assigned the reservation identifier. In this case, LoadLeveler will use the local machine's host name to construct the full reservation identifier.

When **-R** is specified, **-u**, **-g**, and **-h** are ignored.

Description

All users can issue this command. Reservations satisfying all criteria specified by **-u**, **-g**, **-h** or **-B**, **-b**, and **-e** will be queried if more than one of these options are present. The **-h** and **-B** flags are mutually exclusive. No reservation information will be returned if both are specified at the same time. By default, the **llqres** command queries all existing reservations.

This command is for the BACKFILL scheduler only.

Examples

When the **BG_ENABLED** keyword is set to **false** in the configuration file, the following examples apply.

1. When issuing the **llqres** command, you should receive output similar to the following:

Id	Owner	ST	Start Time	Duration	#Nodes
-----	----	--	-----	-----	-----
c94n16.30.r	loadl	A	10/16 14:00	120	3
c94n16.31.r	dave	W	10/17 02:00	420	4
c94n16.35.r	carol	W	10/17 13:30	240	2

When issuing the **llqres -l** command, you should receive output similar to the following:

```

===== Reservation c94n16.ppd.pok.ibm.com.30.r =====
              ID: c94n16.ppd.pok.ibm.com.30.r
      Creation Time: Mon Oct 16 08:12:23 EDT 2006
            Owner: loadl
            Group: No_Group
      Start Time: Mon Oct 16 14:00:00 EDT 2006
      Duration: 120 minutes
Expected End Time: Mon Oct 16 16:00:00 EDT 2006
      SHARED: no
REMOVE_ON_IDLE: no
      Status: ACTIVE
      Modified By: loadl
Modification Time: Mon Oct 16 10:21:14 EDT 2006
      Users: 0
      Groups: 1
              loadlusr

```

llqres

```
Nodes: 3
      c94n01.ppd.pok.ibm.com
      c94n11.ppd.pok.ibm.com
      c94n12.ppd.pok.ibm.com
Job Steps: 0
```

```
===== Reservation c94n16.ppd.pok.ibm.com.31.r =====
      ID: c94n16.ppd.pok.ibm.com.31.r
      Creation Time: Mon Oct 16 10:55:18 EDT 2006
      Owner: dave
      Group: No_Group
      Start Time: Tue Oct 17 02:00:00 EDT 2006
      Duration: 420 minutes
      Expected End Time: Tue Oct 17 09:00:00 EDT 2006
      SHARED: no
      REMOVE_ON_IDLE: yes
      Status: WAITING
      Modified By: dave
      Modification Time: Tue Oct 17 10:55:18 EDT 2006
      Users: 0
      Groups: 0
      Nodes: 4
      c94n01.ppd.pok.ibm.com
      c94n02.ppd.pok.ibm.com
      c94n12.ppd.pok.ibm.com
      c94n16.ppd.pok.ibm.com
      Job Steps: 1
      c94n16.ppd.pok.ibm.com.25.0
```

```
===== Reservation c94n16.ppd.pok.ibm.com.35.r =====
      ID: c94n16.ppd.pok.ibm.com.35.r
      Creation Time: Mon Oct 16 10:58:19 EDT 2006
      Owner: carol
      Group: No_Group
      Start Time: Tue Oct 17 13:30:00 EDT 2006
      Duration: 240 minutes
      Expected End Time: Tue Oct 17 17:30:00 EDT 2006
      SHARED: no
      REMOVE_ON_IDLE: no
      Status: WAITING
      Modified By: carol
      Modification Time: Mon Oct 16 10:58:19 EDT 2006
      Users: 2
      jay
      iris
      Groups: 0
      Nodes: 2
      c94n01.ppd.pok.ibm.com
      c94n02.ppd.pok.ibm.com
      Job Steps: 0
```

2. To determine if any reservations will be active on the machine c94n01 before performing an hours worth of maintenance beginning at 8:00 on 10/16/2006, issue the following command:

```
llqres -b 10/16/2006 8:00 -e 10/16/2006 9:00 -h c94n01
```

You should receive output similar to the following:

Id	Owner	ST	Start Time	Duration	#Nodes
-----	---	---	-----	-----	-----
c94n16.31.r	dave	W	10/16 02:00	420	4

3. There are no job steps associated with a reservation in the following example. The short form is used for host names:

```
llqres -r -s -R c94n16.30.r
```

You should receive output similar to the following:

```
c94n16.30.r!Mon Oct 16 08:12:23 EDT 2006!load!No_Group! \
  Mon Oct 16 14:00:00 EDT
2006!120!Mon Oct 16 16:00:00 EDT 2006!no!no!ACTIVE!load! \
  Mon Oct 16 10:21:14 EDT
2006!0!!1!!load!usr!3!c94n01,c94n11,c94n12!0!
```

4. There are job steps associated with a reservation in the following example. The short form is used for host names:

```
llqres -r -s -R c94n16.31.r
```

You should receive output similar to the following:

```
c94n16.31.r!Oct Oct 16 10:35:18 EDT 2006!dave!No_Group! \
  Mon Oct 16 02:00:00 EDT
2006!420!Mon Oct 16 09:00:00 EDT 2006!no!yes!WAITING!dave! \
  Mon Oct 16 10:55:18
EDT 2006!0!!0!!4!c94n01,c94n02,c94n12,c94n16!1!c94n16.25.0
```

When the **BG_ENABLED** keyword is set to **true** in the configuration file, the following examples apply.

1. When issuing the **llqres** command, you should receive output similar to the following:

```
llqres
ID              Owner      ST  Start Time Duration #Nodes #BG C-nodes
-----
bg1dd1.62.r     ezhong    W   6/13 18:40      5      1      0
bg1dd1.61.r     ezhong    W   6/13 18:40      5      0     128
bg1dd1.60.r     ezhong    W   6/13 18:30      5      0     512
```

2. The following example returns reservation information about all machines:

```
llqres -h all
```

You should receive output similar to the following:

```
ID              Owner      ST  Start Time Duration #Nodes #BG C-nodes
-----
bg1dd1.62.r     ezhong    W   6/13 18:40      5      1      0
```

3. The following example generates a list of base partitions for R010:

```
llqres -B R010
```

You should receive output similar to the following:

```
ID              Owner      ST  Start Time Duration #Nodes #BG C-nodes
-----
bg1dd1.60.r     ezhong    A   6/13 18:30      5      0     512
```

4. The following example generates a list of all base partitions in the Blue Gene system in raw mode:

```
llqres -r -B all
```

You should receive output similar to the following:

```
bg1dd1.rchland.ibm.com.61.r!Tue 13 Jun 2006 06:28:09 PM CDT! \
  ezhong!No_Group!Tue 13 Jun 2006 06:40:00 PM
CDT!5!Tue 13 Jun 2006 06:45:00 PM CDT!no!no!WAITING!ezhong! \
  Tue 13 Jun 2006 06:28:09 PM
CDT!0!!0!!0!!1!bg1dd1.rchland.ibm.com.200.0!128!MESH!0x0x0!1!R011 \
  (J111;J113;J115;J117)
bg1dd1.rchland.ibm.com.60.r!Tue 13 Jun 2006 06:26:36 PM CDT! \
  ezhong!No_Group!Tue 13 Jun 2006 06:30:00 PM
CDT!5!Tue 13 Jun 2006 06:35:00 PM CDT!no!no!ACTIVE!ezhong! \
  Tue 13 Jun 2006 06:26:36 PM
CDT!0!!0!!0!!1!bg1dd1.rchland.ibm.com.199.0!512!MESH!1x1x1!1!R010
```

5. The following example shows a fragment of output for a system where Blue Gene support is enabled:

```
llqres -l
```

You should receive output similar to the following:

```

===== Reservation bgldd1.rchland.ibm.com.62.r =====
      ID: bgldd1.rchland.ibm.com.62.r
      Creation Time: Tue 13 Jun 2006 06:28:25 PM CDT
      Owner: ezhong
      Group: No_Group
      Start Time: Tue 13 Jun 2006 06:40:00 PM CDT
      Duration: 5 minutes
      Expected End Time: Tue 13 Jun 2006 06:45:00 PM CDT
      SHARED: no
      REMOVE_ON_IDLE: no
      Status: WAITING
      Modified By: ezhong
      Modification Time: Tue 13 Jun 2006 06:28:25 PM CDT
      Users: 0
      Groups: 0
      Nodes: 1
      bgldd1.rchland.ibm.com
      Job Steps: 1
      bgldd1.rchland.ibm.com.201.0
      BG C-nodes: 0
      BG Connection:
      BG Shape:
      BG BPs: 0
===== Reservation bgldd1.rchland.ibm.com.61.r =====
      ID: bgldd1.rchland.ibm.com.61.r
      Creation Time: Tue 13 Jun 2006 06:28:09 PM CDT
      Owner: ezhong
      Group: No_Group
      Start Time: Tue 13 Jun 2006 06:40:00 PM CDT
      Duration: 5 minutes
      Expected End Time: Tue 13 Jun 2006 06:45:00 PM CDT
      SHARED: no
      REMOVE_ON_IDLE: no
      Status: WAITING
      Modified By: ezhong
      Modification Time: Tue 13 Jun 2006 06:28:09 PM CDT
      Users: 0
      Groups: 0
      Nodes: 0
      Job Steps: 1
      bgldd1.rchland.ibm.com.200.0
      BG C-nodes: 128
      BG Connection: MESH
      BG Shape: 0x0x0
      BG BPs: 1
      R011(J111;J113;J115;J117)
===== Reservation bgldd1.rchland.ibm.com.60.r =====
      ID: bgldd1.rchland.ibm.com.60.r
      Creation Time: Tue 13 Jun 2006 06:26:36 PM CDT
      Owner: ezhong
      Group: No_Group
      Start Time: Tue 13 Jun 2006 06:30:00 PM CDT
      Duration: 5 minutes
      Expected End Time: Tue 13 Jun 2006 06:35:00 PM CDT
      SHARED: no
      REMOVE_ON_IDLE: no
      Status: ACTIVE
      Modified By: ezhong
      Modification Time: Tue 13 Jun 2006 06:26:36 PM CDT
      Users: 0
      Groups: 0
      Nodes: 0
      Job Steps: 1
      bgldd1.rchland.ibm.com.199.0
      BG C-nodes: 512

```

```

BG Connection: MESH
BG Shape: 1x1x1
BG BPs: 1
R010

```

6. The following example shows reserving resources during the time period from 18:35 to 18:45:

```
llqres -b 18:35 -e 18:45
```

You should receive output similar to the following:

ID	Owner	ST	Start Time	Duration	#Nodes	#BG	C-nodes
bg1dd1.62.r	ezhong	W	6/13 18:40		5	1	0
bg1dd1.61.r	ezhong	W	6/13 18:40		5	0	128

7. The following example shows that there are no job steps associated with the reservation. The short form is used for *hostnames*:

```
llqres -r -s c94n16.30.r
```

You should receive output similar to the following:

```

c94n16.30.r!Fri July 21 08:12:23 EDT 2006!load!No_Group! \
  Fri July 21 14:00:00 EDT
2006!120!Fri July 21 16:00:00 EDT 2006!no!no!ACTIVE!load! \
  Fri July 21 10:21:14 EDT
2006!0!!1!!load!usr!3!c94n01,c94n11,c94n12!0!!0!0x0x0!

```

8. The following example shows that there are job steps associated with the reservation. The short form is used for *hostnames*:

```
llqres -r -s c94n16.31.r
```

You should receive output similar to the following:

```

c94n16.31.r!Fri July 21 10:35:18 EDT 2006!dave!No_Group! \
  Thu Jul 22 02:00:00 EDT
2006!420!Thu Jul 22 09:00:00 EDT 2006!no!yes!WAITING!dave! \
  Fri July 21 10:55:18 EDT
2006!0!!0!!4!c94n01,c94n02,c94n12,c94n16!1!c94n16.25.0!0!0x0x0!

```

llrmres - Cancel a reservation

Purpose

llrmres – Cancels a reservation in LoadLeveler.

Syntax

```
llrmres { -? | -H | -v | [ -q ] { [-u user_list] [-g group_list]
[-h all | -h host_list] [-B all | -B base_partition_list] |
[-R all | -R reservation_list] } }
```

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- q Specifies quiet mode: print no messages other than error messages.
- u *user_list* Specifies a blank-delimited list of user IDs. Reservations to be canceled are owned by one of these users.
- g *group_list* Specifies a blank-delimited list of LoadLeveler groups. Reservations to be canceled are owned by one of these groups.
- h **all**
- h *host_list* Specifies a blank-delimited list of machines or all machines. Reservations to be canceled use one or more of these machines.
- B **all**
- B *base_partition_list* Specifies a blank-delimited list of Blue Gene base partitions or all Blue Gene base partitions. Reservations to be canceled use one or more of these Blue Gene base partitions.
- R **all**
- R *reservation_list* Is a blank-delimited list of reservation identifiers to be canceled. A LoadLeveler administrator can specify the reserved word **all** to cancel all reservations in the system. Nonadministrators can specify the reserved word **all** to cancel all of the reservations that they own. The format of a full LoadLeveler reservation identifier is [*host.*]*rid*[.*r*].
 where:
 - *host* is the name of the machine that assigned the reservation identifier.
 - *rid* is the number assigned to the reservation when it was created. An *rid* is required.
 - *r* indicates that this is a reservation ID (*r* is optional).
 The reservation identifier may be specified in an abbreviated form, *rid*[.*r*], when the command is invoked on the same machine that

assigned the reservation identifier. In this case, LoadLeveler will use the local machine's host name to construct the full reservation identifier.

Description

The **llrmres** command is for LoadLeveler administrators and owners of a reservation. Owners of reservations can cancel their own reservations. A LoadLeveler administrator can cancel any reservation. The state of a job step will not be changed directly by the cancellation of a reservation. Reservations satisfying all criteria specified by the **-u**, **-g**, **-h**, or **-B** flags will be canceled if more than one of these options are present. The **-h** and **-B** flags are mutually exclusive. The **llqres** command can be used to see the status of the reservation.

This command is for the BACKFILL scheduler only.

Examples

1. To request to cancel all reservations owned by user ID iris that use machine c188f2n01, issue the following command. Note that this request can be made either by an administrator or the user iris.

```
llrmres -u iris -h c188f2n01
```

You should receive a response similar to the following:

The request to remove reservations has been sent to the central manager.

2. To request that the LoadLeveler administrator cancel all reservations, issue:

```
llrmres -R all
```

You should receive a response similar to the following:

The request to remove reservations has been sent to the central manager.

3. To request that the LoadLeveler administrator cancel all reservations of Blue Gene resources, issue:

```
llrmres -B all
```

You should receive a response similar to the following:

The request to remove reservations has been sent to the central manager.

4. To request that the LoadLeveler administrator cancel all reservations on nodes that are not Blue Gene compute nodes (C-nodes), issue:

```
llrmres -h all
```

You should receive a response similar to the following:

The request to remove reservations has been sent to the central manager.

Security

LoadLeveler administrators and users can issue this command.

llrnscheduler - Run the central manager's scheduling algorithm

Purpose

llrnscheduler – Runs the central manager's scheduling algorithm when the internal scheduling interval is disabled.

Syntax

```
llrnscheduler [-?] | [-H] | [-v] | [-q]
```

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- q Specifies quiet mode: print no messages other than error messages.

Description

The **llrnscheduler** command is used to run the central manager's scheduling algorithm when the internal scheduling interval has been disabled so that an external program can control when the central manager attempts to schedule job steps. The **llrnscheduler** command sends a request to the central manager to run the scheduling algorithm. The central manager's scheduling algorithm will run only once each time the **llrnscheduler** command is invoked. Each time the scheduling algorithm runs, the central manager will schedule as many job steps as the current available resources allow.

The request to run the scheduling algorithm is ignored if the internal scheduling interval has not been disabled by setting the **NEGOTIATOR_INTERVAL** configuration keyword to 0. If **NEGOTIATOR_INTERVAL** is set to 0, the **llstatus** command will report that the scheduler interval is disabled.

Security

LoadLeveler administrators can issue this command.

llstatus - Query machine status

Purpose

llstatus – Returns status information about machines in the LoadLeveler cluster.

Syntax

```
llstatus [-?] [-H] [-v] [-W] [-R] [-F] [-M] [-l] [-a] [-C] [-b]
          [-B {base_partition_list | all}] [-P {partition_list | all}]
          [-X {cluster_list | all}] [-f category_list] [-r category_list]
          [-h hostlist | hostlist]
```

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- W Specifies that the width of columns in tabular output will be increased to fit the widest entry.
- R Lists the machine consumable resources associated with each machine for which status is requested. This option should not be used with any other option.
- F Lists all of the floating consumable resources associated with the LoadLeveler cluster. This option should not be used with any other option.
- M Lists the Multiple Chip Modules (MCMs) available on a machine, where:
 - Available CPUs**
Are the CPU IDs of the CPUs available for LoadLeveler on this MCM.
 - Used CPUs**
Are the CPU IDs of the CPUs used by LoadLeveler jobs on this MCM.
 - Adapters**
Are the switch adapters connected to this MCM of the form:
[used windows/available windows, used rCxt blocks/available \ rCxtBlocks]
 - where:
 - used windows**
Is the number of windows used.
 - available windows**
Is the total number of windows on the adapter.
 - used rCxt blocks**
Is the number of rCxt blocks used.
 - available rCxt blocks**
Is the total number of rCxt blocks on the adapter.
 - Total Tasks**
Is the total number of tasks that are running using CPUs of this MCM.

llstatus

Note that the < > notation will be used to list individual CPU IDs instead of the CPU count () notation when the **RSET_SUPPORT** configuration file keyword is set to **RSET_CONSUMABLE_CPUS** or **RSET_MCM_AFFINITY**.

- l** Specifies that a long listing be generated for each machine for which status is requested. If **-l** is *not* specified, the standard list is generated.
- a** Displays information for each virtual adapter followed by information for each physical adapter it manages. This flag also displays the port number on each InfiniBand adapter port.
- C** Displays cluster stanza information defined in the administration file. Only fields that contain data are displayed. If **-C** is specified without the **-X** flag, cluster stanza information will be reported from one outbound local Schedd. If the **-X** and **-C** flags are specified, cluster stanza information will be reported from an inbound Schedd in each available remote cluster specified with the **-X** flag. If the **-h** flag is specified, cluster stanza information will only be reported from the specified hosts.
- b** Displays Blue Gene system information. It is valid only when specified as a single option or in combination with the **-l** flag. When used as single option, **llstatus** displays a short listing of information on the Blue Gene system. When used the **-l** flag, **llstatus** displays a detailed listing of the components of the Blue Gene system.
- P** {*partition_list* | **all**}
Displays Blue Gene partition information. It is valid only when specified as a single option. When this flag is specified, **llstatus** displays information on all Blue Gene partitions in the *partition_list*.
partition_list Is a blank-delimited list of Blue Gene partitions.
all Is the reserved word indicating that the command is to return information for all partitions.
- B** {*base_partition_list* | **all**}
Displays Blue Gene base partition information. It is valid only when specified as a single option. When this flag is specified, **llstatus** displays information about **all** Blue Gene base partitions in the *base_partition_list*.
base_partition_list Is a blank-delimited list of Blue Gene base partitions.
all Is the reserved word indicating that the command is to return information for all base partitions.
- X** {*cluster_list* | **all**}
Indicates that you can specify the **-X** flag with either:
cluster_list Specifies a blank-delimited list of clusters for which status is requested.
all Is a reserved word which specifies that status is requested for all accessible clusters.
- f** *category_list*
Is a blank-delimited list of categories you want to query. Each category you specify must be preceded by a percent sign. The *category_list* cannot contain duplicate entries. This flag allows you to create a customized version of the standard **llstatus** listing. The output fields produced by this

flag all have a fixed length. The output is displayed in the order in which you specify the categories. *category_list* can be one or more of the following:

%a	Hardware architecture
%act	Number of job steps dispatched by the Schedd daemon on this machine
%cm	Custom Metric value
%cpu	Number of CPUs on this machine
%d	Available disk space in the LoadLeveler execute directory
%i	Number of seconds since last keyboard or mouse activity
%inq	Number of job steps in the job queue of this Schedd machine
%l	Berkeley one-minute load average
%m	Physical memory on this machine
%mt	Maximum number of initiators that can be used simultaneously on this machine
%n	Machine name
%o	Operating system on this machine
%r	Number of initiators used by the startd daemon on this machine
%sca	Availability of the Schedd daemon
%scs	State of the Schedd daemon
%sta	Availability of the startd daemon
%sts	State of the startd daemon
%v	Available swap space (free paging space) of this machine
%X	Local cluster name

-r *category_list*

Is a blank-delimited list of categories you want to query. Each category you specify must be preceded by a percent sign. The *category_list* cannot contain duplicate entries. This flag allows you to create a customized version of the standard **llstatus** listing. The output produced by this flag is considered raw, in that the fields can be variable in length. The output is displayed in the order in which you specify the formats. Output fields are separated by an exclamation point (!). *category_list* can be one or more of the categories listed under the **-f** flag.

-h *hostlist*

hostlist Is a blank-delimited list of machines for which status is requested.

If the **-X** flag is specified in combination with a *hostlist*, the **-h** flag must be specified. For example:

```
llstatus -X my_cluster1 my_cluster2 -h c94n13 c94n14
```

Description

If you have more than a few machines configured for LoadLeveler, consider redirecting the output to a file when using the **-l** flag.

llstatus

Each machine periodically updates the central manager with a snapshot of its situation. Since the information returned by using **llstatus** is a collection of such snapshots, all taken at varying times, the total picture may not be completely consistent.

In most cases, if a *hostlist* is not specified, all machines are queried. However, if the **-X** and **-C** flags are specified without a *hostlist*, the command will run on one inbound Schedd in the remote cluster. In this case, the Schedd version information from the inbound Schedd in the remote cluster will be displayed.

Certain resources such as remote direct-memory access (RDMA) have their available value always calculated by startd. Available **ConsumableCpus** resources are calculated by startd if the value is specified as **all** in the administration file. When the value of a resource is calculated by startd, the **llstatus** command appends a plus (+) sign to the resource name in the output reports. Resources that are automatically created, such as RDMA, have a less than (<) sign appended to them.

Examples

1. This example generates the standard listing where there are two nodes in the cluster. The standard listing is generated when you do *not* specify the **-l** option with the **llstatus** command.

```
llstatus
```

You should receive output similar to the following:

```
Name                               Schedd InQ Act Startd Run LdAvg Idle Arch OpSys
k10n09.ppd.pok.ibm.com             Avail  3  1 Run      1 2.72  0 R6000 AIX53
k10n12.ppd.pok.ibm.com             Avail  0  0 Idle     0 0.00 365 R6000 AIX53

R6000/AIX53                         2 machines  3 jobs  1 running
Total Machines                     2 machines  3 jobs  1 running
```

```
The Central Manager is defined on k10n09.ppd.pok.ibm.com
```

```
The BACKFILL scheduler is in use
```

```
Cluster name is cluster2
```

```
All machines on the machine_list are present.
```

The **standard listing** includes the following fields:

Act Number of job steps dispatched by the Schedd daemon on this machine.

Arch The hardware architecture of the machine as listed in the configuration file.

Idle The number of seconds since keyboard or mouse activity in a login session was detected. Highest number displayed is 9999.

InQ Number of job steps in the job queue of this Schedd machine.

LdAvg Berkeley one-minute load average on this machine.

Name Hostname of the machine.

OpSys

The operating system on this machine.

Run The number of initiators used by the startd daemon to run LoadLeveler jobs on this machine. One initiator is used for each serial job step and one initiator is used for each task of a parallel job step.

Schedd

State of the Schedd daemon, which can be one of the following:

- Down
- Drned (Drained)
- Drning (Draining)
- Avail (Available)

For more information, see “The Schedd daemon” on page 9.

Startd State of the startd daemon, which can be:

- Busy
- Down
- Drned (Drained)
- Drning (Draining)
- Flush
- Idle
- None
- Run (Running)
- Suspnd (Suspend)

For more information, see “The startd daemon” on page 10.

Total Machines

The standard listing includes the following summary fields:

jobs The number of job steps in LoadLeveler job queues.

machines

The number of machines in the cluster that have made a status report to the Central Manager.

running

The number of initiators used by all the startd daemons in the LoadLeveler cluster. One initiator is used for each serial job step. One initiator is used for each task of a parallel job step.

The standard listing also contains summary information for the cluster such as the type of scheduler and the cluster name.

2. This example generates the long listing. The long listing is generated when you specify the **-l** flag with the **llstatus** command. See Appendix B, “Sample command output,” on page 673 for sample output of long listings.

```
llstatus -l c271f2rp02
```

The **long listing** includes the following fields:

Adapter

Network adapter information associated with this machine.

- For a switch adapter, the information format is:

```
adapter_name(network_type, interface_name,
interface_address, multilink_address,
switch_node_number or
adapter_logical_id, available_adapter_windows/
total_adapter_windows, unused rCxt blocks/
total rCxt blocks, adapter_fabric_connectivity,
adapter_state)
```

For example:

```
|
|
|
|
|
|
|
```

llstatus

```
Adapter = networks(striped,c60f1rp01m10.ppd.pok.ibm.com, \
10.10.10.1,,-1,32/32,1596/1596 rCxt Blks,1,READY) \
en0(ethernet,c60f1rp01.ppd.pok.ibm.com,9.114.88.65,) \
network1(aggregate,,,10.10.10.1,-1,32/32,1596/1596 rCxt Blks,1, \
READY)m10(multilink,c60f1rp01m10.ppd.pok.ibm.com,10.10.10.1,)
```

Possible values for adapter_state are:

ErrAdapter

The adapter information is incorrect. More information might be available in the network table API log.

ErrDown

The adapter has been reported as down.

ErrInternal

An error occurred while accessing the adapter. More information might be available in the network table API log and if the **D_FULLDEBUG** and **D_ADAPTER** flags are specified for **STARTD_DEBUG**.

ErrNotConnected

The adapter is not connected to the network.

ErrNotInitialized

An error occurred initializing access to the adapter. More information might be available if the **D_FULLDEBUG** and **D_ADAPTER** flags are specified for **STARTD_DEBUG**.

ErrNTBL

An error occurred while accessing the network table API. More information might be available if the **D_FULLDEBUG** and **D_ADAPTER** flags are specified for **STARTD_DEBUG**.

ErrNTBLVersion

The version of the available network table API is not compatible with the version required by LoadLeveler.

ErrPerm

The network table API reported that LoadLeveler does not have permission to access the adapter. More information might be available in the network table API log.

ErrPNSD

An error occurred in the network table API. More information might be available in the network table API log.

ErrType

The network table API reported that the adapter is not a switch adapter. Check the adapter configuration.

MachineDown

The machine to which the adapter is attached could not be reached by LoadLeveler to query status.

NOT READY

An unspecified problem occurred when accessing the adapter state. More information might be available if the **D_FULLDEBUG** and **D_ADAPTER** flags are specified for **STARTD_DEBUG**.

READY

The adapter can be used for communication.

- For non-switch adapters, the format is:

```
adapter_name(network_type, interface_name, interface_address,
multilink_address)
```

Arch Hardware architecture of this machine.

AvailableClasses

List of available classes and the associated number of available initiators on this machine.

Completed

The number of job steps in this state on this Schedd machine.

Config Time Stamp

Date and time of last configuration or reconfiguration.

ConfiguredClasses

List of configured classes and the associated number of configured initiators on this machine.

ConsumableResources

List of consumable resources associated with this machine. Each element of this list has the format: resource_name(available, total).

Note: The individual CPU ID < > notation will be used to list individual CPU IDs instead of the CPU count () notation for machines where the **RSET_SUPPORT** configuration file keyword is set to **RSET_MCM_AFFINITY** or **RSET_CONSUMABLE_CPUS**. The CPU count () notation will be used when the **RSET_SUPPORT** configuration file keyword is set to **RSET_USER_DEFINED** or **RSET_NONE**, or if this keyword is not specified in the configuration file.

CONTINUE

The expression, defined following C conventions in the configuration file, that evaluates to true or false (T or F). This determines whether suspended jobs are continued on this machine.

Cpus Number of CPUs on this machine.

CustomMetric

This value can be the number assigned to the **CUSTOM_METRIC** keyword or the exit code of the executable associated with the **CUSTOM_METRIC_COMMAND** keyword or the default value of 1.

Disk Available space, in kilobytes (less 512KB) in LoadLeveler's execute directory on this machine.

DrainedClasses

List of classes which have been drained. If a job step is in a class named on this list, that job step will not start on this machine.

DrainingClasses

List of classes which are currently being drained on this machine. If a job step is in a class named on this list, that job step will not start on this machine.

Entered Current State

Date and time when machine state was set.

FabricConnectivity

Represents the current state of connectivity between the machine and the switch through the switch adapters. The format of the field is: network_id: connectivity, network_id: connectivity... where

lstatus

connectivity is either 1 or 0. A value of 1 indicates an active connection from the machine to a given network_id through one of the switch adapters.

If a machine does not have switch adapters, the **FabricConnectivity** field has no meaning and should be ignored by the user.

Feature

Set of all features on this machine.

FreeLargePageMemory

Free Large Page memory.

In LoadLeveler for Linux, the **FreeLargePageMemory** field has no meaning and should be ignored by the user.

FreeRealMemory

Free real memory, in megabytes, on this machine. This value should track closely with the "fre" value of the **vmstat** command and the "free" value of the **svmon -G** command whose units are 4KB blocks.

Held The number of job steps in this state on this Schedd machine.

Idle The number of job steps in this state on this Schedd machine.

Keyboard Idle

Number of seconds since last keyboard or mouse activity.

KILL The expression, defined following C conventions in the configuration file, that evaluates to true or false (T or F). This determines whether jobs running on this machine should be sent the SIGKILL signal.

LargePageMemory

Configured Large Page physical memory.

In LoadLeveler for Linux, the **LargePageMemory** field has no meaning and should be ignored by the user.

LargePageSize

The size of a Large Page memory block.

In LoadLeveler for Linux, the **LargePageSize** field has no meaning and should be ignored by the user.

LoadAvg

Berkely one-minute load average on machine.

Machine

Fully qualified name of the machine.

Machine Mode

The type of job this machine can run. This can be: **batch**, **interactive**, or **general**.

MACHPRIO

Actual expression that determines machine priority, defined in the configuration file.

MasterMachPriority

The machine priority for the parallel master node.

Max_Starters

Maximum number of initiators that can be used simultaneously on this machine.

Mcms The MCMs information associated with this machine has the format:

mcm_info ... mcm_info

The format of *mcm_info* is:

```
MCMnumber Available Cpus: < cpulist > (Total Cpus)
Used Cpus: < cpulist > (Total Cpus)
Adapters: adapater_info .... adapter_info
Total Tasks: (Tasks)
```

where:

Available Cpus

Are the CPUs available for LoadLeveler on this MCM.

Used Cpus

Are the CPUs used by LoadLeveler jobs from this MCM.

Adapters

Are the switch adapters connected to this MCM.

Total tasks

Is the total number of tasks that are running using CPUs of this MCM.

where *number* is the MCM sequence number:

cpulist Is a blank-delimited list of individual CPU IDs or CPU ranges, or a combination of both CPU IDs and CPU ranges associated with the MCM.

adapter_info

Has the format:

```
[used windows/available windows, used
memory/available memory]
```

where:

used windows

Is the total number of windows used.

available windows

Is the total number of available windows.

used memory

Is the total memory used. It is the number of rCxt blocks for the High Performance Switch adapter.

available memory

Is the total available memory. It is the number of rCxt blocks for the High Performance Switch adapter.

Tasks Is the total number of tasks that are running using the CPUs of this MCM.

Memory

Regular physical memory, in megabytes, on this machine.

Name Hostname of the machine.

OpSys

Operating system on this machine.

PagesFreed

Pages freed per second. This value corresponds to the "fr" value of the **vmstat** command output.

In LoadLeveler for Linux, the **PagesFreed** field has no meaning and should be ignored by the user.

PagesPaged In

Pages paged in from paging space per second. This value corresponds to the "pi" value of the **vmstat** command output.

In LoadLeveler for Linux, the **PagesPagedIn** field has no meaning and should be ignored by the user.

PagesPagedOut

Pages paged out to paging space per second. This value corresponds to the "po" value of the **vmstat** command output.

In LoadLeveler for Linux, the **PagesPagedOut** field has no meaning and should be ignored by the user.

PagesScanned

Pages scanned by the page-replacement algorithm per second. This value corresponds to the "sr" value of the **vmstat** command output.

In LoadLeveler for Linux, the **PagesScanned** field has no meaning and should be ignored by the user.

Pending

The number of job steps in this state on this Schedd machine.

Pool The identifier of the pool where this startd machine is located.

Prestarted_Starters

The maximum number of Prestarted Starters that can be started on this machine at any time.

Remove Pending

The number of job steps in this state on this Schedd machine.

ReservationPermitted

Indicates whether or not the node can be reserved. It is displayed as T or F (true or false).

Reservations

The IDs of reservations that will use the node now or in the future.

RSetSupportType

Indicates the type of RSet support set up on a machine. Possible values are:

RSET_CONSUMABLE_CPUS

Creates and attaches RSets for tasks with the number of CPUs derived from **ConsumableCPUs**.

RSET_MCM_AFFINITY

Creates and attaches RSets for tasks to satisfy memory and affinity requests.

RSET_NONE

Indicates that LoadLeveler RSet support is not available.

RSET_USER_DEFINED

Attaches user-created RSets to Tasks.

Running

The number of initiators used by the startd daemon to run

LoadLeveler jobs. One initiator is used for each serial job step. One initiator is used for each task of a parallel job step.

Running steps

The list of job steps currently running on this machine.

ScheddAvail

Flag indicating if machine is running a Schedd daemon (0=no, 1=yes).

ScheddRunning

The number of job steps submitted to this machine that are running somewhere in the LoadLeveler cluster.

ScheddState

The state of the Schedd daemon on this machine, which can be one of the following:

- Down
- Drned (Drained)
- Drning (Draining)
- Avail (Available)

Speed Speed associated with the machine.

SMT Indicates whether the Simultaneous Multi-Threading (SMT) function is turned on, off, or is not supported in the running machine. Valid values are **Enabled**, **Disabled**, or **Not Supported**.

START

The expression, defined following C conventions in the configuration file, that evaluates to true or false (T or F). This determines whether jobs can be started on this machine.

StartdAvail

Flag indicating if machine is running a startd daemon (0=no, 1=yes).

Starting

The number of job steps in this state on this Schedd machine.

State State of the startd daemon, which can be:

- Busy
- Down
- Drained
- Draining
- Flush
- Idle
- None
- Running
- Suspend

For more information, see “The startd daemon” on page 10.

Subnet

The TCP/IP subnet that this machine resides on.

SUSPEND

The expression, defined following C conventions in the configuration file, that evaluates to true or false (T or F). This determines whether running jobs should be suspended on this machine.

SYSPRIO

Actual expression that determines overall system priority of a job step. Defined in the configuration file.

TimeStamp

The date and time the central manager last received a status update from this Schedd machine.

Tmp Available space, in kilobytes (less than 512 KB) in the /tmp directory on this machine.

Total Jobs

The number of total job steps submitted to this Schedd machine.

TotalMemory

The sum of configured regular and Large Page memory.

Unexpanded

The number of job steps in this state on this Schedd machine.

VACATE

The expression, defined following C conventions in the configuration file, that evaluates to true or false (T or F). This determines whether suspended jobs are vacated on this machine.

Virtual Memory

Available swap space (free paging space) in kilobytes, on this machine.

- 3. This example generates a listing of cluster information defined in the administration file and Schedd version for **cluster2**. Only fields with data are displayed.

```
llstatus -X cluster2 -C
```

The output representing a cluster is delineated with a cluster header and Schedd version information from a remote cluster similar to the following:

```
===== Cluster cluster2 =====
llstatus: Sending request to Schedd "c188f2n08.ppd.pok.ibm.com" in
cluster "cluster2"

cluster2: type = cluster
Local = True
inbound_schedd_port = 9605
secure_schedd_port = 9607
multicluster_security = NOT_SET
ssl_cipher_list = ALL:eNULL:!aNULL
inbound_hosts = c188f2n08.ppd.pok.ibm.com
outbound_hosts = c188f2n08.ppd.pok.ibm.com
exclude_classes = badtesters(cluster3) OKtesters(cluster1)
cluster1: type = cluster
Local = False
inbound_schedd_port = 9605
secure_schedd_port = 9607
multicluster_security = NOT_SET
ssl_cipher_list = ALL:eNULL:!aNULL
inbound_hosts = c188f2n02.ppd.pok.ibm.com
outbound_hosts = c188f2n02.ppd.pok.ibm.com
exclude_users = load1(cluster2)
exclude_groups = april(cluster3)
include_classes = No_Class
cluster3: type = cluster
Local = False
inbound_schedd_port = 1966
secure_schedd_port = 9607
multicluster_security = NOT_SET
ssl_cipher_list = ALL:eNULL:!aNULL
inbound_hosts = c94n02.ppd.pok.ibm.com
```

```

outbound_hosts = c94n02.ppd.pok.ibm.com(cluster1)
                  c94n06.ppd.pok.ibm.com(cluster2)

clusterB schedd gateway Version:
c188f2n08.ppd.pok.ibm.com 3.3.2.5 rmer 2006/08/03 \
RHEL 4.0 132

```

4. This example generates a listing of all of the consumable resources associated with all of the machines in the LoadLeveler cluster.

```
llstatus -R
```

You should receive output similar to the following:

```

Machine                               Consumable Resource(Available, Total)
-----
c209f1n01.ppd.pok.ibm.com             ConsumableCpus(4,4)+ ConsumableMemory \
(1.000 gb,1.000 gb) n01_res(123,500)
c209f1n02.ppd.pok.ibm.com             ConsumableCpus< 0 1 2 3 4>< 0 1 2 3 4 > \
n02_res(123,500) Frame5(10,10)
c209f1n05.ppd.pok.ibm.com             ConsumableCpus(4,4)+ ConsumableMemory \
(1.000 gb,1.000 gb) spice2g6(250,360)

```

Resources with "+" appended to their names have the Total value reported from Startd.

Note: The individual CPU ID < > notation will be used to list individual CPU IDs instead of the CPU count () notation for machines where the **RSET_SUPPORT** configuration file keyword is set to **RSET_MCM_AFFINITY** or **RSET_CONSUMABLE_CPUS**. The CPU count () notation will be used when the **RSET_SUPPORT** configuration file keyword is set to **RSET_USER_DEFINED** or **RSET_NONE**, or if this keyword is not specified in the configuration file.

5. This example generates a listing of information related to MCMs of machines in the LoadLeveler cluster.

```
llstatus -M
```

You should receive output similar to the following:

```

Machine                               MCM details
-----
c61f2sq01.ppd.pok.ibm.com
MCM0
  Available Cpus :< 0-15 >(16)
  Used Cpus      :< >(0)
  Adapters       :
  Total Tasks    : (0)
MCM1
  Available Cpus :< 16-29 >(14)
  Used Cpus      :< 16-27 >(12)
  Adapters       :sn1[16/16,798/798 rCxt Blks]
                  sn0[12/16,790/798 rCxt Blks]
  Total Tasks    : (4)
c61f2sq02.ppd.pok.ibm.com
MCM0
  Available Cpus :< 0-1 >(2)
  Used Cpus      :< >(0)
  Adapters       :
  Total Tasks    : (0)
MCM1

```

llstatus

```
Available Cpus :< 2-3 >(2)
Used Cpus      :< >(0)
Adapters      :
Total Tasks    :(0)
```

Note: The **-M** option will list the MCM information only when the **RSET_SUPPORT** configuration file keyword is set to **RSET_MCM_AFFINITY**.

- This example generates a listing of all of the floating consumable resources associated with all of the machines in the LoadLeveler cluster. This option should not be specified with any other option.

```
llstatus -F
```

You should receive output similar to the following:

Floating Resource	Available	Total
EDA_licenses	20	29
Frame5	15	20
WorkBench6	5	7
XYZ_software	6	6

- This example generates a customized and formatted standard listing.

```
llstatus -f %n %scs %inq %m %v %sts %l %o
```

You should receive output similar to the following:

Name	Schedd	InQ	Memory	FreeVMemory	Startd	LdAvg	OpSys
115.pok.ibm.com	Avail	0	128	22708	Run	0.23	AIX53
116.pok.ibm.com	Avail	3	224	16732	Run	0.51	AIX53
R6000/AIX53			2 machines	3 jobs	3	running	
Total Machines			2 machines	3 jobs	3	running	

The Central Manager is defined on 115.pok.ibm.com

The BACKFILL scheduler is in use

All machines on the machine_list are present.

- This example generates a customized and unformatted (raw) standard listing.
Customized, Unformatted Standard Listing: A customized Output fields are separated by an exclamation point (!).

```
llstatus -r %n %scs %inq %m %v %sts %l %o
```

You should receive output similar to the following:

```
115.pok.ibm.com!Avail!0!128!22688!Running!0.14!AIX53
116.pok.ibm.com!Avail!3!224!16668!Running!0.37!AIX53
```

- This example generates a listing containing information about the status of adapters associated with all of the machines in the LoadLeveler cluster:

```
llstatus -a
```

You should receive output similar to the following:

```

c271f2rp02.ppd.pok.ibm.com
ml0(multilink,c271f2san02.ppd.pok.ibm.com,10.10.10.6,)
networks(striped,c271f2san02.ppd.pok.ibm.com,10.10.10.6,,-1,500/512, \
1474/1596 rCxt Blks,1,READY)
network1(aggregate,,10.10.10.6,-1,500/512,500M/512M,1,READY)
sn0(switch,c271f2s0n02.ppd.pok.ibm.com,192.168.0.6,10.10.10.6,2,250 \
/256,737/798 rCxt Blks,1,READY,MCM0)
sn1(switch,c271f2s1n02.ppd.pok.ibm.com,192.168.1.6,10.10.10.6,0,250 \
/256,737/798 rCxt Blks,1,READY,MCM0)
en0(ethernet,c271f2rp02.ppd.pok.ibm.com,9.114.175.82,)

```

- For a switch adapter, the information format is:

```

adapter_name(network_type, interface_name, interface_address,
multilink_address, switch_node_number or
adapter_logical_id, available_adapter_windows/total_adapter_windows,
unused rCxt blocks/total rCxt blocks,
adapter_fabric_connectivity, adapter_state[,adapter mcm id])

```

- For non-switch adapters, the format is:

```

adapter_name(network_type, interface_name, interface_address,
multilink_address)

```

- For InfiniBand, this example displays the port number on each InfiniBand adapter. The **llstatus** command does not show port information for adapters that are not InfiniBand adapters:

```
llstatus -a
```

You should receive output similar to the following:

```

=====
c171f6sq08.ppd.pok.ibm.com
ehca0(InfiniBand,,,-1,0/0,0/0 rCxt Blks,101,READY)
eth0(ethernet,c171f6sq08.ppd.pok.ibm.com,9.114.136.59,)
network1833865768265265971s(striped,,,-1,64/64,0/0 rCxt Blks,101, \
READY)
network18338657682652659714(aggregate,,,-1,64/64,0/0 rCxt Blks,1, \
READY)
ib1(InfiniBand,192.168.9.59,192.168.9.59,,2,64/64,0/0 rCxt Blks,1, \
READY,2)
network18338657682652659712(aggregate,,,-1,64/64,0/0 rCxt Blks,1, \
READY)
ib0(InfiniBand,192.168.8.59,192.168.8.59,,2,64/64,0/0 rCxt Blks,1, \
READY,1)

=====
c171f6sq07.ppd.pok.ibm.com
ehca0(InfiniBand,,,-1,0/0,0/0 rCxt Blks,101,READY)
eth0(ethernet,c171f6sq07.ppd.pok.ibm.com,9.114.136.58,)
network1833865768265265971s(striped,,,-1,64/64,0/0 rCxt Blks,101, \
READY)
network18338657682652659714(aggregate,,,-1,64/64,0/0 rCxt Blks,1, \
READY)
ib1(InfiniBand,192.168.9.58,192.168.9.58,,1,64/64,0/0 rCxt Blks,1, \
READY,2)
network18338657682652659712(aggregate,,,-1,64/64,0/0 rCxt Blks,1, \
READY)
ib0(InfiniBand,192.168.8.58,192.168.8.58,,1,64/64,0/0 rCxt Blks,1, \
READY,1)

```

Note: The adapter MCM ID will be printed only for High Performance Switch adapters and only when the **RSET_SUPPORT** configuration file keyword is set to **RSET_MCM_AFFINITY**. A value of MCM-1 for this field means that none of the CPUs from the physical MCM, where that adapter is connected to, is part of the machine's partition.

llstatus

10. This example generates a listing containing information about the status of the Blue Gene system in the LoadLeveler cluster.

```
llstatus -b
```

You should receive output similar to the following:

```
Name          Base Partitions  c-nodes   InQ   Run
BGL            1x2x4            8x16x32   4     1
```

11. This example generates information for base Blue Gene partition R001:

```
llstatus -B R001
```

You should receive output similar to the following:

```
Base Partition Id: R001
```

```
Base Partition State: UP
```

```
Location: (0,0,3)
```

```
C-node Memory: 512 mb
```

```
Sub Divided Busy: True
```

```
Node Card List:
```

```
NodeCardId=J102  NodeCardState=UP  Quarter=Q1  PartitionState=NONE \
Partition=NONE
NodeCardId=J104  NodeCardState=UP  Quarter=Q1  PartitionState=NONE \
Partition=NONE
NodeCardId=J106  NodeCardState=UP  Quarter=Q1  PartitionState=READY \
Partition=R001_J106_32
NodeCardId=J108  NodeCardState=UP  Quarter=Q1  PartitionState=NONE \
Partition=NONE
NodeCardId=J111  NodeCardState=UP  Quarter=Q2  PartitionState=NONE \
Partition=NONE
NodeCardId=J113  NodeCardState=UP  Quarter=Q2  PartitionState=NONE \
Partition=NONE
NodeCardId=J115  NodeCardState=UP  Quarter=Q2  PartitionState=NONE \
Partition=NONE
NodeCardId=J117  NodeCardState=UP  Quarter=Q2  PartitionState=NONE \
Partition=NONE
NodeCardId=J203  NodeCardState=UP  Quarter=Q3  PartitionState=NONE \
Partition=NONE
NodeCardId=J205  NodeCardState=UP  Quarter=Q3  PartitionState=NONE \
Partition=NONE
NodeCardId=J207  NodeCardState=UP  Quarter=Q3  PartitionState=NONE \
Partition=NONE
NodeCardId=J209  NodeCardState=UP  Quarter=Q3  PartitionState=NONE \
Partition=NONE
NodeCardId=J210  NodeCardState=UP  Quarter=Q4  PartitionState=NONE \
Partition=NONE
NodeCardId=J212  NodeCardState=UP  Quarter=Q4  PartitionState=NONE \
Partition=NONE
NodeCardId=J214  NodeCardState=UP  Quarter=Q4  PartitionState=NONE \
Partition=NONE
NodeCardId=J216  NodeCardState=UP  Quarter=Q4  PartitionState=NONE \
Partition=NONE
```

12. This example generates information for Blue Gene partition R001_J106_32:

```
llstatus -P R001_J106_32
```

You should receive output similar to the following:

```
Partition Id: R001_J106_32
```

```
Partition State: READY
```

```
Description: Generated via genSmallBlock
```

```
Owner: blocksom
```

```
Connection: MESH
```

```
Size: 32
```

```
Shape: 0x0x0
```

```
Mode: COPROCESSOR
```

```
MloaderImage: /bgl/BlueLight/dd1driver/bglsys/bin/mmcs-mloader.rts
```

```
BlrtsImage: /bgl/BlueLight/dd1driver/bglsys/bin/rts_hw.rts
```

```
LinuxImage: /bgl/BlueLight/dd1driver/bglsys/bin/zImage.elf
```

```
RamDiskImage: /bgl/BlueLight/ddldriver/bglsys/bin/ramdisk.elf
Small Partitions: True
Base Partition List: R001
Node Card List: J106
```

13. The following example shows output for the **llstatus -l** command that displays the SMT state for the machine:

```
=====
Name = devf1n01.clusters.com
Machine = devf1n01.clusters.com
Arch = ppc64
OpSys = Linux2
.
.
.
Feature = SMT
.
.
.
SMT = Enabled | Disabled | Not Supported
TimeStamp = Fri 16 Jun 2006 04:44:06 PM CST
.
.
.
```

Security

LoadLeveler administrators and users can issue this command.

llsubmit - Submit a job

Purpose

llsubmit – Submits a job to LoadLeveler to be dispatched based upon job requirements in the job command file.

Syntax

```
llsubmit [-H] [-?] [-v] [-q] [-X {cluster_list | any}] [cmdfile | - ]
```

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- q Specifies quiet mode: print no messages other than error messages.
- X {*cluster_list* | **any**}
 - Is a blank-delimited list of cluster names or the reserved word **any** where:
 - A single cluster name indicates that the job is to be submitted to that cluster.
 - A list of multiple cluster names indicates that the job is to be submitted to one of the clusters as determined by the installation exit **CLUSTER_METRIC**.
 - The reserved word **any** indicates the job is to be submitted to any cluster defined by the installation exit **CLUSTER_METRIC**.

If a *cluster_list* is specified with the -X option on the **llsubmit** command, the specified *cluster_list* takes precedence over any *cluster_list* statements already specified in the job command file.

cmdfile

Is the name of the job command file containing LoadLeveler commands.

- Specifies that LoadLeveler commands that would normally be in the job command file are read from stdin. When entry is complete, press Ctrl-D to end the input.

Description

- Users with **uid** or **gid** equal to 0 are not allowed to issue the **llsubmit** command.
- When a LoadLeveler job ends, you may receive UNIX mail notification indicating the job exit status. For example, you could get the following mail message:


```
Your LoadLeveler job
myjob1
exited with status 139.
```

The return code 139 is from the user's job, and is not a LoadLeveler return code.
- For more information on writing a program to filter job scripts, see "Filtering a job script" on page 70.
- The **llsubmit** command will display an error and fail to submit the job if the **resources** keyword in the job command file does not match the resources to be enforced and LoadLeveler is set to check for the **resources** specification. For more information, see "Defining usage policies for consumable resources" on page 56.

- If the `LL_RES_ID` environment variable is set, the **lsubmit** command will set the requested reservation ID of the submitted job steps using the value of the `LL_RES_ID` environment variable. When the central manager receives the job steps from the Schedd, it will bind the job steps to a reservation, if specified. If the job steps cannot be bound to the reservation, they will be placed in the **NotQueued** state and the requested reservation ID will keep the same value. If the value of `LL_RES_ID` is set to blank, it will be treated as if it were unset.
- If you want to submit a job to run on a specific type of machine (for example, one with **Arch = i386** and **OpSys= Linux2**), you must specify a requirements statement that includes the **Arch** and **OpSys** requirements.
- In a multicluster environment, job identifiers are assigned by the local cluster and are retained by the job regardless of what cluster the job runs in.

If the job was submitted as a remote job in a multicluster environment, the host represented in `host.jobid.stepid`, is the name of the local Schedd machine that assigned the `jobid`. To determine the managing Schedd machine, issue the `llq -l` command to obtain the Schedd Host field.

If the administrator has not defined a **CLUSTER_METRIC** for the local cluster, the **lsubmit** command will display an error and fail to submit the job if the user specifies the **-X** flag with a `cluster_list` or the reserved word **any**. The **lsubmit** command will also display an error and fail to submit the job if the user specifies the **-X cluster_name** in the following instances:

- The local cluster is not in the multicluster environment
- The specified cluster name is not configured
- The specified cluster name does not have `inbound_hosts` specified

Examples

1. This example shows a job command file named `qtrlyrun.cmd` is submitted:

```
lsubmit qtrlyrun.cmd
```
2. This example shows a job being submitted to a remote cluster:

```
lsubmit -X cluster1 jcf.cmd
```

Results

1. The following shows the results of the **lsubmit qtrlyrun.cmd** command issued from the machine **earth**:

```
lsubmit: The job "earth.505" has been submitted.
```

Note that 505 is the job ID generated by LoadLeveler.
2. The following shows the results from a remote submit:

```
Job c188f2n08.ppd.pok.ibm.com.21 assigned to local outbound \
    Schedd c188f2n08.ppd.pok.ibm.com.
Job c188f2n08.ppd.pok.ibm.com.21 assigned to remote inbound \
    Schedd c188f2n02.ppd.pok.ibm.com.
Job c188f2n08.ppd.pok.ibm.com.21 has been submitted to cluster "cluster1"
lsubmit: The job "c188f2n08.ppd.pok.ibm.com.21" has been submitted.
```

Related Information

Subroutines: **lsubmit**

Security

LoadLeveler administrators and users can issue this command.

llsummary - Return job resource information for accounting

Purpose

llsummary – Returns job resource information on completed jobs for accounting purposes.

You must enable the recording of accounting data in order to generate any of the four throughput reports. To do this, specify **ACCT=A_ON** in your **LoadL_config** file. For detailed usage of the **ACCT** keyword, see “Gathering job accounting data” on page 57.

Syntax

```
llsummary [-?] [-H] [-v] [-x] [-l] [-s MM/DD/YYYY to MM/DD/YYYY]
[-e MM/DD/YYYY to MM/DD/YYYY] [-g group]
[-G unixgroup] [-a allocated] [-r report] [-j jobname]
[-d section] [-c class] [-u user] [filelist]
```

Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
- x Provides extended information. Using -x can produce a very long report. This option is meaningful only when used with the -l option. You must enable the recording of accounting data in order to collect information with the -x flag. To do this, specify **ACCT=A_ON A_DETAIL** in your **LoadL_config** file.
- l Specifies that the long form of output is displayed.
- s Specifies a range for the start date (queue date) for accounting data to be included in this report. The format for entering the date is either *MM/DD/YYYY* (where *MM* is month, *DD* is day, and *YYYY* is year), *MM/DD/YY* (where *YY* is a two-digit year value), or a string of digits representing the number of seconds since 1970. If a two-digit year value is used, then 69-99 maps to 1969-1999, and 00-68 maps to 2000-2068. The default is to include all the data in the report.
- e Specifies a range for the end date (completion date) for accounting data to be included in this report. The format for entering the date is either *MM/DD/YYYY* (where *MM* is month, *DD* is day, and *YYYY* is year), *MM/DD/YY* (where *YY* is a two-digit year value), or a string of digits representing the number of seconds since 1970. The default is to include all the data in the report.
- u *user*
Specifies the user ID for whom accounting data is reported.
- c *class*
Specifies the class for which accounting data is reported. For reports of all formats (short, long and extended), **llsummary** will report information about every job which contains at least one step of the specified class. For the short format, **llsummary** also reports a job count and step count for each class; for these counts, a job’s class is determined by the class of its first step.

-g *group*

Specifies the LoadLeveler group for which accounting data is reported. For reports of all formats (short, long and extended), **llsummary** reports information about every job which contains at least one step of the specified group. For the short format, **llsummary** also reports a job count and step count for each group; for these counts, a job's group is determined by the group of its first step.

-G *unixgroup*

Specifies the UNIX group for which accounting data is reported.

-a *allocated*

Specifies the hostname that was allocated to run the job. You can specify the allocated host in short or long form.

-r *report*

Specifies the report type. You must enable the recording of accounting data in order to collect information with the **-r** flag. To do this, specify **ACCT=A_ON A_DETAIL** in your **LoadL_config** file. You can choose one or more of the following reports:

avgthroughput

Provides average queue time, run time, and CPU time for jobs that ran for at least some period of time.

maxthroughput

Provides maximum queue time, run time, and CPU time for jobs that ran for at least some period of time.

minthroughput

Provides minimum queue time, run time, and CPU time for jobs that ran for at least some period of time.

numeric

Reports CPU times in seconds rather than hours, minutes, and seconds.

resource

Provides CPU usage for all submitted jobs, including those that did not run. This is the default.

throughput

Selects all throughput reports.

-d *section*

Specifies the category (data section) for which you want to generate a report. You can specify one or more of the following: **user**, **group**, **unixgroup**, **class**, **account**, **day**, **week**, **month**, **jobid**, **jobname**, **allocated**.

-j *host.jobid*

Specifies the job for which accounting data is reported.

The format of a full LoadLeveler job identifier is *host.jobid*.

where:

- *host* is the name of the machine that assigned the job identifiers.
- *jobid* is the job number assigned to the job when it was submitted.

The entire *host.jobid* string is required.

filelist

Is a blank-delimited list of files containing the accounting data. If not specified, the default is the local history file on the machine from which the command

llsummary

was issued. You can use the **llacctmrg** command to combine history files on different Schedd machines into a single history file.

Description

In order to create an accounting report with the **llsummary** command, you must have read access to a history file. If a history file name is not specified as an argument, **llsummary** uses the history file in the LoadLeveler spool directory of the local machine as input. By default, the permissions of the spool directory are set by the **llinit** command to 700 at install time. However, these permissions may be changed by a system administrators with root privileges.

The file permissions of the history file created by a **LoadL_schedd** daemon are controlled by the **HISTORY_PERMISSION** configuration keyword. A specification such as **HISTORY_PERMISSION = rw-rw-r--** will result in permission settings of 664. The default settings are 660.

Examples

1. The following example requests summary reports (standard listing) of all the jobs submitted on your machine between the days of January 12, 2006 and March 12, 2006:

```
llsummary -s 01/12/2006 to 03/12/2006
```

2. This example generates the standard listing when you do not specify **-l**, **-r**, or **-d** with **llsummary**.

This sample report includes summaries of the following data:

- Number of jobs, Total CPU usage, per user.
- Number of jobs, Total CPU usage, per class.
- Number of jobs, Total CPU usage, per group.
- Number of jobs, Total CPU usage, per account number.

```
llsummary
```

You should receive output similar to the following:

Name	Jobs	Steps	Job Cpu	Starter Cpu	Leverage
krystal	15	36	0+00:09:50	0+00:00:10	59.0
lixin3	18	54	0+00:08:28	0+00:00:16	31.8
TOTAL	33	90	0+00:18:18	0+00:00:27	40.7

Class	Jobs	Steps	Job Cpu	Starter Cpu	Leverage
small	9	21	0+00:01:03	0+00:00:06	10.5
large	12	36	0+00:13:45	0+00:00:11	75.0
os12	3	9	0+00:00:27	0+00:00:02	13.5
No_Class	9	24	0+00:03:01	0+00:00:06	30.2
TOTAL	33	90	0+00:18:18	0+00:00:27	40.7

Group	Jobs	Steps	Job Cpu	Starter Cpu	Leverage
No_Group	12	30	0+00:09:32	0+00:00:09	63.6
chemistry	7	18	0+00:04:50	0+00:00:05	58.0
engineering	14	42	0+00:03:56	0+00:00:12	19.7
TOTAL	33	90	0+00:18:18	0+00:00:27	40.7

Account	Jobs	Steps	Job Cpu	Starter Cpu	Leverage
33333	16	39	0+00:05:54	0+00:00:11	32.2

22222	15	45	0+00:12:05	0+00:00:13	55.8
99999	2	6	0+00:00:18	0+00:00:01	18.0
TOTAL	33	90	0+00:18:18	0+00:00:27	40.7

The **standard listing** includes the following fields:

Account

Account number specified for the jobs.

Class Class specified or defaulted for the jobs.

Group User's login group.

Job CPU

Total CPU time consumed by user's jobs.

Jobs Count of the total number of jobs submitted by this user, class, group, or account.

Leverage

Ratio of job CPU to starter CPU.

Name User ID submitting jobs.

Starter CPU

Total CPU time consumed by LoadLeveler starter processes on behalf of the user jobs.

Steps Count of the total number of job steps submitted by this user, class, group, or account.

- To generate a throughput report, issue:

```
llsummary -r throughput
```

You should receive output similar to the following. Note that only the user output is shown, the class, group, and account lines are not shown.

Name	Jobs	Steps	AvgQueueTime	AvgRealTime	AvgCPUTime
load1	1	4	0+00:00:03	0+00:05:27	0+00:05:17
user1	2	6	0+00:03:05	0+00:03:45	0+00:03:04
ALL	3	10	0+00:01:52	0+00:04:26	0+00:03:58

Name	Jobs	Steps	MinQueueTime	MinRealTime	MinCPUTime
load1	1	4	0+00:00:01	0+00:02:49	0+00:02:44
user1	2	6	0+00:02:02	0+00:03:43	0+00:03:02
ALL	3	10	0+00:00:01	0+00:02:49	0+00:02:44

Name	Jobs	Steps	MaxQueueTime	MaxRealTime	MaxCPUTime
load1	1	4	0+00:00:06	0+00:12:58	0+00:12:37
user1	2	6	0+00:06:21	0+00:03:48	0+00:03:07
ALL	3	10	0+00:06:21	0+00:12:58	0+00:12:37

The **-r** listing includes the following fields:

AvgCPUTime

Average amount of accumulated CPU time for jobs associated with this user, class, group, or account.

AvgQueueTime

Average amount of time the job spent queued before running for this user, class, group, or account.

llsummary

AvgRealTime

Average amount of accumulated wall clock time for jobs associated with this user, class, group, or account.

MaxCPUTime

Time of the job with the greatest amount of CPU time for this user, class, group, or account.

MaxQueueTime

Time of the job that spent the greatest amount of time in queue before running for this user, class, group, or account.

MaxRealTime

Time of the job with the greatest amount of wall clock time for this user, class, group, or account.

MinCPUTime

Time of the job with the least amount of CPU time for this user, class, group, or account.

MinQueueTime

Time of the job that spent the least amount of time in queue before running for this user, class, group, or account.

MinRealTime

Time of the job with the least amount of wall clock time for this user, class, group, or account.

The ALL line for the Average listing displays the average time for all users, classes, groups, and accounts. The ALL line for the Minimum listing displays the time of the job with the least amount of time for all users, classes, groups, and accounts. The ALL line for the Maximum listing displays the time of the job with the greatest amount of time for all users, classes, groups, and accounts.

4. The following example generates the long listing that contains Blue Gene-specific information:

```
llsummary -l -j job_name
```

You should receive output similar to the following:

```
.  
.  
.  
      Step Type: bluegene  
      Min Processors: 1  
      Max Processors: 1  
      Size Requested: 2048  
      Size Allocated: 2048  
      Shape Requested:  
      Shape Allocated: 32x8x8  
      Wiring Requested: PREFER_TORUS  
      Wiring Allocated: TORUS  
      Rotate: TRUE  
      Partition Requested:  
      Partition Allocated: part130  
      BG Requirements: (Memory == 1024)  
      Error Text:  
      Alloc. Host Count: 1  
.  
.  
.
```

5. The following example generates the long listing that contains multicluster-specific information submitted or moved to a remote cluster:

llsummary -l

You should receive output similar to the following:

```

.
.
.
Scheduling Cluster: cluster1
Submitting Cluster: cluster2
  Sending Cluster: cluster2
  Submitting User: brownap
  Schedd History: c188f2n08.ppd.pok.ibm.com
  Outbound Schedds: c188f2n08.ppd.pok.ibm.com
.
.
.
Cluster input file: /u/brownap/copyfile_input, /tmp/copyfile_input1
Cluster input file: /u/brownap/copyfile_input, /tmp/copyfile_input2
Cluster input file: /u/brownap/copyfile_input, /tmp/copyfile_input3
Cluster output file: /tmp/copyfile_output, /u/brownap/copyfile_output
.
.
.

```

For an explanation of the fields in the long listing, see “llq - Query job status” on page 449. See Appendix B, “Sample command output,” on page 673 for sample output of long listings.

6. For InfiniBand support, the following example includes the port number for the allocated hosts and task instances:

llsummary -l -x

You should receive output similar to the following:

```

.
.
.
Task
----

  Num Task Inst: 4
  Task Instance: c171f6sq07:0:ib0(MPI,IP,-1,Shared,0 rCxt Blks,1), \
    ib1(MPI,IP,-1,Shared,0 rCxt Blks,2)
  Task Instance: c171f6sq07:1:ib0(MPI,IP,-1,Shared,0 rCxt Blks,1), \
    ib1(MPI,IP,-1,Shared,0 rCxt Blks,2)
  Task Instance: c171f6sq08:2:ib0(MPI,IP,-1,Shared,0 rCxt Blks,1), \
    ib1(MPI,IP,-1,Shared,0 rCxt Blks,2)
  Task Instance: c171f6sq08:3:ib0(MPI,IP,-1,Shared,0 rCxt Blks,1), \
    ib1(MPI,IP,-1,Shared,0 rCxt Blks,2)
.
.
.

```

Security

LoadLeveler administrators and users can issue this command.

IIsummary

Chapter 17. Application programming interfaces (APIs)

LoadLeveler provides several application programming interfaces (APIs). These APIs allow application programs written by customers to interact with the LoadLeveler environment using specific data and functions that are a part of LoadLeveler. These interfaces can be subroutines within a library or installation exits.

The header file **llapi.h** defines all of the API data structures and subroutines. This file is located in the **include** subdirectory of the LoadLeveler release directory. You must include this file when you call any API subroutine.

The library **libllapi.a** (AIX) or **libllapi.so** (Linux) is a shared library containing all of the LoadLeveler API subroutines. This library is located in the **lib** subdirectory of the LoadLeveler release directory.

Attention: These APIs are not *thread safe*; they should not be linked to by a threaded application.

Table 86 lists all of the LoadLeveler APIs, along with the intended users and supported operating systems for each, and a reference to the full descriptions of each interface.

Table 86. LoadLeveler API summary

Task / API category	Interface name	Intended users	Supported operating systems
Generate accounting reports "Accounting API" on page 506	GetHistory subroutine	Both administrators and general users	AIX and Linux
	llacctval user exit	Administrators only	AIX and Linux
Checkpoint LoadLeveler jobs "Checkpointing API" on page 510	ckpt subroutine	Provided for backward compatibility only. Use ll_init_ckpt and ll_ckpt instead.	AIX only
	ll_ckpt subroutine	Both administrators and general users	AIX and Linux ¹
	ll_init_ckpt subroutine	Both administrators and general users	AIX only
	ll_set_ckpt_callbacks subroutine	Both administrators and general users	AIX only
	ll_unset_ckpt_callbacks subroutine	Both administrators and general users	AIX only
Process configuration files "Configuration API" on page 519	ll_config_changed subroutine	Both administrators and general users	AIX and Linux
	ll_read_config subroutine	Both administrators and general users	AIX and Linux

Summary of LoadLeveler APIs

Table 86. LoadLeveler API summary (continued)

Task / API category	Interface name	Intended users	Supported operating systems
Access LoadLeveler objects and retrieve data from objects "Data access API" on page 522	ll_deallocate subroutine	Both administrators and general users	AIX and Linux
	ll_free_objs subroutine	Both administrators and general users	AIX and Linux
	ll_get_data subroutine	Both administrators and general users	AIX and Linux
	ll_get_objs subroutine	Both administrators and general users	AIX and Linux
	ll_next_obj subroutine	Both administrators and general users	AIX and Linux
	ll_query subroutine	Both administrators and general users	AIX and Linux
	ll_reset_request subroutine	Both administrators and general users	AIX and Linux
	ll_set_request subroutine	Both administrators and general users	AIX and Linux
Convert an error object into an error message "Error handling API" on page 590	ll_error subroutine	Both administrators and general users	AIX and Linux
Fair share scheduling API "Fair share scheduling API" on page 592	ll_fair_share subroutine	Administrators only	AIX and Linux
Query APIs "Query API" on page 595	ll_free_jobs subroutine	Both administrators and general users	AIX only
	ll_free_nodes subroutine	Both administrators and general users	AIX only
	ll_get_jobs subroutine	Both administrators and general users	AIX only
	ll_get_nodes subroutine	Both administrators and general users	AIX only
Reservation APIs "Reservation API" on page 600	ll_bind subroutine	Both administrators and general users	AIX and Linux
	ll_change_reservation subroutine	Both administrators and general users	AIX and Linux
	ll_init_reservation_param subroutine	Both administrators and general users	AIX and Linux
	ll_make_reservation subroutine	Both administrators and general users	AIX and Linux
	ll_remove_reservation subroutine	Both administrators and general users	AIX and Linux
Submit jobs to LoadLeveler "Submit API" on page 614	llfree_job_info subroutine	Both administrators and general users	AIX and Linux
	llsubmit subroutine	Both administrators and general users	AIX and Linux
	monitor_program user exit	Both administrators and general users	AIX and Linux

Table 86. LoadLeveler API summary (continued)

Task / API category	Interface name	Intended users	Supported operating systems
Perform LoadLeveler control operations and work with an external scheduler “Workload management API” on page 619	ll_cluster subroutine	Both administrators and general users	AIX and Linux
	ll_cluster_auth subroutine	Administrators only	AIX and Linux
	ll_control subroutine	Both administrators and general users may specify the following control operations: <ul style="list-style-type: none"> • LL_CONTROL_HOLD_USER • LL_CONTROL_PRI0_ABS • LL_CONTROL_PRI0_ADJ • LL_CONTROL_START • LL_CONTROL_START_DRAINED All other control operations defined in the <code>llapi.h</code> header file are intended for use by administrators only.	AIX and Linux
	ll_modify subroutine	Both administrators and general users	AIX and Linux
	ll_move_job subroutine	Administrators only	AIX and Linux
	ll_move_spool subroutine	Administrators only	AIX and Linux
	ll_preempt subroutine	Administrators only	AIX only
	ll_preempt_jobs subroutine	Administrators only	AIX and Linux
	ll_run_scheduler subroutine	Administrators only	AIX and Linux
	ll_start_job subroutine	Administrators only	AIX and Linux
ll_start_job_ext subroutine	Administrators only	AIX and Linux	
ll_terminate_job subroutine	Administrators only	AIX and Linux	

¹ This subroutine will run on LoadLeveler for Linux platforms, but it can only checkpoint jobs on AIX.

64-bit support for the LoadLeveler APIs

LoadLeveler for AIX APIs support both 32-bit and 64-bit applications. LoadLeveler for Linux APIs support 32-bit applications on 32-bit platforms and 64-bit applications on 64-bit platforms.

AIX APIs

In LoadLeveler 3.2 or later releases, the LoadLeveler API library (`libllapi.a`) consists of two sets of objects: 32-bit and 64-bit. Both sets of objects and interfaces are provided since the AIX linker cannot create an executable from a mixture of 32-bit and 64-bit objects. They must be all of the same type. Developers attempting to exploit the 64-bit capabilities of the LoadLeveler API library should take into consideration that all interfaces of the LoadLeveler API library are available in both 32-bit and 64-bit formats. Interfaces with the same names are functionally equivalent.

Summary of LoadLeveler APIs

Linux APIs

On Linux, the LoadLeveler 3.4 API library (**libllapi.so**) is a 32-bit library on 32-bit platforms and a 64-bit library on 64-bit platforms. The library **libllapi.so** is a 32-bit library on the following platforms:

- RHEL 3 and RHEL 4 on IBM IA-32 xSeries servers
- SLES 9 and SLES 10 on IBM IA-32 xSeries servers

libllapi.so is a 64-bit library on the following platforms:

- RHEL 3 and RHEL 4 on IBM eServers with AMD Opteron or Intel EM64T processors
- RHEL 4 on IBM POWER servers
- SLES 9 and SLES 10 on IBM eServers with AMD Opteron or Intel EM64T processors
- SLES 9 and SLES 10 on IBM POWER servers

For users running Parallel Operating Environment, there are 32-bit LoadLeveler library RPMs available for installing on the following 64-bit platforms:

- RHEL 4 on IBM eServers with AMD Opteron or Intel EM64T processors
- RHEL 4 on IBM POWER servers
- SLES 9 and SLES 10 on IBM eServers with AMD Opteron or Intel EM64T processors
- SLES 9 and SLES 10 on IBM POWER servers

Note: RHEL 3 does not support a 32-bit library.

Accounting API

This API provides a subroutine for extracting accounting data and a user exit for account validation. Job accounting information saved in a history file can also be queried by using the data access API.

GetHistory subroutine

Purpose

GetHistory – Processes local or global LoadLeveler history files.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int GetHistory(char *filename, int (*func) (LL_job *), int version);
```

Parameters

filename

Specifies the name of the history file.

*(*func) (LL_job *)*

Specifies the user-supplied function you want to call to process each history record. The function must return an integer and must accept as input a pointer to the LL_job structure. The LL_job structure is defined in the **llapi.h** file.

version

Specifies the version of the history record you want to create.

LL_JOB_VERSION in the **llapi.h** file creates an LL_job history record.

Description

GetHistory opens the history file you specify, reads one LL_job accounting record, and calls a user-supplied routine, passing to the routine the address of an LL_job structure. **GetHistory** processes all history records one at a time and then closes the file. Any user can call this subroutine.

The user-supplied function must include the following files:

```
#include <sys/resource.h>
#include <sys/types.h>
#include <sys/time.h>
```

The ll_event_usage structure is part of the LL_job structure and contains the following LoadLeveler defined data:

int *event*

Specifies the event identifier. This is an integer whose value is one of the following:

- 1 Represents a LoadLeveler-generated event.
- 2 Represents an installation-generated event.

char **name*

Specifies a character string identifying the event. This can be one of the following:

- An installation generated string that uses the command **llctl capture eventname**.
- LoadLeveler-generated strings, which can be the following:
 - started
 - checkpoint
 - vacated
 - completed
 - rejected
 - removed

GetHistory subroutine

Return Values

GetHistory returns a zero when successful.

Error Values

GetHistory returns -1 to indicate that the version is not supported or that an error occurred opening the history file.

Examples

Makefiles and examples which use this API are located in the **samples/llphist** subdirectory of the release directory. The examples include the executable **llpjob**, which invokes **GetHistory** to print every record in the history file. In order to compile **llpjob**, the sample Makefile must update the **RELEASE_DIR** field to represent the current LoadLeveler release directory. The syntax for **llpjob** is:

```
llpjob history_file
```

Where *history_file* is a local or global history file.

llacctval user exit

Purpose

LoadLeveler provides the **llacctval** executable to perform account validation.

llacctval – Compares the account number a user specifies in a job command file with the account numbers defined for that user in the LoadLeveler administration file. If the account numbers match, **llacctval** returns a value of zero. Otherwise, it returns a nonzero value.

Syntax

```
program user_name user_group user_acct# acct1 acct2 ...
```

Parameters

program

Is the name of the program that performs the account validation. The default is **llacctval**. The name you specify here must match the value specified on the **ACCT_VALIDATION** keyword in the configuration file.

user_name

Is the name of the user whose account number you want to validate.

user_group

Is the login group name of the user.

user_acct#

Is the account number specified by the user in the job command file.

acct1 acct2 ...

Are the account numbers obtained from the user stanza in the LoadLeveler administration file.

Description

llacctval is invoked from within the **llsubmit** command. If the return code is nonzero, **llsubmit** does not submit the job.

You can replace **llacctval** with your own accounting user exit.

To enable account validation, you must specify the following keyword in the configuration file:

```
ACCT = A_VALIDATE
```

To use your own accounting exit, specify the following keyword in the configuration file:

```
ACCT_VALIDATION = pathname
```

where *pathname* is the name of your accounting exit.

Return Values

If the validation succeeds, the exit status must be zero. If it does not succeed, the exit status must be a nonzero number.

Checkpointing API

This API is used to checkpoint jobs running under LoadLeveler and consists of the following subroutines:

- **ckpt subroutine**
- **ll_ckpt subroutine**
- **ll_init_ckpt subroutine**
- **ll_set_ckpt_callbacks subroutine**
- **ll_unset_ckpt_callbacks subroutine**

|
| The **ll_ckpt** subroutine will run on LoadLeveler for Linux platforms, but it can
| only checkpoint jobs on AIX. All other checkpoint API functions are not supported
| on Linux.

For more information, see “LoadLeveler support for checkpointing jobs” on page 129. For information on checkpointing parallel jobs, see *IBM Parallel Environment for AIX: Operation and Use, Volume 1*.

ckpt subroutine

Purpose

Specify the **ckpt** subroutine in a FORTRAN, C, or C++ program to activate checkpointing from within the application. Whenever this subroutine is invoked, a checkpoint of the program is taken.

Note: This API is obsolete and is supported for backward compatibility only. It calls **ll_init_ckpt**.

C++ syntax

```
extern "C"{void ckpt();}
```

C syntax

```
void ckpt();
```

FORTRAN syntax

```
call ckpt()
```

ll_ckpt subroutine

Purpose

ll_ckpt – Initiates a checkpoint on a specific job step.

Library

LoadLeveler API library **libllapi.a** or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int ll_ckpt (LL_ckpt_info *ckpt_info);
```

Parameters

ckpt_info

A pointer to a **LL_ckpt_info** structure, which has the following fields:

int *version*

The version of the API that the program was compiled with (from llapi.h).

char* *step_id*

A job or step identifier. When a job identifier is specified, the action of the API is taken for all steps of the job. At least one job or step identifier must be specified.

The format of a job identifier is *host.jobid*. The format of a step identifier is *host.jobid.stepid*.

where:

- *host* is the name of the machine that assigned the job and step identifiers.
- *jobid* is the job number assigned to the job when it was submitted.
- *stepid* is the job step number assigned to the job step when it was submitted.

The job or step identifier may be specified in an abbreviated form, *jobid* or *jobid.stepid*, when the API is invoked on the same machine that assigned the job and step identifiers. In this case, LoadLeveler will use the local machine's host name to construct the full job or step identifier.

enum *ckpt_type* *ckptType*

The action to be taken after the checkpoint successfully completes. The values for **enum** *ckpt_type* are:

CKPT_AND_CONTINUE

Allows the job to continue after the checkpoint.

CKPT_AND_TERMINATE

Terminates the job after the checkpoint.

CKPT_AND_HOLD

Puts the job on user hold after the checkpoint. A coschedule job step can be specified with the **CKPT_AND_HOLD** option.

Note: If checkpoint is not successful, the job continues on return regardless of these settings.

enum wait_option *waitType*

The flag used to identify blocking action during checkpoint. By default, ll_ckpt() will block until the checkpoint completes. The values for the enum wait_option are:

CKPT_NO_WAIT

Disables blocking while the job is being checkpointed.

CKPT_WAIT

The job is blocked while being checkpointed. This is the default.

int abort_sig

Identifies the signal to be used to interrupt a checkpoint initiated by the API. Upon receipt of this signal the checkpoint will be aborted. Default is SIGINT.

cr_error_t *cp_error_data

The structure containing error information from the checkpoint operation.

int ckpt_rc

The return code from checkpoint.

int soft_limit

The time, in seconds, indicating the maximum time allocated for a checkpoint operation to complete before the checkpoint operation is aborted. The job is allowed to continue. The value for soft_limit specified here will override any soft limit value specified in the job command file. If the value for soft limit specified by the administration file is less than the value specified here, the administration file value takes precedence.

Values are:

-1 Indicates there is no limit.

0 Indicates the existing soft limit for the job step should be enforced.

Positive integer

Indicates the number of seconds allocated for the limit.

int hard_limit

The time, in seconds, indicating the maximum time allocated for a checkpoint operation to complete before the job is terminated. The value for hard-limit specified here will override any hard limit value specified in the job command file. If the value for hard limit specified by the administration file is less than the value specified here, the administration file value will take precedence.

Values are:

-1 Indicates there is no limit.

0 Indicates the existing hard limit for the job step should be enforced.

Positive integer

Indicates the number of seconds allocated for the limit.

Description

This function initiates a checkpoint for the specified job step. ll_ckpt() will, by default, block until the checkpoint operation completes. To disable blocking, the flag *waitType* must be set to **NO_WAIT**. This function is allowed to be executed by the owner of the job step or a LoadLeveler administrator.

ll_ckpt subroutine

Return Values

- 0 Checkpoint completed successfully.
- 1 Checkpoint event did not receive status and the success or failure of the checkpoint is unclear.

Error Values

- 1 Error occurred attempting to checkpoint.
- 2 Format not valid for job step, not in the form *host.jobid.stepid*.
- 3 Cannot allocate memory.
- 4 API cannot create listen socket.
- 6 Configuration file errors.

ll_init_ckpt subroutine

Purpose

ll_init_ckpt – Initiates a checkpoint from within a serial application.

Library

LoadLeveler API library **libllapi.a**.

Syntax

```
#include "llapi.h"
```

```
int ll_init_ckpt (LL_ckpt_info *ckpt_info);
```

Parameters

ckpt_info

A pointer to a LL_ckpt_info structure, which has the following fields:

int *version*

The version of the API that the program was compiled with (from llapi.h).

char* *step_id*

NULL, not used.

enum ckpt_type *ckptType*

NULL, not used.

enum wait_option *waitType*

NULL, not used.

int *abort_sig*

NULL, not used.

cr_error_t **cp_error_data*

AIX structure containing error info from **ll_init_ckpt**. When the return code indicates the checkpoint was attempted but failed (-7), detailed information is returned in this structure.

int *ckpt_rc*

Return code from checkpoint.

int *soft_limit*

This field is ignored.

int *hard_limit*

This field is ignored.

Description

This subroutine is only available if you have enabled checkpointing. **ll_init_ckpt** initiates a checkpoint from within a serial application. The checkpoint file name will consist of a base name with a suffix of a numeric checkpoint tag to differentiate from an earlier checkpoint file. LoadLeveler sets the **LOADL_CKPT_FILE** environment variable which identifies the directory and file name for checkpoint files.

Return Values

- 0 The checkpoint completed successfully.
- 1 Indicates **ll_init_ckpt()** returned as a result of a restart operation.

Error Values

- 1 Cannot retrieve the job step ID from the LOADL_STEP_ID environment variable.
- 2 Cannot retrieve the checkpoint file name from the LOADL_CKPT_FILE environment variable, checkpoint has not been enabled for the job step (checkpoint not set to yes or interval).
- 3 Cannot allocate memory.
- 4 Checkpoint/restart ID is not valid, checkpointing is not enabled for the job step.
- 5 Request to take checkpoint denied by starter.
- 6 Request to take checkpoint failed, no response from starter, possible communication problem.
- 7 Checkpoint attempted but failed. Details of error can be found in the **LL_ckpt_info** structure.
- 8 Cannot install SIGINT signal handler.

ll_set_ckpt_callbacks subroutine

Purpose

ll_set_ckpt_callbacks – Registers callbacks, which will be invoked, when a job step is checkpointed, resumed, and restarted.

Library

LoadLeveler API library **libllapi.a**

Syntax

```
#include "llapi.h"
```

```
int ll_set_ckpt_callbacks (callbacks_t *cbs);
```

Parameters

cbs

A pointer to a **callbacks_t** structure, which is defined as:

```
typedef struct {
    void (*checkpoint_callback) (void) ;
    void (*restart_callback) (void) ;
    void (*resume_callback) (void) ;
} callbacks_t;
```

Where:

checkpoint_callback

Pointer to the function to be invoked at checkpoint time.

restart_callback

Pointer to the function to be invoked at restart time.

resume_callback

Pointer to the function to be called when an application is resumed after taking a checkpoint.

Description

This function is called to register functions to be invoked when a job step is checkpointed, resumed, and restarted.

Return Values

If successful, a nonnegative integer is returned which is a handle used to identify the particular set of callback functions. The handle can be used as input to the **ll_unset_ckpt_callbacks** function. If an error occurs, a negative number is returned.

Error Values

- 1 Process is not enabled for checkpointing.
- 2 Unable to allocate storage to store callback structure.
- 3 Cannot allocate memory.

ll_unset_ckpt_callbacks subroutine

Purpose

ll_unset_ckpt_callbacks – Unregisters previously registered checkpoint, resume, and restart callbacks.

Library

LoadLeveler API library **libllapi.a**

Syntax

```
#include "llapi.h"
```

```
int ll_unset_ckpt_callbacks(int handle);
```

Parameters

handle

An integer indicating the set of callback functions to be unregistered. This integer is the value returned by the **ll_set_ckpt_callbacks** function which was used to register the callbacks.

Description

This API is called to unregister checkpoint, resume, and restart application callback functions which were previously registered with the **ll_set_ckpt_callbacks** function.

Return Values

0 Success.

Error Values

-1 Unable to unregister callback. Argument not valid, specified handle does not reference a valid callback structure.

Configuration API

This API allows operations on LoadLeveler configuration files and consists of the following subroutines:

- **ll_config_changed** subroutine
- **ll_read_config** subroutine

ll_config_changed subroutine

ll_config_changed subroutine

Purpose

ll_config_changed – Allows users to determine if the LoadLeveler configuration files were changed since the last time they were read.

Library

LoadLeveler API library **libllapi.a**

Syntax

```
#include "llapi.h"  
int ll_config_changed(void);
```

Description

This subroutine is used in conjunction with the **ll_read_config** subroutine to reread the configuration files.

Return Values

An integer value will be returned which indicates whether the configuration files were changed since the last time that they were read.

0 The configuration files were not changed.

1 At least one of the configuration files was changed.

Related Information

Subroutine: **ll_read_config**

ll_read_config subroutine

Purpose

ll_read_config – Allows users to read or reread the LoadLeveler configuration files.

Library

LoadLeveler API library **libllapi.a**

Syntax

```
#include "llapi.h"
int ll_read_config(LL_element **errObj);
```

Parameters

errObj

This is the address of a pointer to a LoadLeveler error object. The pointer to the **LL_element** should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed through the **ll_error** function. The caller should free the error object storage before reusing the pointer.

Description

This subroutine is used in conjunction with the **ll_config_changed** subroutine to determine whether the configuration files were changed since the last time they were read.

Return Values

API_OK

The configuration files was successfully read.

API_CONFIG_ERR

Errors were encountered while reading the configuration files.

Related Information

Subroutines: **ll_config_changed**, **ll_error**

Data access API

This API gives you access to LoadLeveler data objects and allows you to retrieve specific data from those objects. You can use this API to query LoadLeveler daemons for information about:

- BLUE_GENE (Blue Gene system)
- CLASSES (job classes)
- CLUSTER (cluster information)
- FAIRSHARE (fair share scheduling information)
- JOBS (job information)
- MACHINES (machine information)
- MCLUSTERS (multicluster objects)
- RESERVATIONS (reservation information)
- WLMSTAT (AIX Workload Manager)

This API can also be used to query a LoadLeveler history file for accounting information about JOBS.

The data access API consists of the following subroutines:

- **ll_deallocate** subroutine
- **ll_free_objs** subroutine
- **ll_get_data** subroutine
- **ll_get_objs** subroutine
- **ll_next_obj** subroutine
- **ll_query** subroutine
- **ll_reset_request** subroutine
- **ll_set_request** subroutine

Using the data access API

The data access API makes use of a query object to facilitate each request for LoadLeveler data. The query object is used to specify the details of each query request and to manage the list of LoadLeveler objects retrieved. It is initialized by the **ll_query** subroutine to specify the kind of LoadLeveler objects requested and is modified by the **ll_set_request** and **ll_reset_request** subroutines to specify filtering of those objects. It is then passed to the **ll_get_objs** and **ll_next_obj** subroutines to use in traversing the list of objects. To use this API, you need to call the subroutines in the following order:

- Call **ll_query** to initialize the query object and define the type of objects you want to query. See “ll_query subroutine” on page 578 for more information.
- Call **ll_set_request** to filter the objects you want to query. See “ll_set_request subroutine” on page 580 for more information.
- Call **ll_reset_request** to reset the filter on the objects you want to query. See “ll_reset_request subroutine” on page 579 for more information.
- Call **ll_get_objs** to retrieve a list of objects from a LoadLeveler daemon or history file. See “ll_get_objs subroutine” on page 574 for more information.
- Call **ll_get_data** to retrieve specific data from an object. See “ll_get_data subroutine” on page 532 for more information.
- Call **ll_next_obj** to retrieve the next object in the list. See “ll_next_obj subroutine” on page 577 for more information.
- Call **ll_free_objs** to free the list of objects you received. See “ll_free_objs subroutine” on page 531 for more information.
- Call **ll_deallocate** to end the query. See “ll_deallocate subroutine” on page 530 for more information.

In a multicluster configuration, you need to use the “ll_cluster subroutine” on page 620 to access data from a remote cluster. The **ll_cluster** subroutine is used to define the cluster prior to any data access API calls.

To see code that uses these subroutines, refer to “Examples of using the data access API” on page 584. For more information on LoadLeveler objects, see “Understanding the LoadLeveler data access object model.”

Understanding the LoadLeveler data access object model

The **ll_get_data** subroutine of the data access API allows you to access the LoadLeveler object model. A LoadLeveler object model consists of a root object with zero or more associated objects that have attributes and associations to other objects. An attribute is a characteristic of the object and generally has a primitive data type (such as integer, float, or character). Attributes and related objects are retrieved from an object using the “ll_get_data subroutine” on page 532. One of the arguments to the **ll_get_data** subroutine is the specification. Specifications are used to identify which attribute or related object to retrieve. The job name, submission time, and job priority are examples of attributes.

Objects are associated with one or more other objects through relationships. An object can be associated with other objects through more than one relationship, or through the same relationship. For example, A Job object is associated with a Credential object and to Step objects through two different relationships. A Job object can be associated with more than one Step object through the same relationship of “having a Step.” When an object is associated through different relationships, different specifications are used to retrieve the appropriate object.

When an object is associated with more than one object through the same relationship, there are GetFirst and GetNext specifications associated with the relationship. You must use the GetFirst specification to initialize access to the first such associated object. You must use the GetNext specification to get the remaining objects in succession. After the last object has been retrieved, GetNext will return NULL.

The specification tables for the **ll_get_data** subroutine in the data access API in *TWS LoadLeveler: Using and Administering* describe the specifications and elements available when you use the **ll_get_data** subroutine. Each specification name describes the object you need to specify and the attribute returned. For example, the specification **LL_JobGetFirstStep** includes the object you need to specify (**Job**) and the value returned (**LL_element* Step**). The tables are sorted alphabetically by object.

You can use the **ll_get_data** subroutine to access both attributes and associated objects. See the “ll_get_data subroutine” on page 532 for more information.

It is important to keep in mind how the list of objects returned by the **ll_get_objs** subroutine have been defined by the previous data access API calls when getting specific pieces of data using the **ll_get_data** subroutine.

Several factors can affect what and when data attributes are available or relevant:

- Data related to certain LoadLeveler features (such as Blue Gene or multicluster) either will not be available, or the data will not be relevant unless the feature is configured
- What source returned the data object
- What type of job is being queried

- What is the state of the job being queried

Understanding the Blue Gene object model

To access the Blue Gene model, initialize the query object using `ll_query` with the `BLUE_GENE` query_type. The root of the Blue Gene model is the `BgMachine` object. The `BgMachine` is associated with one or more `BgBP` objects, one or more `BgSwitch` objects, one or more `BgWire` objects, and one or more `BgPartition` objects. These objects show how the Blue Gene system is configured.

Each `BgBP` object is associated with one or more `BgNodeCard` objects.

Each `BgPartition` object is associated with one or more `BgSwitch` objects.

Each `BgSwitch` object is associated with one or more `BgPortConn` objects.

`BgMachine` objects can only be obtained from the `LL_CM` query daemon.

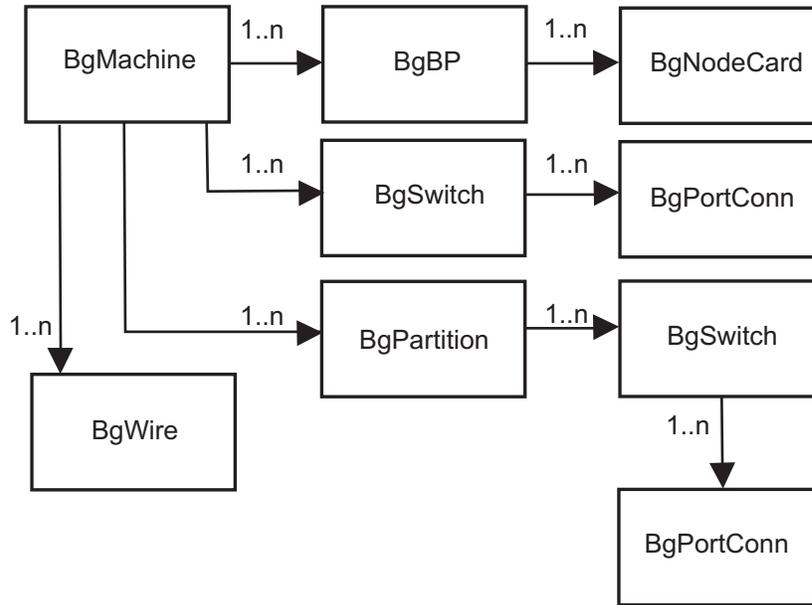


Figure 44. TWS LoadLeveler Blue Gene object model

See Table 87 on page 533 for a listing of `BLUE_GENE` specifications.

Understanding the Class object model

To access the Class model, initialize the query object using `ll_query` with the `CLASSES` query_type. The root of the class model is the `Class` object. The class is associated with zero or more `ResourceReq` objects and one or more `ClassUser` objects depending on the configuration of the class that the `Class` object represents.

`Class` objects can only be obtained from the `LL_CM` query daemon.

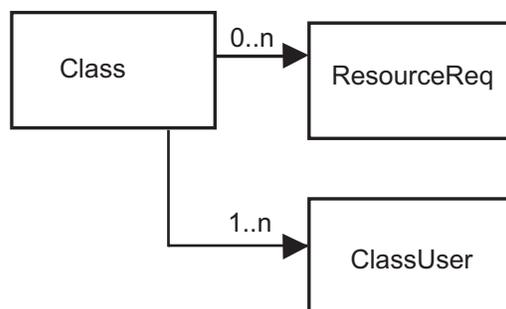


Figure 45. TWS LoadLeveler Class object model

See Table 88 on page 537 for a listing of CLASS specifications.

Understanding the Cluster object model

To access the Cluster model, initialize the query object using `ll_query` with the `CLUSTER` query_type. The root of the cluster model is the Cluster object. The Cluster is associated with one or more Resource objects.

Cluster objects can only be obtained from the LL_CM query daemon.

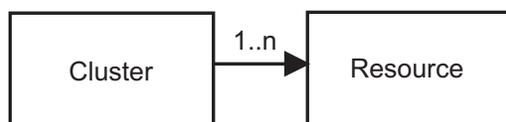


Figure 46. TWS LoadLeveler Cluster object model

See Table 89 on page 540 for a listing of CLUSTER specifications.

Understanding the Fairshare object model

To access the Fairshare model, initialize the query object using `ll_query` with the `FAIRSHARE` query_type. The Fairshare object model consists of Fairshare objects only. These objects represent fair share scheduling information.

Fairshare objects can only be obtained from the LL_CM query daemon.



Figure 47. TWS LoadLeveler Fairshare object model

See Table 90 on page 541 for a listing of FAIRSHARE specifications.

Understanding the Job object model

To access the Job model, initialize the query object using `ll_query` with the `JOBS` query_type. The root of the job model is the Job object. The Job is associated with a single Credential object, one or more Step objects, and zero or more ClusterFile objects.

The Step object represents one executable unit of the Job (all the tasks that are executed together). The Step is associated with one or more Machine objects, one

Data access API

| or more Node objects, one or more MachUsage objects, and one or more
| AdapterReq objects. The only relevant data within the Machine object when linked
| to a Step is the machine name. The list of Machines represents all of the hosts
| assigned to a step that is in a running-like state. If two or more nodes are running
| on the same host, the Machine object for the host occurs only once in the Step's
| Machine list.

| The MachUsage object is associated with one or more DispUsage objects. The
| DispUsage object is associated with one or more EventUsage objects. The
| MachUsage, DispUsage, and EventUsage objects contain accounting information
| available from the history file only.

| Each Node object manages a set of executables that share common requirements
| and preferences. The Node object is associated with one or more Task objects.

| The Task object represents one or more copies of the same executable. The Task
| object is associated with one or more ResourceReq objects and one or more
| TaskInstance objects.

| The TaskInstance object is associated with one or more AdapterUsage objects and
| one or more Adapter objects. The only relevant data within the Adapter objects
| associated with each TaskInstance is the adapter name. The list of Machines
| represents all of the hosts where one or more nodes of the step are running.

| Job objects can only be obtained from the LL_CM, LL_SCHEDD, LL_STARTD, and
| LL_HISTORY_FILE query daemon.
|

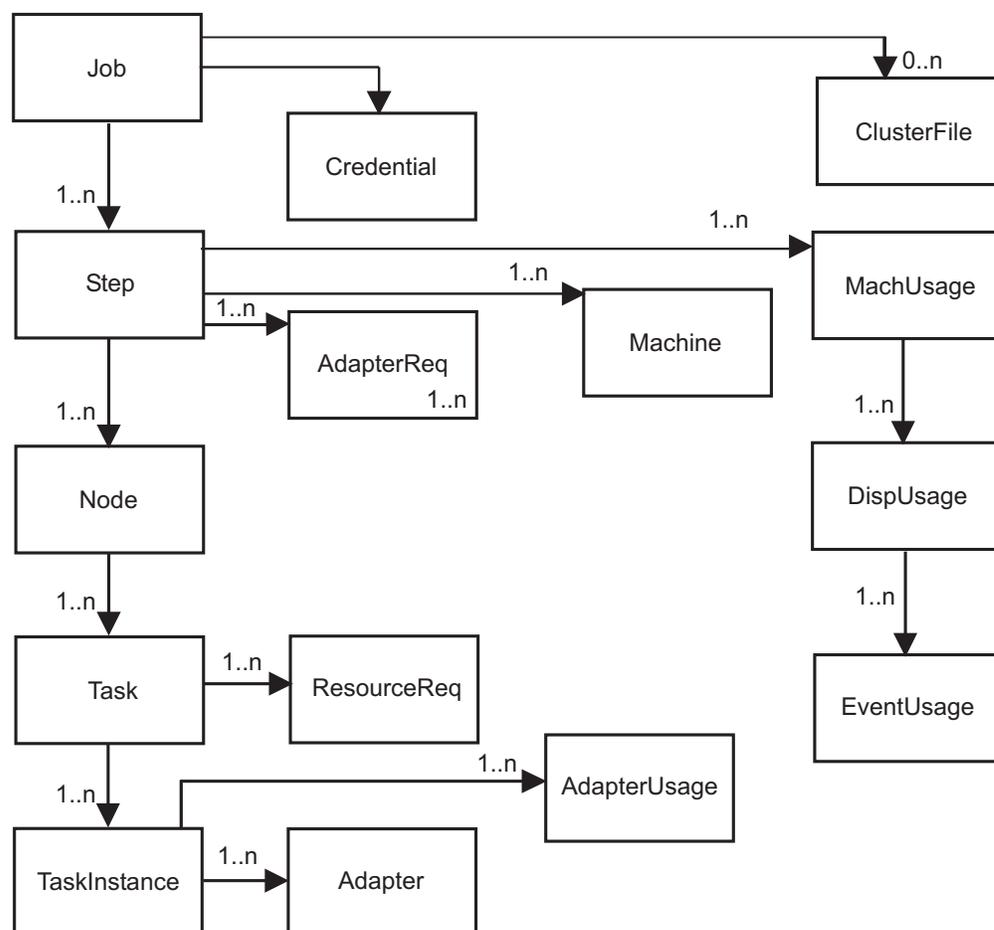


Figure 48. TWS LoadLeveler Job object model

See Table 91 on page 542 for a listing of JOB specifications.

Understanding the Machine object model

To access the Machine model, initialize the query object using `ll_query` with the `MACHINES` query_type. The root of the machine model is the Machine object. The machine is associated with one or more Resource objects, one or more Adapter objects, and one or more MCM objects depending on the configuration of the machine that the Machine object represents.

Machine objects can only be obtained from the LL_CM query daemon.

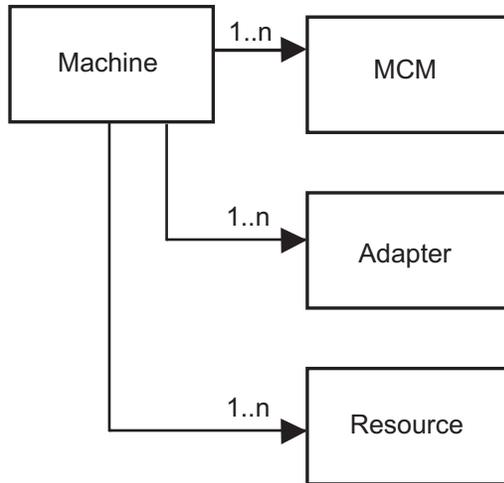


Figure 49. TWS LoadLeveler Machine object model

See Table 92 on page 566 for a listing of MACHINE specifications.

Understanding the MCluster object model

To access the MCluster model, initialize the query object using `ll_query` with the `MCLUSTERS` `query_type`. The MCluster object model consists of MCluster objects only. These objects represent configuration information for multicluster clusters.

MCluster objects can only be obtained from the `LL_SCHEDD` query daemon.

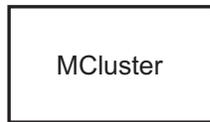


Figure 50. TWS LoadLeveler MCluster object model

See Table 93 on page 571 for a listing of MCLUSTER specifications.

Understanding the Reservations object model

To access the Reservations model, initialize the query object using `ll_query` with the `RESERVATIONS` `query_type`. The Reservations object model consists of Reservations objects only. These objects represent reservation information.

Reservations objects can only be obtained from the `LL_CM` query daemon.



Figure 51. TWS LoadLeveler Reservations object model

See Table 94 on page 572 for a listing of RESERVATIONS specifications.

Understanding the Wlmstat object model

To access the Wlmstat model, initialize the query object using ll_query with the WLMSTAT query_type. The Wlmstat object model consists of Wlmstat objects only. These objects return WLM statistics about running steps. Wlmstat objects must be queried for a specific step.

WlmStat objects can only be obtained from the LL_STARTD query daemon.



Figure 52. TWS LoadLeveler Wlmstat object model

See Table 95 on page 573 for a listing of WLMSTAT specifications.

ll_deallocate subroutine

Purpose

`ll_deallocate` – Deallocates the *query_element* allocated by the `ll_query` subroutine.

Library

LoadLeveler API library `libllapi.a` (AIX) or `libllapi.so` (Linux)

Syntax

```
#include "llapi.h"

int ll_deallocate (LL_element *query_element);
```

Parameters

query_element

Is a pointer to the `LL_element` returned by the `ll_query` function.

Description

query_element is the input field for this subroutine.

Return Values

This subroutine returns a zero to indicate success.

Error Values

-1 You specified a *query_element* that is not valid.

Related Information

Subroutines: `ll_free_objs`, `ll_get_data`, `ll_get_objs`, `ll_next_obj`, `ll_query`, `ll_reset_request`, `ll_set_request`

ll_free_objs subroutine

Purpose

ll_free_objs – Frees all of the **LL_element** objects in the *query_element* list that were obtained by the **ll_get_objs** subroutine. You must free the *query_element* by using the **ll_deallocate** subroutine.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int ll_free_objs (LL_element *query_element);
```

Parameters

query_element

Is a pointer to the **LL_element** returned by the **ll_query** function.

Description

query_element is the input field for this subroutine.

Return Values

This subroutine returns a zero to indicate success.

Error Values

-1 You specified a *query_element* that is not valid.

Related Information

Subroutines: **ll_free_objs**, **ll_get_data**, **ll_get_objs**, **ll_query**, **ll_reset_request**, **ll_set_request**

ll_get_data subroutine

Purpose

`ll_get_data` – Returns data from a valid `LL_element`.

Note: Before you use this subroutine, make sure you are familiar with the concepts discussed in “Understanding the Job object model” on page 525.

Library

LoadLeveler API library `libllapi.a` (AIX) or `libllapi.so` (Linux)

Syntax

```
#include "llapi.h"

int ll_get_data (LL_element *element, enum LLAPI_Specification specification,
                void* resulting_data_type);
```

Parameters

element

Is a pointer to the `LL_element` returned by the `ll_get_objs` subroutine.

specification

Specifies the data field within the data object you want to read.

resulting_data_type

Is a pointer to the location where you want the data stored. If the call returns a nonzero value, an error has occurred and the contents of the location are undefined.

Description

object and *specification* are input fields, while *resulting_data_type* is an output field.

The `ll_get_data` subroutine of the data access API allows you to access LoadLeveler objects. The parameters of `ll_get_data` are a LoadLeveler object (`LL_element`), a specification that indicates what information about the object is being requested, and a pointer to the area where the information being requested should be stored.

If the specification indicates an attribute of the element that is passed in, the result pointer must be the address of a variable of the appropriate type, and must be initialized to NULL. The type returned by each specification is found in the specification tables. If the specification queries the connection to another object, the returned value is of type `LL_element`. You can use a subsequent `ll_get_data` call to query information about the new object.

The data type `char*` and any arrays of type `int` or `char` must be freed by the caller.

`LL_element` pointers cannot be freed by the caller.

For the specifications, `LL_MachineOperatingSystem` and `LL_MachineArchitecture`, *resulting_data_type* returns the string "???" if a query is made before the associated records are updated with their actual values by the appropriate startd daemons.

Return Values

This subroutine returns a zero to indicate success.

Error Values

- 1 You specified an *object* that is not valid.
- 2 You specified an LLAPI_Specification that is not valid.

Related Information

Subroutines: ll_deallocate, ll_free_objs, ll_get_objs, ll_next_obj, ll_query, ll_reset_request, ll_set_request

See “Understanding the Blue Gene object model” on page 524 for more information on the Blue Gene object model.

Table 87. BLUE_GENE specifications for ll_get_data subroutine

Object	Specification	type of resulting data parameter	Description
BgBP	LL_BgBPCnodeMemory	int*	A pointer to an integer indicating the compute node (C-node) memory of the Blue Gene base partition (BgBPComputenodeMemory_t;).
BgBP	LL_BgBPCurrentPartition	char**	A pointer to a string containing the ID of the partition to which the Blue Gene base partition is part of.
BgBP	LL_BgBPCurrentPartitionState	int*	A pointer to an integer indicating the state of the current Blue Gene partition (BgPartitionState_t).
BgBP	LL_BgBPGetFirstNodeCard	LL_element* (BgNodeCard)	A pointer to the element associated with the first node card in the Blue Gene base partition.
BgBP	LL_BgBPGetNextNodeCard	LL_element* (BgNodeCard)	A pointer to the element associated with the next node card in the Blue Gene base partition.
BgBP	LL_BgBPId	char**	A pointer to a string containing the ID of the base partition in the Blue Gene system.
BgBP	LL_BgBPLocation	int**	A pointer to an array indicating the location of the base partition in the Blue Gene system in each dimension.
BgBP	LL_BgBPNodeCardCount	int*	A pointer to an integer indicating the number of node cards in the Blue Gene base partition.
BgBP	LL_BgBPState	int*	A pointer to an integer indicating the state of the Blue Gene base partition (BgBPState_t).
BgBP	LL_BgBPSubDividedBusy	int*	A pointer to an integer indicating that small partitions are active in the Blue Gene base partition.
BgMachine	LL_BgMachineBPSize	int**	A pointer to an array indicating the size of a Blue Gene base partition in compute nodes in each dimension.
BgMachine	LL_BgMachineGetFirstBP	LL_element* (BgBP)	A pointer to the element associated with the first base partition in the Blue Gene base partition list.

ll_get_data subroutine

Table 87. BLUE_GENE specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
BgMachine	LL_BgMachineGetFirstPartition	LL_element* (BgPartition)	A pointer to the element associated with the first partition in the Blue Gene partition list.
BgMachine	LL_BgMachineGetFirstSwitch	LL_element* (BgSwitch)	A pointer to the element associated with the first switch in the Blue Gene switch list.
BgMachine	LL_BgMachineGetFirstWire	LL_element* (BgWire)	A pointer to the element associated with the first wire in the Blue Gene wire list.
BgMachine	LL_BgMachineGetNextBP	LL_element* (BgBP)	A pointer to the element associated with the next base partition in the Blue Gene base partition list.
BgMachine	LL_BgMachineGetNextPartition	LL_element* (BgPartition)	A pointer to the element associated with the next partition in the Blue Gene partition list.
BgMachine	LL_BgMachineGetNextSwitch	LL_element* (BgSwitch)	A pointer to the element associated with the next switch in the Blue Gene switch list.
BgMachine	LL_BgMachineGetNextWire	LL_element* (BgWire)	A pointer to the element associated with the next wire in the Blue Gene wire list.
BgMachine	LL_BgMachinePartitionCount	int*	A pointer to an integer indicating the number of defined partitions in the Blue Gene system.
BgMachine	LL_BgMachineSize	int**	A pointer to an array indicating the size of the Blue Gene system in number of base partitions in each dimension.
BgMachine	LL_BgMachineSwitchCount	int*	A pointer to an integer indicating the number of switches in the Blue Gene system.
BgMachine	LL_BgMachineWireCount	int*	A pointer to an integer indicating the number of wires in the Blue Gene system.
BgNodeCard	LL_BgNodeCardCurrentPartition	char**	A pointer to a string containing the ID of the Blue Gene partition that the node card is assigned to.
BgNodeCard	LL_BgNodeCardCurrentPartitionState	int*	A pointer to an integer indicating the state of the current Blue Gene partition (BgPartitionState_t).
BgNodeCard	LL_BgNodeCardId	char**	A pointer to a string containing the ID of the Blue Gene node card.
BgNodeCard	LL_BgNodeCardQuarter	int*	A pointer to an integer indicating the quarter of the Blue Gene base partition the node card is in (BgQuarter_t).
BgNodeCard	LL_BgNodeCardState	int*	A pointer to an integer indicating the state of the Blue Gene node card (BgNodeCardState_t).
BgPartition	LL_BgPartitionBLRTSImage	char**	A pointer to a string containing the file name of the Blue Gene compute node's kernel image.

Table 87. BLUE_GENE specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
BgPartition	LL_BgPartitionBPCount	int*	A pointer to an integer indicating the number of Blue Gene base partitions in the partition.
BgPartition	LL_BgPartitionBPList	char***	A pointer to an array containing the list of Blue Gene base partition IDs assigned to the partition.
BgPartition	LL_BgPartitionConnection	int*	A pointer to an integer indicating the connection type of the Blue Gene partition (BgConnection_t).
BgPartition	LL_BgPartitionDescription	char**	A pointer to a string containing the Blue Gene partition description.
BgPartition	LL_BgPartitionGetFirstSwitch	LL_element* (BgSwitch)	A pointer to the element associated with the first switch in the Blue Gene partition.
BgPartition	LL_BgPartitionGetNextSwitch	LL_element* (BgSwitch)	A pointer to the element associated with the next switch in the Blue Gene partition.
BgPartition	LL_BgPartitionId	char**	A pointer to a string containing the ID of the Blue Gene partition.
BgPartition	LL_BgPartitionLinuxImage	char**	A pointer to a string containing the file name of the I/O nodes' Linux image.
BgPartition	LL_BgPartitionMLoaderImage	char**	A pointer to a string containing the file name of the machine loader image.
BgPartition	LL_BgPartitionMode	int*	A pointer to an integer indicating the node mode of the Blue Gene partition (BgNodeMode_t).
BgPartition	LL_BgPartitionNodeCardList	char***	A pointer to an array containing the list of node card IDs assigned to the Blue Gene partition.
BgPartition	LL_BgPartitionOwner	char**	A pointer to a string containing the user name of the owner of the Blue Gene partition.
BgPartition	LL_BgPartitionRamDiskImage	char**	A pointer to a string containing the file name of the ram disk image.
BgPartition	LL_BgPartitionShape	int**	A pointer to an array indicating the shape of the Blue Gene partition.
BgPartition	LL_BgPartitionSize	int*	A pointer to an integer indicating the size (number of C-nodes) of the Blue Gene partition.
BgPartition	LL_BgPartitionSmall	int*	A pointer to an integer indicating if the partition is smaller than a Blue Gene base partition.
BgPartition	LL_BgPartitionState	int*	A pointer to an integer indicating the state of the Blue Gene partition (BgPartitionState_t).
BgPartition	LL_BgPartitionSwitchCount	int*	A pointer to an integer indicating the number of switches in the Blue Gene partition.

ll_get_data subroutine

Table 87. BLUE_GENE specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
BgPortConn	LL_BgPortConnCurrentPartition	char**	A pointer to a string containing the ID of the Blue Gene partition to which the connection is assigned.
BgPortConn	LL_BgPortConnCurrentPartitionState	int*	A pointer to an integer indicating the state of the current Blue Gene partition (BgPartitionState_t).
BgPortConn	LL_BgPortConnFromSwitchPort	int*	A pointer to an integer indicating the from switch port ID (BgPort_t).
BgPortConn	LL_BgPortConnToSwitchPort	int*	A pointer to an integer indicating the to switch port ID (BgPort_t).
BgSwitch	LL_BgSwitchBasePartitionId	char**	A pointer to a string containing the ID of the base Blue Gene partition connected to the switch.
BgSwitch	LL_BgSwitchConnCount	int*	A pointer to an integer indicating the number of connections in the Blue Gene switch.
BgSwitch	LL_BgSwitchDimension	int*	A pointer to an integer indicating the dimension the Blue Gene switch is associated with (BgSwitchDimension_t).
BgSwitch	LL_BgSwitchGetFirstConn	LL_element* (BgPortConn)	A pointer to the element associated with the first connection in the Blue Gene switch connection list.
BgSwitch	LL_BgSwitchGetNextConn	LL_element* (BgPortConn)	A pointer to the element associated with the next connection in the Blue Gene switch connection list.
BgSwitch	LL_BgSwitchId	char**	A pointer to a string containing the ID of the Blue Gene switch.
BgSwitch	LL_BgSwitchState	int*	A pointer to an integer indicating the state of the Blue Gene switch (BgSwitchState_t).
BgWire	LL_BgWireCurrentPartition	char**	A pointer to a string containing the ID of the Blue Gene partition which the wire is assigned.
BgWire	LL_BgWireCurrentPartitionState	int*	A pointer to an integer indicating the state of the current Blue Gene partition (BgPartitionState_t).
BgWire	LL_BgWireFromPortCompId	char**	A pointer to a string containing the Blue Gene base partition or the switch the wire source port is part of.
BgWire	LL_BgWireFromPortId	int*	A pointer to an integer indicating the ID of the Blue Gene wire source port (BgPort_t).
BgWire	LL_BgWireId	char**	A pointer to a string containing the ID of the Blue Gene wire.
BgWire	LL_BgWireToPortCompId	char**	A pointer to a string containing the Blue Gene base partition or the switch the wire destination port is part of.

Table 87. BLUE_GENE specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
BgWire	LL_BgWireToPortID	int*	A pointer to an integer indicating the ID of the Blue Gene wire destination port (BgPort_t).
BgWire	LL_BgWireState	int*	A pointer to an integer indicating the state of the Blue Gene wire (BgWireState_t).

See “Understanding the Class object model” on page 524 for more information on the Class object model.

Table 88. CLASSES specifications for ll_get_data subroutine

Object	Specification	type of resulting data parameter	Description
Class	LL_ClassAdmin	char***	A pointer to an array of strings containing administrators for the class. The array ends with a NULL string.
Class	LL_ClassCkptDir	char**	A pointer to a string containing the directory for checkpoint files.
Class	LL_ClassCkptTimeHardLimit	int64_t*	A pointer to a 64-bit integer indicating the checkpoint time hard limit.
Class	LL_ClassCkptTimeSoftLimit	int64_t*	A pointer to a 64-bit integer indicating the checkpoint time soft limit.
Class	LL_ClassComment	char**	A pointer to a string containing the class comment.
Class	LL_ClassConstraints	int*	A pointer to an integer indicating whether the values of Maximum and Free Slots are constrained by MAX_STARTERS and MAXJOBS.
Class	LL_ClassCoreLimitHard	int64_t*	A pointer to a 64-bit integer indicating the core file hard limit.
Class	LL_ClassCoreLimitSoft	int64_t*	A pointer to a 64-bit integer indicating the core file soft limit.
Class	LL_ClassCpuLimitHard	int64_t*	A pointer to a 64-bit integer indicating the CPU hard limit.
Class	LL_ClassCpuLimitSoft	int64_t*	A pointer to a 64-bit integer indicating the CPU soft limit.
Class	LL_ClassCpuStepLimitHard	int64_t*	A pointer to a 64-bit integer indicating the CPU hard limit.
Class	LL_ClassCpuStepLimitSoft	int64_t*	A pointer to a 64-bit integer indicating the CPU soft limit.
Class	LL_ClassDataLimitHard	int64_t*	A pointer to a 64-bit integer indicating the data hard limit.
Class	LL_ClassDataLimitSoft	int64_t*	A pointer to a 64-bit integer indicating the data soft limit.
Class	LL_ClassDefWallClockLimitHard	int64_t*	A pointer to a 64-bit integer indicating the default wall clock hard limit.

ll_get_data subroutine

Table 88. CLASSES specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Class	LL_ClassDefWallClockLimitSoft	int64_t*	A pointer to a 64-bit integer indicating the default wall clock soft limit.
Class	LL_ClassExcludeGroups	char***	A pointer to an array of strings containing groups not permitted to use the class. The array ends with a NULL string.
Class	LL_ClassExcludeUsers	char***	A pointer to an array of strings containing users not permitted to use the class. The array ends with a NULL string.
Class	LL_ClassFileLimitHard	int64_t*	A pointer to a 64-bit integer indicating the file size hard limit.
Class	LL_ClassFileLimitSoft	int64_t*	A pointer to a 64-bit integer indicating the file size soft limit.
Class	LL_ClassFreeSlots	int*	A pointer to an integer indicating the number of available initiators.
Class	LL_ClassGetFirstResourceRequirement	LL_element* (ResourceReq)	A pointer to the element associated with the first resource requirement.
Class	LL_ClassGetFirstUser	LL_element* (ClassUser)	A pointer to the element associated with the first user defined in the class.
Class	LL_ClassGetNextResourceRequirement	LL_element* (ResourceReq)	A pointer to the element associated with the next resource requirement.
Class	LL_ClassGetNextUser	LL_element* (ClassUser)	A pointer to the element associated with the next user defined in the class.
Class	LL_ClassIncludeGroups	char***	A pointer to an array of strings containing groups permitted to use the class. The array ends with a NULL string.
Class	LL_ClassIncludeUsers	char***	A pointer to an array of strings containing users permitted to use the class. The array ends with a NULL string.
Class	LL_ClassMaximumSlots	int*	A pointer to an integer indicating the total number of configured initiators.
Class	LL_ClassMaxJobs	int*	A pointer to an integer indicating the maximum number of job steps that can run at any time.
Class	LL_ClassMaxProcessors	int*	A pointer to an integer indicating the maximum number of processors for a parallel job step.
Class	LL_ClassMaxProtocolInstances	int*	A pointer to an integer indicating the maximum number of adapter windows per protocol per task.
Class	LL_ClassMaxTotalTasks	int*	A pointer to an integer indicating the value for Max_total_tasks.
Class	LL_ClassName	char**	A pointer to a string containing the name of the class.

Table 88. CLASSES specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Class	LL_ClassNice	int*	A pointer to an integer indicating the nice value.
Class	LL_ClassPreemptClass	char**	A pointer to a string containing the PREEMPT_CLASS rule.
Class	LL_ClassPriority	int*	A pointer to an integer indicating the class system priority.
Class	LL_ClassRssLimitHard	int64_t*	A pointer to a 64-bit integer indicating the resident set size hard limit.
Class	LL_ClassRssLimitSoft	int64_t*	A pointer to a 64-bit integer indicating the resident set size soft limit.
Class	LL_ClassStackLimitHard	int64_t*	A pointer to a 64-bit integer indicating the stack size hard limit.
Class	LL_ClassStackLimitSoft	int64_t*	A pointer to a 64-bit integer indicating the stack size soft limit.
Class	LL_ClassStartClass	char**	A pointer to a string containing the START_CLASS rule.
Class	LL_ClassWallClockLimitHard	int64_t*	A pointer to a 64-bit integer indicating the wall clock hard limit.
Class	LL_ClassWallClockLimitSoft	int64_t*	A pointer to a 64-bit integer indicating the wall clock soft limit.
ClassUser	LL_ClassUserMaxIdle	int*	A pointer to an integer indicating the maximum number of idle job steps.
ClassUser	LL_ClassUserMaxJobs	int*	A pointer to an integer indicating the maximum number of running job steps.
ClassUser	LL_ClassUserMaxQueued	int*	A pointer to an integer indicating the maximum number of total job steps in the queue.
ClassUser	LL_ClassUserMaxTotalTasks	int*	A pointer to an integer indicating the maximum number of running tasks.
ClassUser	LL_ClassUserName	char**	A pointer to a string containing the user name.
ResourceReq	LL_ResourceRequirementName	char**	A pointer to a string containing the resource requirement name.
ResourceReq	LL_ResourceRequirementValue	int*	A pointer to an integer indicating the value of the resource requirement.
ResourceReq	LL_ResourceRequirementValue64	int64_t*	A pointer to a 64-bit integer indicating the value of the resource requirement.

See “Understanding the Cluster object model” on page 525 for more information on the Cluster object model.

ll_get_data subroutine

Table 89. CLUSTER specifications for ll_get_data subroutine

Object	Specification	type of resulting data parameter	Description
Cluster	LL_ClusterClusterMetric	char**	A pointer to a string containing the CLUSTER_METRIC string.
Cluster	LL_ClusterClusterRemoteJobFilter	char**	A pointer to a string containing the CLUSTER_REMOTE_JOB_FILTER string.
Cluster	LL_ClusterClusterUserMapper	char**	A pointer to a string containing the CLUSTER_USER_MAPPER string.
Cluster	LL_ClusterDefinedResourceCount	int*	A pointer to an integer indicating the number of consumable resources defined in the cluster.
Cluster	LL_ClusterDefinedResources	char***	A pointer to an array containing the names of consumable resources defined in the cluster. The array ends with a NULL string.
Cluster	LL_ClusterEnforcedResourceCount	int*	A pointer to an integer indicating the number of enforced resources
Cluster	LL_ClusterEnforcedResources	char***	A pointer to an array of characters indicating the number of enforced resources
Cluster	LL_ClusterEnforceMemory	int*	A pointer to a boolean integer indicating absolute memory limit.
Cluster	LL_ClusterEnforceSubmission	int*	A pointer to a boolean integer indicating resources required at time of submission.
Cluster	LL_ClusterGetFirstResource	LL_element* (Resource)	A pointer to the element associated with the first resource.
Cluster	LL_ClusterGetNextResource	LL_element* (Resource)	A pointer to the element associated with the next resource.
Cluster	LL_ClusterMusterEnvironment	int*	A pointer to an integer indicating that the multicluster environment is enabled.
Cluster	LL_ClusterSchedulerType	char**	A pointer to a string containing the scheduler type.
Cluster	LL_ClusterSchedulingResourceCount	int*	A pointer to an integer indicating the number of consumable resources considered by the scheduler for the cluster.
Cluster	LL_ClusterSchedulingResources	char***	A pointer to an array containing the names of consumable resources considered by the scheduler for the cluster. The array ends with a NULL string.
Resource	LL_ResourceAvailableValue	int*	A pointer to an integer indicating the value of available resources.
Resource	LL_ResourceAvailableValue64	int64_t*	A pointer to a 64-bit integer indicating the value of available resources.
Resource	LL_ResourceInitialValue	int*	A pointer to an integer indicating the initial resource value.
Resource	LL_ResourceInitialValue64	int64_t*	A pointer to a 64-bit integer indicating the initial resource value.

Table 89. CLUSTER specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Resource	LL_ResourceName	char**	A pointer to a string containing the resource name.

See “Understanding the Fairshare object model” on page 525 for more information on the Fairshare object model.

Table 90. FAIRSHARE specifications for ll_get_data subroutine

Object	Specification	type of resulting data parameter	Description
FairShare	LL_FairShareAllocatedShares	int**	A pointer to an array indicating the number of allocated shares for each user or group. LL_FairShareNumberOfEntries indicates the size of this array.
FairShare	LL_FairShareCurrentTime	time_t*	A pointer to a time_t structure indicating the time that the information is obtained.
FairShare	LL_FairShareEntryNames	char***	A pointer to an array containing the names of users or groups whose fair share information has been returned. LL_FairShareNumberOfEntries indicates the size of this array.
FairShare	LL_FairShareEntryTypes	int**	A pointer to an array indicating the types of names: 0 for users, 1 for groups. LL_FairShareNumberOfEntries indicates the size of this array.
FairShare	LL_FairShareInterval	int*	A pointer to an integer indicating the time interval that is most important to fair share scheduling.
FairShare	LL_FairShareNumberOfEntries	int*	A pointer to an integer indicating the number of users or groups for which the fair share information has been returned.
FairShare	LL_FairShareTotalShares	int*	A pointer to an integer indicating the total number of shares in the cluster.
FairShare	LL_FairShareUsedBgShares	int**	A pointer to an array indicating the number of Blue Gene shares used by each user or group.
FairShare	LL_FairShareUsedShares	int**	A pointer to an array indicating the number of shares used by each user or group. LL_FairShareNumberOfEntries indicates the size of this array.

See “Understanding the Job object model” on page 525 for more information on the Job object model.

Note the following in Table 91 on page 542:

- Any specifications that are only available from a specific query_daemon will have that described in the “Description” column.

ll_get_data subroutine

- Any specification that does not have any detailed query_daemon in the "Description" column is available from the LL_CM, LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE query_daemon.

Table 91. JOBS specifications for ll_get_data subroutine

Object	Specification	type of resulting data parameter	Description
Adapter	LL_AdapterName	char**	A pointer to a string containing the adapter name. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
AdapterReq	LL_AdapterReqCommLevel	int*	A pointer to the integer indicating the adapter's communication level. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
AdapterReq	LL_AdapterReqInstances	int*	A pointer to an integer containing the requested adapter instances. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
AdapterReq	LL_AdapterReqMode	char**	A pointer to a string containing the requested adapter mode (IP or US). Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
AdapterReq	LL_AdapterReqProtocol	char**	A pointer to a string containing the requested adapter protocol. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
AdapterReq	LL_AdapterReqRcxtBlocks	int*	A pointer to the integer indicating the number of rCxt blocks requested for the adapter usage. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
AdapterReq	LL_AdapterReqTypeName	char**	A pointer to a string containing the requested adapter type. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
AdapterReq	LL_AdapterReqUsage	int*	A pointer to the integer indicating the requested adapter usage. This integer will be one of the values defined in the Usage enum. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
AdapterUsage	LL_AdapterUsageDeviceDriver	char**	A pointer to a string containing the name of the adapter device being used. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
AdapterUsage	LL_AdapterUsageLmc	int*	A pointer to an integer indicating the logical mask on an InfiniBand adapter port. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
AdapterUsage	LL_AdapterUsageMode	char**	A pointer to a string containing the adapter usage mode of IP or US. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
AdapterUsage	LL_AdapterUsageNetmask	char**	A pointer to a string containing the netmask of the adapter in the AdapterUsage object. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
AdapterUsage	LL_AdapterUsagePortNumber	int*	A pointer to an integer indicating the port number on an InfiniBand adapter port. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
AdapterUsage	LL_AdapterUsageProtocol	char**	A pointer to a string containing the task's protocol. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
AdapterUsage	LL_AdapterUsageRcxtBlocks	int*	A pointer to the integer indicating the number of rCxt blocks associated with the adapter usage. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
AdapterUsage	LL_AdapterUsageTag	char**	A pointer to a character string that indicates which switch table the adapter usage is in. Adapter usages with the same tag are in the same switch table. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
AdapterUsage	LL_AdapterUsageWindow	int*	Contains the adapter window assigned to the task. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
AdapterUsage	LL_AdapterUsageWindowMemory64	uint64_t*	A pointer to an unsigned 64-bit integer indicating the number of bytes used by the adapter window. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
ClusterFile	LL_ClusterFileLocalPath	char**	A pointer to a string containing the expanded local file pathname. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
ClusterFile	LL_ClusterFileRemotePath	char**	A pointer to a string containing the expanded remote file pathname. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Credential	LL_CredentialGid	int*	A pointer to an integer containing the UNIX gid of the user submitting the job.
Credential	LL_CredentialGroupName	char**	A pointer to a string containing the UNIX group name of the user submitting the job.
Credential	LL_CredentialUid	int*	A pointer to an integer containing the UNIX uid of the person submitting the job.

ll_get_data subroutine

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Credential	LL_CredentialUserName	char**	A pointer to a string containing the user ID of the user submitting the job.
DispUsage	LL_DispUsageEventUsageCount	int*	A pointer to an integer indicating the count of event usages. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DispUsageGetFirstEventUsage	LL_element* (EventUsage)	A pointer to the element associated with the first event usage. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DispUsageGetNextEventUsage	LL_element* (EventUsage)	A pointer to the element associated with the next event usage. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DispUsageStarterIdrss64	int64_t*	A pointer to a 64-bit integer indicating the amount of unshared memory in the data segment of a process. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DispUsageStarterInblock64	int64_t*	A pointer to a 64-bit integer indicating the number of times the file system performed input. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DispUsageStarterIsrss64	int64_t*	A pointer to a 64-bit integer indicating the unshared stack size. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DispUsageStarterIxrss64	int64_t*	A pointer to a 64-bit integer indicating the amount of memory used by the text segment that was also shared among other processes. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DispUsageStarterMajflt64	int64_t*	A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DispUsageStarterMaxrss64	int64_t*	A pointer to a 64-bit integer indicating the maximum resident set size utilized. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DispUsageStarterMinflt64	int64_t*	A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DispUsageStarterMsgrcv64	int64_t*	A pointer to a 64-bit integer indicating the number of IPC messages received. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DispUsageStarterMsgsnd64	int64_t*	A pointer to a 64-bit integer indicating the number of IPC messages sent. Data is available from the LL_HISTORY_FILE only.

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
DispUsage	LL_DisUsageStarterNivcsw64	int64_t*	A pointer to a 64-bit integer indicating the number of involuntary context switches. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStarterNsignals64	int64_t*	A pointer to a 64-bit integer indicating the number of signals delivered. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStarterNswap64	int64_t*	A pointer to a 64-bit integer indicating the number of times swapped out. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStarterNvcsw64	int64_t*	A pointer to a 64-bit integer indicating the number of context switches due to voluntarily giving up processor. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStarterOublock64	int64_t*	A pointer to a 64-bit integer indicating the number of times the file system performed output. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStarterSystemTime64	int64_t*	A pointer to a 64-bit integer indicating the CPU system time. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStarterUserTime64	int64_t*	A pointer to a 64-bit integer indicating the CPU user time. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStepIdrss64	int64_t*	A pointer to a 64-bit integer indicating the amount of unshared memory in the data segment of a process. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStepInblock64	int64_t*	A pointer to a 64-bit integer indicating the number of times the file system performed input. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStepIsrss64	int64_t*	A pointer to a 64-bit integer indicating the unshared stack size. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStepIxrss64	int64_t*	A pointer to a 64-bit integer indicating the amount of memory used by the text segment that was also shared among other processes. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStepMajflt64	int64_t*	A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LL_HISTORY_FILE only.

ll_get_data subroutine

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
DispUsage	LL_DisUsageStepMaxrss64	int64_t*	A pointer to a 64-bit integer indicating the maximum resident set size utilized. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStepMinflt64	int64_t*	A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStepMsgrcv64	int64_t*	A pointer to a 64-bit integer indicating the number of IPC messages received. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStepMsgsnd64	int64_t*	A pointer to a 64-bit integer indicating the number of IPC messages sent. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStepNivcsw64	int64_t*	A pointer to a 64-bit integer indicating the number of involuntary context switches. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStepNsignals64	int64_t*	A pointer to a 64-bit integer indicating the number of signals delivered. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStepNswap64	int64_t*	A pointer to a 64-bit integer indicating the number of times swapped out. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStepNvcsw64	int64_t*	A pointer to a 64-bit integer indicating the number of context switches due to voluntarily giving up processor. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStepOublock64	int64_t*	A pointer to a 64-bit integer indicating the number of times the file system performed output. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStepSystemTime64	int64_t*	A pointer to a 64-bit integer indicating the CPU system time. Data is available from the LL_HISTORY_FILE only.
DispUsage	LL_DisUsageStepUserTime64	int64_t*	A pointer to a 64-bit integer indicating the CPU user time. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageEventID	int*	A pointer to an integer indicating the event ID. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageEventName	char**	A pointer to a string containing the event name. Data is available from the LL_HISTORY_FILE only.

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
EventUsage	LL_EventUsageEventTimestamp	int*	A pointer to an integer indicating the event timestamp. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStarterIdrss64	int64_t*	A pointer to a 64-bit integer indicating the amount of unshared memory in the data segment of a process. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStarterInblock64	int64_t*	A pointer to a 64-bit integer indicating the number of times the file system performed input. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStarterIsrss64	int64_t*	A pointer to a 64-bit integer indicating the unshared stack size. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStarterIxrss64	int64_t*	A pointer to a 64-bit integer indicating the amount of memory used by the text segment that was also shared among other processes. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStarterMajflt64	int64_t*	A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStarterMaxrss64	int64_t*	A pointer to a 64-bit integer indicating the maximum resident set size utilized. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStarterMinflt64	int64_t*	A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStarterMsgrcv64	int64_t*	A pointer to a 64-bit integer indicating the number of IPC messages received. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStarterMsgsnd64	int64_t*	A pointer to a 64-bit integer indicating the number of IPC messages sent. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStarterNivcsw64	int64_t*	A pointer to a 64-bit integer indicating the number of involuntary context switches. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStarterNsignals64	int64_t*	A pointer to a 64-bit integer indicating the number of signals delivered. Data is available from the LL_HISTORY_FILE only.

ll_get_data subroutine

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
EventUsage	LL_EventUsageStarterNswap64	int64_t*	A pointer to a 64-bit integer indicating the number of times swapped out. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStarterNvcsw64	int64_t*	A pointer to a 64-bit integer indicating the number of context switches due to voluntarily giving up processor. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStarterOublock64	int64_t*	A pointer to a 64-bit integer indicating the number of times the file system performed output. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStarterSystemTime64	int64_t*	A pointer to a 64-bit integer indicating the CPU system time. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStarterUserTime64	int64_t*	A pointer to a 64-bit integer indicating the CPU user time. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStepIdrssi64	int64_t*	A pointer to a 64-bit integer indicating the amount of unshared memory in the data segment of a process. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStepInblock64	int64_t*	A pointer to a 64-bit integer indicating the number of times the file system performed input. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStepIsrssi64	int64_t*	A pointer to a 64-bit integer indicating the unshared stack size. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStepIxrssi64	int64_t*	A pointer to a 64-bit integer indicating the amount of memory used by the text segment that was also shared among other processes. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStepMajflt64	int64_t*	A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStepMaxrss64	int64_t*	A pointer to a 64-bit integer indicating the maximum resident set size utilized. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStepMinflt64	int64_t*	A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LL_HISTORY_FILE only.

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
EventUsage	LL_EventUsageStepMsgrcv64	int64_t*	A pointer to a 64-bit integer indicating the number of IPC messages received. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStepMsgsnd64	int64_t*	A pointer to a 64-bit integer indicating the number of IPC messages sent. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStepNivcsw64	int64_t*	A pointer to a 64-bit integer indicating the number of involuntary context switches. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStepNsignals64	int64_t*	A pointer to a 64-bit integer indicating the number of signals delivered. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStepNswap64	int64_t*	A pointer to a 64-bit integer indicating the number of times swapped out. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStepNvcsw64	int64_t*	A pointer to a 64-bit integer indicating the number of context switches due to voluntarily giving up processor. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStepOublock64	int64_t*	A pointer to a 64-bit integer indicating the number of times the file system performed output. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStepSystemTime64	int64_t*	A pointer to a 64-bit integer indicating the CPU system time. Data is available from the LL_HISTORY_FILE only.
EventUsage	LL_EventUsageStepUserTime64	int64_t*	A pointer to a 64-bit integer indicating the CPU user time. Data is available from the LL_HISTORY_FILE only.
Job	LL_JobCredential	LL_element* (Credential)	A pointer to the element associated with the job credential. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Job	LL_JobGetFirstClusterInputFile	LL_element* (ClusterFile)	A pointer to the element associated with the first input ClusterFile.
Job	LL_JobGetFirstClusterOutputFile	LL_element* (ClusterFile)	A pointer to the element associated with the first output ClusterFile.
Job	LL_JobGetFirstStep	LL_element* (Step)	A pointer to the element associated with the first step of the job, to be used in subsequent ll_get_data calls. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Job	LL_JobGetNextClusterInputFile	LL_element* (ClusterFile)	A pointer to the element associated with the next input ClusterFile.

ll_get_data subroutine

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Job	LL_JobGetNextClusterOutputFile	LL_element* (ClusterFile)	A pointer to the element associated with the next output ClusterFile.
Job	LL_JobGetNextStep	LL_element* (Step)	A pointer to the element associated with the next step. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Job	LL_JobIsRemote	int*	A pointer to an integer. If the integer contains the value 1, the job is remote. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Job	LL_JobJobQueueKey	int*	A pointer to an integer indicating the key used to write the job to the spool. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Job	LL_JobLocalOutboundSchedds	char***	A pointer to an array containing a list of local outbound Schedds. The last Schedd in the list is the current outbound Schedd. Data is available when the multicluster environment is configured. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Job	LL_JobName	char**	A pointer to a character string containing the job name.
Job	LL_JobRequestedCluster	char***	A pointer to an array containing the list of user-requested clusters. Data is available when the multicluster environment is configured. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Job	LL_JobSchedd	char**	A pointer to a string containing the Schedd managing the job.
Job	LL_JobScheddHistory	char***	A pointer to an array containing a list of managing Schedds. The last Schedd in the list is the current managing Schedd. Data is available when the multicluster environment is configured. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Job	LL_JobSchedulingCluster	char**	A pointer to a string containing the name of the cluster where the job is scheduled. Data is available when the multicluster environment is configured. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Job	LL_JobSendingCluster	char**	A pointer to a string containing the name of the sending cluster. Data is available when the multicluster environment is configured. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Job	LL_JobStepCount	int*	A pointer to an integer indicating the number of steps connected to the job.
Job	LL_JobStepType	int*	A pointer to an integer indicating the type of job, which can be INTERACTIVE_JOB or BATCH_JOB.
Job	LL_JobSubmitHost	char**	A pointer to a character string containing the name of the host machine from which the job was submitted.
Job	LL_JobSubmitTime	time_t*	A pointer to the time_t structure indicating when the job was submitted.
Job	LL_JobSubmittingCluster	char**	A pointer to a string containing the name of the submitting cluster. Data is available when the multicluster environment is configured. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Job	LL_JobSubmittingUser	char**	A pointer to a string containing the name of the submitting user. Data is available when the multicluster environment is configured. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Job	LL_JobUsersJCF	char**	A pointer to a single string containing all of the user's job command file keyword statements. Data is available from the LL_SCHEDD only. Data is available when the multicluster environment is configured.
Job	LL_JobVersionNum	int*	A pointer to an integer indicating the job's version number.
Machine	LL_MachineName	char**	A pointer to a string containing the machine name.
MachUsage	LL_MachUsageDispUsageCount	int*	A pointer to an integer indicating the count of dispatch usages. Data is available from the LL_HISTORY_FILE only.
MachUsage	LL_MachUsageGetFirstDispUsage	LL_element* (DispUsage)	A pointer to the element associated with the first dispatch usage. Data is available from the LL_HISTORY_FILE only.
MachUsage	LL_MachUsageGetNextDispUsage	LL_element* (DispUsage)	A pointer to the element associated with the next dispatch usage. Data is available from the LL_HISTORY_FILE only.
MachUsage	LL_MachUsageMachineName	char**	A pointer to a string containing the machine name. Data is available from the LL_HISTORY_FILE only.
MachUsage	LL_MachUsageMachineSpeed	double*	A pointer to a double containing the machine speed. Data is available from the LL_HISTORY_FILE only.

ll_get_data subroutine

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Node	LL_NodeGetFirstTask	LL_element* (Task)	A pointer to the element associated with the first task for this node.
Node	LL_NodeGetNextTask	LL_element* (Task)	A pointer to the element associated with the next task for this node.
Node	LL_NodeInitiatorCount	int*	A pointer to an integer indicating the number of tasks running on the node.
Node	LL_NodeMaxInstances	int*	A pointer to an integer indicating the maximum number of machines requested.
Node	LL_NodeMinInstances	int*	A pointer to an integer indicating the minimum number of machines requested.
Node	LL_NodeRequirements	char**	A pointer to a string containing the node requirements.
Node	LL_NodeTaskCount	int*	A pointer to an integer indicating the different types of tasks running on the node.
ResourceReq	LL_ResourceRequirementName	char**	A pointer to a string containing the resource requirement name. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
ResourceReq	LL_ResourceRequirementValue	int*	A pointer to an integer indicating the value of the resource requirement. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
ResourceReq	LL_ResourceRequirementValue64	int64_t*	A pointer to a 64-bit integer indicating the value of the resource requirement. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepAccountNumber	char**	A pointer to a string containing the account number specified by the user submitting the job.
Step	LL_StepAcctKey	int64_t*	A pointer to a 64-bit integer that can be used to identify all of the AIX accounting records for the job step. Data is available from the LL_HISTORY_FILE only.
Step	LL_StepBgErrorText	char**	A pointer to a string containing the error text for the Blue Gene job record in the Blue Gene database. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepBgJobId	char**	A pointer to a string containing the ID of the Blue Gene job in the Blue Gene database. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Step	LL_StepBgJobState	int*	A pointer to an integer indicating the state of the Blue Gene job in the Blue Gene database (BgJobState_t). Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepBgPartitionAllocated	char**	A pointer to a string containing the ID of the Blue Gene partition allocated for the job. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepBgPartitionRequested	char**	A pointer to a string containing the ID of the Blue Gene partition requested for the job step. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepBgPartitionState	int*	A pointer to an integer indicating the state of the Blue Gene partition allocated for the job step (BgPartitionState_t). Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepBgShapeAllocated	int**	A pointer to an array indicating the shape of the Blue Gene compute nodes allocated for the job step. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepBgShapeRequested	int**	A pointer to an array indicating the shape of Blue Gene compute nodes requested for the job step. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepBgSizeAllocated	int*	A pointer to an integer indicating the number of Blue Gene compute nodes allocated for the job step. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepBgSizeRequested	int*	A pointer to an integer indicating the number of Blue Gene compute nodes requested for the job step. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepBgWiringAllocated	int*	A pointer to an integer indicating the allocated type of wiring for the Blue Gene job (BgConnection_t). Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepBgWiringRequested	int*	A pointer to an integer indicating the requested type of wiring for the Blue Gene job (BgConnection_t). Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.

ll_get_data subroutine

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Step	LL_StepBlocking	int*	A pointer to an integer representing blocking as specified by the user in the job command file. <ul style="list-style-type: none"> • Returns -1 if unlimited is specified • Returns 0 if blocking is unspecified Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepBulkXfer	int*	A pointer to an integer that is set to 1 if the step requested bulk transfer and 0 if it did not. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepCheckpointable	int*	A pointer to an integer indicating if checkpointing was enabled via the checkpoint keyword (0=disabled, 1=enabled). Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepCheckpointing	int*	A pointer to an integer indicating that a checkpoint is currently being taken for the step. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepCkptAccumTime	int*	A pointer to an integer indicating the amount of accumulated time, in seconds, that the job step has spent checkpointing. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepCkptExecuteDirectory	char**	A pointer to a string containing the directory where the job step's executable will be saved. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepCkptFailStartTime	time_t*	A pointer to a time_t structure indicating the start time of the last unsuccessful checkpoint. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepCkptFile	char**	A pointer to a string containing the directory and file name which contain checkpoint information for the last successful checkpoint.
Step	LL_StepCkptGoodElapseTime	int*	A pointer to an integer indicating the amount of time, in seconds, it took for the last successful checkpoint to complete. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Step	LL_StepCkptGoodStartTime	time_t*	A pointer to a time_t structure indicating the start time of the last successful checkpoint. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepCkptRestart	int*	A pointer to an integer indicating the value specified by the user for the restart_from_ckpt keyword (0= no, 1= yes). Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepCkptRestartSameNodes	int*	A pointer to a string indicating the value specified by the user for the restart_on_same_nodes keyword (0= no, 1= yes). Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepCkptTimeHardLimit	int*	A pointer to an integer indicating the hard limit set by the user in the ckpt_time_limit keyword.
Step	LL_StepCkptTimeHardLimit64	int64_t*	A pointer to a 64-bit integer indicating the hard limit set by the user in the ckpt_time_limit keyword.
Step	LL_StepCkptTimeSoftLimit	int*	A pointer to an integer indicating the soft limit set by the user in ckpt_time_limit keyword.
Step	LL_StepCkptTimeSoftLimit64	int64_t*	A pointer to a 64-bit integer indicating the soft limit set by the user in ckpt_time_limit keyword.
Step	LL_StepClassSystemPriority	int*	A pointer to an integer indicating the class priority of the job step. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepComment	char**	A pointer to a string indicating the comment specified by the user submitting the job.
Step	LL_StepCompletionCode	int*	A pointer to an integer indicating the completion code of the step.
Step	LL_StepCompletionDate	time_t*	A pointer to a time_t structure indicating the completion date of the step.
Step	LL_StepCoreLimitHard	int*	A pointer to an integer indicating the core hard limit set by the user in the core_limit keyword.
Step	LL_StepCoreLimitHard64	int64_t*	A pointer to a 64-bit integer indicating the core hard limit set by the user in the core_limit keyword.
Step	LL_StepCoreLimitSoft	int*	A pointer to an integer indicating the core soft limit set by the user in the core_limit keyword.

ll_get_data subroutine

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Step	LL_StepCoreLimitSoft64	int64_t*	A pointer to a 64-bit integer indicating the core soft limit set by the user in the core_limit keyword.
Step	LL_StepCoschedule	int*	A pointer to an integer indicating if a job step is coscheduled set by the user in the coschedule keyword.
Step	LL_StepCpuLimitHard	int*	A pointer to an integer indicating the CPU hard limit set by the user in the cpu_limit keyword.
Step	LL_StepCpuLimitHard64	int64_t*	A pointer to a 64-bit integer indicating the CPU hard limit set by the user in the cpu_limit keyword.
Step	LL_StepCpuLimitSoft	int*	A pointer to an integer indicating the CPU soft limit set by the user in the cpu_limit keyword.
Step	LL_StepCpuLimitSoft64	int64_t*	A pointer to a 64-bit integer indicating the CPU soft limit set by the user in the cpu_limit keyword.
Step	LL_StepCpuStepLimitHard	int*	A pointer to an integer indicating the CPU step hard limit set by the user in the job_cpu_limit keyword.
Step	LL_StepCpuStepLimitHard64	int64_t*	A pointer to a 64-bit integer indicating the CPU step hard limit set by the user in the job_cpu_limit keyword.
Step	LL_StepCpuStepLimitSoft	int*	A pointer to an integer indicating the CPU step soft limit set by the user in the job_cpu_limit keyword.
Step	LL_StepCpuStepLimitSoft64	int64_t*	A pointer to a 64-bit integer indicating the CPU step soft limit set by the user in the job_cpu_limit keyword.
Step	LL_StepDataLimitHard	int*	A pointer to an integer indicating the data hard limit set by the user in the data_limit keyword.
Step	LL_StepDataLimitHard64	int64_t*	A pointer to a 64-bit integer indicating the data hard limit set by the user in the data_limit keyword.
Step	LL_StepDataLimitSoft	int*	A pointer to an integer indicating the data soft limit set by the user in the data_limit keyword.
Step	LL_StepDataLimitSoft64	int64_t*	A pointer to a 64-bit integer indicating the data soft limit set by the user in the data_limit keyword.
Step	LL_StepDependency	char**	A pointer to a string containing the step dependency value. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Step	LL_StepDispatchTime	time_t*	A pointer to a time_t structure indicating the time the negotiator dispatched the job. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepEnvironment	char**	A pointer to a string containing the environment variables set by the user in the executable. Data is available from LL_SCHEDD and LL_HISTORY_FILE.
Step	LL_StepErrorFile	char**	A pointer to a string containing the standard error file name used by the executable.
Step	LL_StepExecSize	int*	A pointer to an integer indicating the executable size.
Step	LL_StepFavoredJob	int*	A pointer to an integer that specifies whether the step is favored using the llfavorjob command. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepFileLimitHard	int*	A pointer to an integer indicating the file hard limit set by the user in the file_limit keyword.
Step	LL_StepFileLimitHard64	int64_t*	A pointer to a 64-bit integer indicating the file hard limit set by the user in the file_limit keyword.
Step	LL_StepFileLimitSoft	int*	A pointer to an integer indicating the file soft limit set by the user in the file_limit keyword.
Step	LL_StepFileLimitSoft64	int64_t*	A pointer to a 64-bit integer indicating the file soft limit set by the user in the file_limit keyword.
Step	LL_StepGetFirstAdapterReq	LL_element* (AdapterReq)	A pointer to the element associated with the first adapter requirement. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepGetFirstMachine	LL_element* (Machine)	A pointer to the element associated with the first machine in the step. Data is available from LL_CM, LL_SCHEDD, and LL_STARTD.
Step	LL_StepGetFirstMachUsage	LL_element* (MachUsage)	A pointer to the element associated with the first machine usage in the list of machine usages where a single machine usage corresponds to any machine that was ever assigned to the step. Data is available from the LL_HISTORY_FILE only.
Step	LL_StepGetFirstNode	LL_element* (Node)	A pointer to the element associated with the first node of the step.
Step	LL_StepGetMasterTask	LL_element* (Task)	A pointer to the element associated with the master task of the step.

ll_get_data subroutine

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Step	LL_StepGetNextAdapterReq	LL_element* (AdapterReq)	A pointer to the element associated with the next adapter requirement. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepGetNextMachine	LL_element* (Machine)	A pointer to the element associated with the next machine of the step. Data is available from LL_CM, LL_SCHEDD, and LL_STARTD.
Step	LL_StepGetNextMachUsage	LL_element* (MachUsage)	A pointer to the element associated with the next machine usage in the list of machine usages where a single machine usage corresponds to any machine that was ever assigned to the step. Data is available from the LL_HISTORY_FILE only.
Step	LL_StepGetNextNode	LL_element* (Node)	A pointer to the element associated with the next node of the step.
Step	LL_StepGroupSystemPriority	int*	A pointer to an integer indicating the group priority of a job step. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepHoldType	int*	A pointer to an integer indicating the hold state of the step (user, system, and so on). The value returned is in the HoldType enum. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepHostList	char***	A pointer to an array containing the list of hosts in the host.list file associated with the step. The array ends with a null string. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepID	char**	A pointer to a string containing the ID of the step.
Step	LL_StepImageSize	int*	A pointer to an integer indicating the image size of the executable. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepImageSize64	int64_t*	A pointer to a 64-bit integer indicating the image size of the executable. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepInputFile	char**	A pointer to a string containing the standard input file name used by the executable.
Step	LL_StepIwd	char**	A pointer to a string containing the initial working directory name used by the executable.
Step	LL_StepJobClass	char**	A pointer to a string containing the class of the step.

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Step	LL_StepLargePage	char**	A pointer to a string containing the Large Page level of support associated with the job step.
Step	LL_StepLoadLevelerGroup	char**	A pointer to a string containing the name of the LoadLeveler group specified by the step.
Step	LL_StepMachineCount	int*	A pointer to an integer indicating the number of machines assigned to the step when it is in a running-like state. Data is available from LL_CM, LL_SCHEDD, and LL_STARTD.
Step	LL_StepMachUsageCount	int*	A pointer to an integer indicating the number of machine usage objects corresponding to the set of machines that were ever assigned to the step. Data is available from the LL_HISTORY_FILE only.
Step	LL_StepMessages	char**	A pointer to a string containing a list of messages from LoadLeveler. This data is available from the LL_CM only.
Step	LL_StepName	char**	A pointer to a string containing the name of the step.
Step	LL_StepNodeCount	int*	A pointer to an integer indicating the number of node objects associated with the step.
Step	LL_StepNodeUsage	int*	A pointer to an integer indicating the node usage specified by the user (SHARED or NOT_SHARED). The value returned is in the enum Usage.
Step	LL_StepOutputFile	char**	A pointer to a character string containing the standard output file name used by the executable.
Step	LL_StepParallelMode	int*	A pointer to an integer indicating the mode of the step.
Step	LL_StepPreemptable	int*	A pointer to an integer indicating whether the job step is preemptable. The integer is set to 0 if the job step is not preemptable and is set to 1 if the job step is preemptable.
Step	LL_StepPreemptWaitList	char***	A pointer to an array containing the job steps that an idle job must preempt. The array ends with a NULL string. This data is available from the LL_CM only.
Step	LL_StepPriority	int*	A pointer to an integer indicating the priority of the step.
Step	LL_StepQueueSystemPriority	int*	A pointer to an integer indicating the adjusted system priority of the job step. This data is available from the LL_CM only.

ll_get_data subroutine

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Step	LL_StepRequestedReservationID	char**	A pointer to a string containing the step's requested reservation ID.
Step	LL_StepReservationID	char**	A pointer to a string containing the step's reservation ID.
Step	LL_StepRestart	int*	A pointer to an integer representing whether restart is specified as yes (default value) or no by the user in the job command file. <ul style="list-style-type: none"> • 1 indicates yes • 0 indicates no
Job	LL_StepRsetName	char**	A pointer to a character string containing the RSet name used by the step. If no RSet name was used, the value is NULL.
Step	LL_StepRssLimitHard	int*	A pointer to an integer indicating the RSS hard limit set by the user in the rss_limit keyword.
Step	LL_StepRssLimitHard64	int64_t*	A pointer to a 64-bit integer indicating the RSS hard limit set by the user in the rss_limit keyword.
Step	LL_StepRssLimitSoft	int*	A pointer to an integer indicating the RSS soft limit set by the user in the rss_limit keyword.
Step	LL_StepRssLimitSoft64	int64_t*	A pointer to a 64-bit integer indicating the RSS soft limit set by the user in the rss_limit keyword.
Step	LL_StepShell	char**	A pointer to a character string containing the shell name used by the executable.
Step	LL_StepSMTRequired	int*	A pointer to an integer indicating the required SMT state: 0 if SMT function is required, 1 if SMT function is not required.
Step	LL_StepStackLimitHard	int*	A pointer to an integer indicating the stack hard limit set by the user in the stack_limit keyword.
Step	LL_StepStackLimitHard64	int64_t*	A pointer to a 64-bit integer indicating the stack hard limit set by the user in the stack_limit keyword.
Step	LL_StepStackLimitSoft	int*	A pointer to an integer indicating the stack soft limit set by the user in the stack_limit keyword.
Step	LL_StepStackLimitSoft64	int64_t*	A pointer to a 64-bit integer indicating the stack soft limit set by the user in the stack_limit keyword.
Step	LL_StepStartCount	int*	A pointer to an integer indicating the number of times the step has been started. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Step	LL_StepStartDate	time_t*	A pointer to a time_t structure indicating the value the user specified in the startdate keyword. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepStarterIdrss64	int64_t*	A pointer to a 64-bit integer indicating the amount of unshared memory in the data segment of a process. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStarterInblock64	int64_t*	A pointer to a 64-bit integer indicating the number of times the file system performed input. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStarterIsrss64	int64_t*	A pointer to a 64-bit integer indicating the unshared stack size. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStarterIxrss64	int64_t*	A pointer to a 64-bit integer indicating the amount of memory used by the text segment that was also shared among other processes. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStarterMajflt64	int64_t*	A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStarterMaxrss64	int64_t*	A pointer to a 64-bit integer indicating the maximum resident set size utilized. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStarterMinflt64	int64_t*	A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStarterMsgrcv64	int64_t*	A pointer to a 64-bit integer indicating the number of IPC messages received. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStarterMsgsnd64	int64_t*	A pointer to a 64-bit integer indicating the number of IPC messages sent. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStarterNivcsw64	int64_t*	A pointer to a 64-bit integer indicating the number of involuntary context switches. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.

ll_get_data subroutine

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Step	LL_StepStarterNsignals64	int64_t*	A pointer to a 64-bit integer indicating the number of signals delivered. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStarterNswap64	int64_t*	A pointer to a 64-bit integer indicating the number of times swapped out. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStarterNvcsw64	int64_t*	A pointer to a 64-bit integer indicating the number of context switches due to voluntarily giving up processor. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStarterOublock64	int64_t*	A pointer to a 64-bit integer indicating the number of times the file system performed output. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStarterSystemTime64	int64_t*	A pointer to a 64-bit integer indicating the CPU system time. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStarterUserTime64	int64_t*	A pointer to a 64-bit integer indicating the CPU user time. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStartTime	time_t*	A pointer to a time_t structure indicating the time at which the starter process for the job started. Data is available from the LL_SCHEDD and LL_HISTORY_FILE.
Step	LL_StepState	int*	A pointer to an integer indicating the state of the Step (Idle, Pending, Starting, and so on). The value returned is in the StepState enum. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepStepIdrss64	int64_t*	A pointer to a 64-bit integer indicating the amount of unshared memory in the data segment of a process. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStepInblock64	int64_t*	A pointer to a 64-bit integer indicating the number of times the file system performed input. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStepIsrss64	int64_t*	A pointer to a 64-bit integer indicating the unshared stack size. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Step	LL_StepStepIxrss64	int64_t*	A pointer to a 64-bit integer indicating the amount of memory used by the text segment that was also shared among other processes. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStepMajflt64	int64_t*	A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStepMaxrss64	int64_t*	A pointer to a 64-bit integer indicating the maximum resident set size utilized. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStepMinflt64	int64_t*	A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStepMsgrcv64	int64_t*	A pointer to a 64-bit integer indicating the number of IPC messages received. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStepMsgsnd64	int64_t*	A pointer to a 64-bit integer indicating the number of IPC messages sent. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStepNivcsw64	int64_t*	A pointer to a 64-bit integer indicating the number of involuntary context switches. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStepNsignals64	int64_t*	A pointer to a 64-bit integer indicating the number of signals delivered. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStepNswap64	int64_t*	A pointer to a 64-bit integer indicating the number of times swapped out. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStepNvcsw64	int64_t*	A pointer to a 64-bit integer indicating the number of context switches due to voluntarily giving up processor. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStepOublock64	int64_t*	A pointer to a 64-bit integer indicating the number of times the file system performed output. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.

ll_get_data subroutine

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Step	LL_StepStepSystemTime64	int64_t*	A pointer to a 64-bit integer indicating the CPU system time. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepStepUserTime64	int64_t*	A pointer to a 64-bit integer indicating the CPU user time. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepSystemPriority	int*	A pointer to an integer indicating the overall system priority of the job step. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepTaskGeometry	char**	A pointer to a string containing the values specified in the task_geometry statement by the user in the job command file. The syntax is the same as specified in the statement , {(task id, task id, ...) (task id, task id, ...) ...}. If unspecified, a null string is returned. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepTaskInstanceCount	int*	A pointer to an integer indicating the number of task instances in the step. Data is available from the LL_SCHEDD, LL_STARTD, and LL_HISTORY_FILE.
Step	LL_StepTasksPerNodeRequested	int*	A pointer to an integer representing the tasks per node specified by the user in the job command file. If unspecified, the integer will contain a 0. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepTotalNodesRequested	char**	A pointer to a string containing the values specified by the user in the job command file node statement. The syntax is the same as specified in the statement, [min],[max], where min contains the minimum number of nodes requested and max contains the maximum nodes requested. If unspecified, a null string is returned. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepTotalTasksRequested	int*	A pointer to an integer representing the total tasks specified by the user in the job command file. If unspecified, the integer will contain a 0. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
Step	LL_StepUserSystemPriority	int*	A pointer to an integer indicating the user system priority of the job step. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Step	LL_StepWallClockLimitHard	int*	A pointer to an integer indicating the wall clock hard limit set by the user in the wall_clock_limit keyword.
Step	LL_StepWallClockLimitHard64	int64_t*	A pointer to a 64-bit integer indicating the wall clock hard limit set by the user in the wall_clock_limit keyword.
Step	LL_StepWallClockLimitSoft	int*	A pointer to an integer indicating the wall clock soft limit set by the user in the wall_clock_limit keyword.
Step	LL_StepWallClockLimitSoft64	int64_t*	A pointer to a 64-bit integer indicating the wall clock soft limit set by the user in the wall_clock_limit keyword.
Step	LL_StepWallClockUsed	int*	A pointer to an integer that is the number of seconds of elapsed time for this step. This specification is valid only when SCHEDULER_TYPE = API , otherwise a value of zero is returned. The value does not include any time that a job step has spent in a preempted by suspend state or doing a checkpoint. Data is available from the LL_STARTD only.
Task	LL_TaskExecutable	char**	A pointer to a string containing the name of the executable.
Task	LL_TaskExecutableArguments	char**	A pointer to a string containing the arguments passed by the user in the executable.
Task	LL_TaskGetFirstResourceRequirement	LL_element (ResourceReq)	A pointer to the element associated with the first resource requirement.
Task	LL_TaskGetFirstTaskInstance	LL_element* (TaskInstance)	A pointer to the element associated with the first task instance.
Task	LL_TaskGetNextResourceRequirement	LL_element* (ResourceReq)	A pointer to the element associated with the next resource requirement.
Task	LL_TaskGetNextTaskInstance	LL_element* (TaskInstance)	A pointer to the element associated with the next task instance.
Task	LL_TaskIsMaster	int*	A pointer to an integer, where 1 indicates master task.
Task	LL_TaskTaskInstanceCount	int*	A pointer to an integer indicating the number of task instances.
TaskInstance	LL_TaskInstanceAdapterCount	int*	A pointer to the integer indicating the number of adapters. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
TaskInstance	LL_TaskInstanceCpuList	int*	A pointer to the integer indicating the number of CPUs used by a given task instance object. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.

ll_get_data subroutine

Table 91. JOBS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
TaskInstance	LL_TaskInstanceGetFirstAdapter	LL_element* (Adapter)	A pointer to the element associated with the first adapter. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
TaskInstance	LL_TaskInstanceGetFirstAdapterUsage	LL_element* (AdapterUsage)	A pointer to the element associated with the first adapter usage. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
TaskInstance	LL_TaskInstanceGetNextAdapter	LL_element* (Adapter)	A pointer to the element associated with the next adapter. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
TaskInstance	LL_TaskInstanceGetNextAdapterUsage	LL_element* (AdapterUsage)	A pointer to the element associated with the next adapter usage. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
TaskInstance	LL_TaskInstanceMachineAddress	char**	A pointer to a string containing the IP address of the machine assigned to a task. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.
TaskInstance	LL_TaskInstanceMachineName	char**	A pointer to the string indicating the machine assigned to a task.
TaskInstance	LL_TaskInstanceTaskID	int*	A pointer to the integer indicating the task ID. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE.

See “Understanding the Machine object model” on page 527 for more information on the Machine object model.

Table 92. MACHINES specifications for ll_get_data subroutine

Object	Specification	type of resulting data parameter	Description
Adapter	LL_AdapterAvailWindowCount	int*	A pointer to an integer indicating the number of adapter windows not in use.
Adapter	LL_AdapterCommInterface	int*	Contains the adapter’s communication interface.
Adapter	LL_AdapterInterfaceAddress	char**	A pointer to a string containing the adapter’s interface IP address.
Adapter	LL_AdapterInterfaceNetmask	char**	A pointer to a string containing the netmask of an adapter.
Adapter	LL_AdapterMaxWindowSize64	uint64_t*	A pointer to an unsigned 64-bit integer indicating the maximum allocatable adapter window memory.
Adapter	LL_AdapterMCMId	int*	A pointer to an integer indicating the MCM ID for the adapter.

Table 92. MACHINES specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Adapter	LL_AdapterMemory64	uint64_t*	A pointer to an unsigned 64-bit integer indicating the amount of total adapter memory.
Adapter	LL_AdapterMinWindowSize64	int*	A pointer to the integer indicating the minimum allocatable adapter window memory.
Adapter	LL_AdapterName	char**	A pointer to a string containing the adapter name.
Adapter	LL_AdapterRcxtBlocks	int*	A pointer to the integer indicating the number of rCxt blocks available on an adapter.
Adapter	LL_AdapterTotalWindowCount	int*	A pointer to the integer indicating the number of windows on the adapter.
Adapter	LL_AdapterWindowList	int**	A pointer to an array indicating window numbers for the adapter. LL_AdapterTotalWindowCount indicates the size of this array. If the adapter has no windows, LL_AdapterTotalWindowCount is zero and LL_AdapterWindowList is null.
Machine	LL_MachineAdapterList	char***	A pointer to an array containing a list of the types of adapters associated with the machine. The array ends with a NULL string.
Machine	LL_MachineArchitecture	char**	A pointer to a string containing the machine architecture. The string may result in "???" if a query is made before the associated records are updated with their actual values by the appropriate startd daemons.
Machine	LL_MachineAvailableClassList	char***	A pointer to an array containing the currently available job classes defined on the machine. The array ends with a NULL string.
Machine	LL_MachineConfigTimeStamp	int*	A pointer to an integer containing the date and time value of the last configuration or reconfiguration.
Machine	LL_MachineConfiguredClassList	char***	A pointer to an array containing the initiators on the machine. The array ends with a NULL string.
Machine	LL_MachineContinueExpr	char**	A pointer to a string containing the machine's continue control expression.
Machine	LL_MachineCpuList	int*	A pointer to an integer containing the list of CPUs on the machine.
Machine	LL_MachineCPUs	int*	A pointer to an integer containing the number of CPUs on the machine.
Machine	LL_MachineDisk	int*	A pointer to an integer indicating the disk space in KBs in the machine's execute directory.

ll_get_data subroutine

Table 92. MACHINES specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Machine	LL_MachineDisk64	int64_t*	A pointer to a 64-bit integer indicating the disk space in KBs in the machine's execute directory.
Machine	LL_MachineDrainClassList	char***	A pointer to an array containing the drain class list on the machine. The array ends with a NULL string.
Machine	LL_MachineDrainingClassList	char***	A pointer to an array containing the draining class list on the machine. The array ends with a NULL string.
Machine	LL_MachineFeatureList	char***	A pointer to an array containing the features defined on the machine. The array ends with a NULL string.
Machine	LL_MachineFreeRealMemory	int*	A pointer to an integer indicating the amount of free real memory in MBs on the machine.
Machine	LL_MachineFreeRealMemory64	int64_t*	A pointer to a 64-bit integer indicating the amount of free real memory in MBs on the machine.
Machine	LL_MachineGetFirstAdapter	LL_element* (Adapter)	A pointer to the element associated with the machine's first adapter.
Machine	LL_MachineGetFirstMCM	LL_element* (MCM)	A pointer to the element associated with the machine's first MCM.
Machine	LL_MachineGetFirstResource	LL_element* (Resource)	A pointer to the element associated with the machine's first resource.
Machine	LL_MachineGetNextAdapter	LL_element* (Adapter)	A pointer to the element associated with the machine's next adapter.
Machine	LL_MachineGetNextMCM	LL_element* (MCM)	A pointer to the element associated with the machine's next MCM.
Machine	LL_MachineGetNextResource	LL_element* (Resource)	A pointer to the element associated with the machine's next resource.
Machine	LL_MachineKbdddIdle	int*	A pointer to an integer indicating the number of seconds since the kbddd daemon detected keyboard mouse activity.
Machine	LL_MachineKillExpr	char**	A pointer to a string containing the machine's kill control expression.
Machine	LL_MachineLargePageCount64	int64_t*	A pointer to a 64-bit integer indicating the number of Large Pages defined on the machine.
Machine	LL_MachineLargePageFree64	int64_t*	A pointer to a 64-bit integer indicating the number of Large Pages free.
Machine	LL_MachineLargePageSize64	int64_t*	A pointer to a 64-bit integer indicating the size of the machine's Large Page.
Machine	LL_MachineLoadAverage	double*	A pointer to a double containing the load average on the machine.
Machine	LL_MachineMachineMode	char**	A pointer to a string containing the configured machine mode.

Table 92. MACHINES specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Machine	LL_MachineMaxTasks	int*	A pointer to an integer indicating the maximum number of tasks this machine can run at one time.
Machine	LL_MachineName	char**	A pointer to a string containing the machine name.
Machine	LL_MachineOperatingSystem	char**	A pointer to a string containing the operating system on the machine. The string may result in "???" if a query is made before the associated records are updated with their actual values by the appropriate startd daemons.
Machine	LL_MachinePagesFreed	int*	A pointer to an integer indicating the number of pages freed per second by the page replacement algorithm.
Machine	LL_MachinePagesFreed64	int64_t*	A pointer to a 64-bit integer indicating the number of pages freed per second by the page replacement algorithm.
Machine	LL_MachinePagesPagedIn	int*	A pointer to an integer indicating the number of pages paged in per second from paging space.
Machine	LL_MachinePagesPagedIn64	int64_t*	A pointer to a 64-bit integer indicating the number of pages paged in per second from paging space.
Machine	LL_MachinePagesPagedOut	int*	A pointer to an integer indicating the number of pages paged out per second to paging space.
Machine	LL_MachinePagesPagedOut64	int64_t*	A pointer to a 64-bit integer indicating the number of pages paged out per second to paging space.
Machine	LL_MachinePagesScanned	int*	A pointer to an integer indicating the number of pages scanned per second by the page replacement algorithm.
Machine	LL_MachinePagesScanned64	int64_t*	A pointer to a 64-bit integer indicating the number of pages scanned per second by the page replacement algorithm.
Machine	LL_MachinePoolList	int**	A pointer to an array indicating the pool numbers to which this machine belongs. The size of the array can be determined by using LL_MachinePoolListSize.
Machine	LL_MachinePoolListSize	int*	A pointer to an integer indicating the number of pools configured for the machine.
Machine	LL_MachinePrestartedStarters	int*	A pointer to an integer indicating the number of prestarted Starters on a machine.
Machine	LL_MachineRealMemory	int*	A pointer to an integer indicating the physical memory in MBs on the machine.

ll_get_data subroutine

Table 92. MACHINES specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Machine	LL_MachineRealMemory64	int64_t*	A pointer to a 64-bit integer indicating the physical memory in MBs on the machine.
Machine	LL_MachineReservationList	char***	A pointer to an array containing the list of reservation IDs using this machine. The array ends with a NULL string.
Machine	LL_MachineReservationPermitted	int*	A pointer to an integer to determine if this machine can be reserved.
Machine	LL_MachineScheddRunningJobs	int*	A pointer to an integer indicating a list of the running jobs assigned to Schedd.
Machine	LL_MachineScheddState	int*	A pointer to an integer indicating the machine's Schedd state.
Machine	LL_MachineScheddTotalJobs	int*	A pointer to an integer indicating the total number of jobs assigned to the Schedd.
Machine	LL_MachineSpeed	double*	A pointer to a double containing the configured speed of the machine.
Machine	LL_MachineStartdRunningJobs	int*	A pointer to an integer containing the number of running jobs known by the startd daemon.
Machine	LL_MachineStartdState	char**	A pointer to a string containing the state of the startd daemon.
Machine	LL_MachineStartExpr	char**	A pointer to a string containing the machine's start control expression.
Machine	LL_MachineStepList	char***	A pointer to an array containing the steps running on the machine. The array ends with a NULL string.
Machine	LL_MachineSuspendExpr	char**	A pointer to a string containing the machine's suspend control expression.
Machine	LL_MachineTimeStamp	time_t*	A pointer to a time_t structure indicating the time the machine last reported to the negotiator.
Machine	LL_MachineUsedCpuList	int*	A pointer to an integer containing the list of CPUs being used on the machine.
Machine	LL_MachineVacateExpr	char**	A pointer to a string containing the machine's vacate control expression.
Machine	LL_MachineVirtualMemory	int*	A pointer to an integer indicating the free swap space in KBs on the machine.
Machine	LL_MachineVirtualMemory64	int64_t*	A pointer to a 64-bit integer indicating the free swap space in KBs on the machine.
MCM	LL_MCMCPUList	int**	A pointer to an array indicating the list of CPUs on the MCM.
MCM	LL_MCMCPUs	int*	A pointer to an integer containing the number of CPUs within the MCM.
MCM	LL_MCMID	int*	A pointer to an integer containing the ID of the MCM.

Table 92. MACHINES specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Resource	LL_ResourceAvailableValue	int*	A pointer to an integer indicating the value of available resources.
Resource	LL_ResourceAvailableValue64	int64_t*	A pointer to a 64-bit integer indicating the value of available resources.
Resource	LL_ResourceInitialValue	int*	A pointer to an integer indicating the initial resource value.
Resource	LL_ResourceInitialValue64	int64_t*	A pointer to a 64-bit integer indicating the initial resource value.
Resource	LL_ResourceName	char**	A pointer to a string containing the resource name.

See “Understanding the MCluster object model” on page 528 for more information on the MCluster object model.

Table 93. MCLUSTERS specifications for ll_get_data subroutine

Object	Specification	type of resulting data parameter	Description
MCluster	LL_MClusterExcludeClasses	char***	A pointer to an array containing a list of exclude classes.
MCluster	LL_MClusterExcludeGroups	char***	A pointer to an array containing a list of exclude groups.
MCluster	LL_MClusterExcludeUsers	char***	A pointer to an array containing a list of exclude users.
MCluster	LL_MClusterInboundHosts	char***	A pointer to an array containing a list of inbound machines.
MCluster	LL_MClusterInboundScheddPort	int*	A pointer to an integer containing the cluster Schedd port number.
MCluster	LL_MClusterIncludeClasses	char***	A pointer to an array containing a list of include classes.
MCluster	LL_MClusterIncludeGroups	char***	A pointer to an array containing a list of include groups.
MCluster	LL_MClusterIncludeUsers	char***	A pointer to an array containing a list of include users.
MCluster	LL_MClusterLocal	int*	A pointer to an integer. If the integer contains the value 1, the cluster is local. If the integer contains the value 0, the cluster is remote.
MCluster	LL_MClusterMulticlusterSecurity	char**	A pointer to a string containing the security method for the multicluster.
MCluster	LL_MClusterName	char*	A pointer to a string containing the cluster name.
MCluster	LL_MClusterOutboundHosts	char***	A pointer to an array containing a list of outbound machines.
MCluster	LL_MClusterSecureScheddPort	int*	A pointer to an integer containing the secure Schedd port for the cluster.

ll_get_data subroutine

Table 93. MCLUSTERS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
MCluster	LL_MClusterSslCipherList	char**	A pointer to a string containing the list of cipher for SSL.

See “Understanding the Reservations object model” on page 528 for more information on the Reservations object model.

Table 94. RESERVATIONS specifications for ll_get_data subroutine

Object	Specification	type of resulting data parameter	Description
Reservation	LL_ReservationBgBPs	char***	A pointer to an array containing the Blue Gene base partitions reserved by the reservation. For a partially reserved base partition, the node cards reserved will be listed in parenthesis after the BP name.
Reservation	LL_ReservationBgCNodes	int*	A pointer to an integer indicating the number of Blue Gene C-nodes that are reserved.
Reservation	LL_ReservationBgConnection	char**	A pointer to a string containing the Blue Gene connection.
Reservation	LL_ReservationBgShape	int**	A pointer to an array indicating the Blue Gene shape.
Reservation	LL_ReservationCreateTime	time_t*	A pointer to the time_t structure indicating the creation time of the reservation.
Reservation	LL_ReservationDuration	int*	A pointer to an integer containing the reservation duration in the unit of minutes.
Reservation	LL_ReservationGroup	char**	A pointer to a string containing the LoadLeveler group that owns the reservation.
Reservation	LL_ReservationGroups	char***	A pointer to an array containing the LoadLeveler groups whose users may run jobs in the reservation. The array ends with a NULL string.
Reservation	LL_ReservationID	char**	A pointer to a string containing the ID of the reservation.
Reservation	LL_ReservationJobs	char***	A pointer to an array containing the job steps bound to the reservation. The array ends with a NULL string.
Reservation	LL_ReservationMachines	char***	A pointer to an array containing the machines reserved by the reservation. The array ends with a NULL string.
Reservation	LL_ReservationModeRemoveOnIdle	int*	A pointer to an integer indicating that RESERVATION_REMOVE_ON_IDLE mode is on if 1; off if 0.

Table 94. RESERVATIONS specifications for ll_get_data subroutine (continued)

Object	Specification	type of resulting data parameter	Description
Reservation	LL_ReservationModeShared	int*	A pointer to an integer indicating that RESERVATION_SHARED mode is on if 1; off if 0.
Reservation	LL_ReservationModifiedBy	char**	A pointer to a string containing the user ID who last modified the reservation.
Reservation	LL_ReservationModifyTime	time_t*	A pointer to the time_t structure indicating the last modification time.
Reservation	LL_ReservationOwner	char**	A pointer to a string containing the owner of the reservation.
Reservation	LL_ReservationStartTime	time_t*	A pointer to the time_t structure indicating the beginning time of the reservation.
Reservation	LL_ReservationStatus	int*	A pointer to an integer containing the state of the reservation that takes one of the Reservation_State_t values in llapi.h.
Reservation	LL_ReservationUsers	char***	A pointer to an array containing the users who may run jobs in the reservation. The array ends with a NULL string.

See “Understanding the Wlmstat object model” on page 529 for more information on the Wlmstat object model.

Table 95. WLMSTAT specifications for ll_get_data subroutine

Object	Specification	type of resulting data parameter	Description
WlmStat	LL_WlmStatCpuSnapshotUsage	int*	A pointer to CPU usage obtained from the AIX Workload Manager.
WlmStat	LL_WlmStatCpuTotalUsage	int64_t*	A pointer to total CPU usage obtained from the AIX Workload Manager.
WlmStat	LL_WlmStatMemoryHighWater	int64_t*	A pointer to real memory high water mark obtained from the AIX Workload Manager.
WlmStat	LL_WlmStatMemorySnapshotUsage	int*	A pointer to real memory usage obtained from the Workload Manager.

ll_get_objs subroutine

Purpose

ll_get_objs – Sends a query request to the daemon you specify along with the request data you specified in the **ll_set_request** subroutine. **ll_get_objs** receives a list of objects matching the request.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
LL_element * ll_get_objs (LL_element *query_element ,LL_Daemon query_daemon,
                        char *hostname,int * number_of_objs,
                        int * error_code);
```

Parameters

query_element

Is a pointer to the **LL_element** returned by the **ll_query** function.

query_daemon

Specifies the LoadLeveler daemon you want to query or whether you want to query job information stored in a history file. The enum **LL_Daemon** is defined in llapi.h as:

```
enum LL_Daemon {LL_STARTD, LL_SCHEDD, LL_CM, LL_MASTER, LL_STARTER,
               LL_HISTORY_FILE};
```

Table 96 indicates which daemons respond to which query flags:

Table 96. *query_daemon* summary

When query_type (in ll_query) is:	query_flags (in ll_set_request) can be:	Responded to by these daemons:
BLUE_GENE	QUERY_ALL	negotiator (LL_CM)
	QUERY_BG_BASE_PARTITION	negotiator (LL_CM)
	QUERY_BG_PARTITION	negotiator (LL_CM)
CLASSES	QUERY_ALL	negotiator (LL_CM)
	QUERY_CLASS	negotiator (LL_CM)
CLUSTER	QUERY_ALL	negotiator (LL_CM)
FAIRSHARE	QUERY_ALL	negotiator (LL_CM)
	QUERY_GROUP	negotiator (LL_CM)
	QUERY_USER	negotiator (LL_CM)

Table 96. query_daemon summary (continued)

When query_type (in ll_query) is:	query_flags (in ll_set_request) can be:	Responded to by these daemons:
JOBS	QUERY_ALL	negotiator (LL_CM), Schedd (LL_SCHEDD), or history file (LL_HISTORY_FILE)
	QUERY_BG_JOB	negotiator (LL_CM)
	QUERY_CLASS	negotiator (LL_CM) or Schedd (LL_SCHEDD)
	QUERY_ENDDATE	history file (LL_HISTORY_FILE)
	QUERY_GROUP	negotiator (LL_CM)
	QUERY_HOST	negotiator (LL_CM)
	QUERY_JOBID	negotiator (LL_CM) or Schedd (LL_SCHEDD)
	QUERY_PROCID	startd (LL_STARTD)
	QUERY_RESERVATION_ID	negotiator (LL_CM) or Schedd (LL_SCHEDD)
	QUERY_STARTDATE	history file (LL_HISTORY_FILE)
	QUERY_STEPID	negotiator (LL_CM), Schedd (LL_SCHEDD), or startd (LL_STARTD)
MACHINES	QUERY_ALL	negotiator (LL_CM)
	QUERY_HOST	negotiator (LL_CM)
MCLUSTERS	QUERY_ALL	Schedd (LL_SCHEDD)
RESERVATIONS	QUERY_ALL	negotiator (LL_CM)
	QUERY_BG_BASE_PARTITION	negotiator (LL_CM)
	QUERY_GROUP	negotiator (LL_CM)
	QUERY_HOST	negotiator (LL_CM)
	QUERY_RESERVATION_ID	negotiator (LL_CM)
	QUERY_USER	negotiator (LL_CM)
WLMSTAT	QUERY_STEPID	startd (LL_STARTD)

hostname

Specifies the *hostname* of the daemon or the history file name to be queried.

When *query_type* is **JOBS**, if *query_daemon* is:

LL_SCHEDD or LL_STARTD

The local machine is queried if the *hostname* is NULL.

LL_CM

The *hostname* is ignored.

LL_HISTORY_FILE

The *hostname* represents the history file to obtain data from. If the *hostname* is NULL, an error is returned.

When *query_type* is **MCLUSTER**, the *query_daemon* must be **LL_SCHEDD**. If you specify NULL for the *hostname*:

- The cluster specified by the **ll_cluster** API is the local cluster, a configured outbound Schedd daemon for the local cluster is queried.

ll_get_objs subroutine

- The cluster specified by the `ll_cluster` API is a remote cluster, a configured inbound Schedd daemon for the remote cluster is queried.

number_of_objs

Is a pointer to an integer representing the number of objects received from the daemon.

error_code

Is a pointer to an integer representing the error code issued when the function returns a NULL value. For more information, see "Error values."

Description

query_element, *query_daemon*, and *hostname* are the input fields for this subroutine. *number_of_objs* and *error_code* are output fields.

Each LoadLeveler daemon returns only the objects that it knows about.

Return Values

This subroutine returns a pointer to the first object in the list. You must use the `ll_next_obj` subroutine to access the next object in the list.

Error Values

This subroutine returns a NULL to indicate failure. The *error_code* parameter is set to one of the following:

- 1 *query_element* not valid.
- 2 *query_daemon* not valid.
- 3 Cannot resolve *hostname*.
- 4 Request type for specified daemon not valid.
- 5 System error.
- 6 No valid objects meet the request.
- 7 Configuration error.
- 9 Connection to daemon failed.
- 10 Error processing history file (LL_HISTORY_FILE query only).
- 11 History file must be specified in the hostname argument (LL_HISTORY_FILE query only).
- 12 Unable to access the history file (LL_HISTORY_FILE query only).

Related Information

Subroutines: `ll_cluster`, `ll_deallocate`, `ll_free_objs`, `ll_get_data`, `ll_next_obj`, `ll_query`, `ll_set_request`

ll_next_obj subroutine

Purpose

ll_next_obj – Returns the next object in the *query_element* list you specify.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
LL_element * ll_next_obj (LL_element *query_element);
```

Parameters

query_element

Is a pointer to the **LL_element** returned by the **ll_query** function.

Description

query_element is the input field for this subroutine.

Use this subroutine in conjunction with the **ll_get_objs** subroutine to “loop” through the list of objects queried.

Return Values

This subroutine returns a pointer to the next object in the list.

Error Values

NULL Indicates an error or the end of the list of objects.

Related Information

Subroutines: **ll_cluster**, **ll_deallocate**, **ll_free_objs**, **ll_get_data**, **ll_get_objs**, **ll_query**, **ll_set_request**

ll_query subroutine

Purpose

ll_query – Initializes the query object and defines the type of query you want to perform. The **LL_element** created and the corresponding data returned by this function is determined by the *query_type* you select.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
LL_element * ll_query (enum QueryType query_type);
```

Parameters

query_type

Can be:

- **BLUE_GENE** (to query information about the Blue Gene system)
- **CLASSES** (to query information about job classes)
- **CLUSTER** (to query cluster information)
- **FAIRSHARE** (to query fair share scheduling information)
- **JOBS** (to query job information)
- **MACHINES** (to query machine information)
- **MCLUSTERS** (to query multicluster objects)

Multicluster objects only exist when LoadLeveler has a multicluster configuration.

- **RESERVATIONS** (to query reservation information)
- **WLMSTAT** (to query AIX Workload Manager)

Description

query_type is the input field for this subroutine.

This subroutine is used in conjunction with other data access subroutines to query information about job and machine objects. You must call **ll_query** prior to using the other data access subroutines.

Return Values

This subroutine returns a pointer to an **LL_element** object. The pointer is used by subsequent data access subroutine calls.

Error Values

NULL The subroutine was unable to create the appropriate pointer.

Related Information

Subroutines: **ll_cluster**, **ll_deallocate**, **ll_free_objs**, **ll_get_data**, **ll_get_objs**, **ll_next_obj**, **ll_reset_request**, **ll_set_request**

ll_reset_request subroutine

Purpose

ll_reset_request – Resets the request data to NULL for the *query_element* you specify.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int ll_reset_request (LL_element *query_element);
```

Parameters

query_element

Is a pointer to the **LL_element** returned by the **ll_query** function.

Description

query_element is the input field for this subroutine.

This subroutine is used in conjunction with **ll_set_request** to change the data requested with the **ll_get_objs** subroutine.

Return Values

This subroutine returns a zero to indicate success.

Error Values

-1 The subroutine was unable to reset the appropriate data.

Related Information

Subroutines: **ll_deallocate**, **ll_free_objs**, **ll_get_data**, **ll_get_objs**, **ll_next_obj**, **ll_query**, **ll_set_request**

ll_set_request subroutine

Purpose

ll_set_request – Determines the data requested during a subsequent **ll_get_objs** call to query specific objects. You can filter your queries based on the *query_type*, *object_filter*, and *data_filter* you select.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int ll_set_request (LL_element *query_element, QueryFlags query_flags,
                  char **object_filter, DataFilter data_filter);
```

Parameters

query_element

Is a pointer to the **LL_element** returned by the **ll_query** subroutine.

query_flags

Table 97 provides information on *query_type* (in **ll_query**) and related *query_flags*:

Table 97. *query_flags* summary

When <i>query_type</i> (in ll_query) is:	<i>query_flags</i> can be:	Flag description:
BLUE_GENE	QUERY_ALL	Query Blue Gene base partitions, switches, and wires.
	QUERY_BG_BASE_PARTITION	Query Blue Gene base partitions, including any small partitions allocated on the base partition.
	QUERY_BG_PARTITION	Query partitions defined on the Blue Gene system.
CLASSES	QUERY_ALL	Query all classes.
	QUERY_CLASS	Query by LoadLeveler class.
CLUSTER	QUERY_ALL	Query cluster information from central manager.
FAIRSHARE	QUERY_ALL	Query returns all available information.
	QUERY_GROUP	Query by LoadLeveler group.
	QUERY_USER	Query by user ID.

Table 97. query_flags summary (continued)

When query_type (in ll_query) is:	query_flags can be:	Flag description:
JOBS	QUERY_ALL	Query all jobs.
	QUERY_BG_JOB	Query by bluegene type jobs only.
	QUERY_CLASS	Query by LoadLeveler class.
	QUERY_ENDDATE	Query by job end dates. History file query only.
	QUERY_GROUP	Query by LoadLeveler group.
	QUERY_HOST	Query by machine name.
	QUERY_JOBID	Query by job ID.
	QUERY_PROCID	Query by process ID of a task of a job step.
	QUERY_RESERVATION_ID	Query job steps bound to a particular reservation.
	QUERY_STARTDATE	Query by job start dates. History file query only.
	QUERY_STEPID	Query by step ID.
MACHINES	QUERY_ALL	Query all machines.
	QUERY_HOST	Query by machine names.
MCLUSTERS	QUERY_ALL	Query all multiclusters.
RESERVATIONS	QUERY_ALL	Query all reservations.
	QUERY_BG_BASE_PARTITION	Query by Blue Gene base partitions or all for all base partitions.
	QUERY_GROUP	Query by LoadLeveler group that owns the reservations.
	QUERY_HOST	Query by machine name.
	QUERY_RESERVATION_ID	Query by reservation ID.
RESERVATIONS	QUERY_USER	Query by user ID that owns the reservations.
	QUERY_STEPID	Query by step ID.

object_filter

Specifies search criteria. The value you specify for *object_filter* is related to the value you specify for *query_flags* as shown in Table 98:

Table 98. object_filter value related to the query flags value

If you specify:	Note:
QUERY_ALL	You do not need an <i>object_filter</i> .
QUERY_BG_BASE_PARTITION	The <i>object_filter</i> must contain a list of base partition IDs. For all base partitions, the first entry in the list must contain the string "all".
QUERY_BG_PARTITION	The <i>object_filter</i> must contain a list of partition IDs. For all partitions, the first entry in the list must contain the string "all".
QUERY_CLASS	The <i>object_filter</i> must contain a list of LoadLeveler class names.
QUERY_GROUP	The <i>object_filter</i> must contain a list of LoadLeveler group names.

Il_set_request subroutine

Table 98. *object_filter* value related to the query flags value (continued)

If you specify:	Note:
QUERY_HOST	The <i>object_filter</i> must contain a list of LoadLeveler machine names. When the query type is JOBS , the machine names must be the names of machines to which the jobs are submitted.
QUERY_JOBID	The <i>object_filter</i> must contain a list of job IDs (in the form <i>host.jobid</i>).
QUERY_PROCID	The <i>object_filter</i> must contain a list with a single process ID of a task of a job step.
QUERY_RESERVATION_ID	The <i>object_filter</i> must contain a list of reservation IDs.
QUERY_STARTDATE or QUERY_ENDDATE	The <i>object_filter</i> must contain a list of two start dates or two end dates having the format <i>MM/DD/YYYY</i> .
QUERY_STEPID	The <i>object_filter</i> must contain a list of step IDs (in the form <i>host.jobid.stepid</i>).
QUERY_USER	The <i>object_filter</i> must contain a list of user IDs.

The last entry in the *object_filter* array must be NULL.

data_filter

Filters the data returned from the object you query. The value you specify for *data_filter* is related to the value you specify for *query_type*. The *data_filter* must be **ALL_DATA** (the default) when:

- You query a history file for job information
- You specify **JOBS** and *query_flags* **QUERY_PROCID**
- You specify **BLUE_GENE**, **CLASSES**, **CLUSTER**, **MACHINES**, **MCLUSTERS**, **RESERVATIONS**, or **WLMSTAT**

Description

query_element, *query_flags*, *object_filter*, and *data_filter* are the input fields for this subroutine.

The **QUERY_PROCID** flag should not be used in combination with any other *query_flags*.

Do not use the **QUERY_BG_BASE_PARTITION** flag in combination with the **QUERY_HOST** flag.

You can request certain combinations of object filters by calling **Il_set_request** more than once. When you do this, the query flags you specify are or-ed together. The following are valid combinations of object filters:

- **QUERY_JOBID** and **QUERY_STEPID**: the result is the union of both queries and any other query flags (such as, **QUERY_HOST**) will be ignored
- **QUERY_STARTDATE** and **QUERY_ENDDATE**: the result is the intersection of both queries
- **QUERY_HOST**, **QUERY_USER**, **QUERY_GROUP**, **QUERY_CLASS**, and **QUERY_RESERVATION_ID**: the result is the intersections of all of the queries
- When the *query_type* is **RESERVATIONS**, **QUERY_RESERVATION_ID** takes precedence and any other query flags are ignored (with the exception of

| **QUERY_ALL**, which always replaces any other query flags). **QUERY_HOST** and
| **QUERY_BG_BASE_PARTITION** are mutually exclusive and setting one will
| clear the other.

| To query jobs owned by certain users and on specific machines, issue
| **ll_set_request** first with **QUERY_USER** and the appropriate user IDs, and then
| issue it again with **QUERY_HOST** and the appropriate host names.

For example, suppose you issue **ll_set_request** with a user ID list of anton and meg, and then issue it again with a host list of k10n10 and k10n11. The objects returned are all of the jobs on k10n10 and k10n11 which belong to anton or meg.

Note that if you use two consecutive calls with the same flag, the second call will replace the previous call.

Also, you should not use the **QUERY_ALL** flag in combination with any other flag, since **QUERY_ALL** will replace any existing requests.

For history file queries, *query_flags* is restricted to the following: **QUERY_ALL**, **QUERY_STARTDATE**, **QUERY_ENDDATE**.

Return Values

This subroutine returns a zero to indicate success.

Error Values

- 1 You specified a *query_element* that is not valid.
- 2 You specified a *query_flag* that is not valid.
- 3 You specified an *object_filter* that is not valid.
- 4 You specified a *data_filter* that is not valid.
- 5 A system error occurred.

Related Information

Subroutines: **ll_cluster**, **ll_deallocate**, **ll_free_objs**, **ll_get_data**, **ll_get_objs**, **ll_next_obj**, **ll_query**, **ll_reset_request**

Examples of using the data access API

These examples are provided in the `samples/lldata_access` subdirectory of the release directory (usually `/usr/lpp/LoadL/full`).

Example 1: The following example shows how LoadLeveler's data access API can be used to obtain machine, job, and cluster information. The program consists of three steps:

1. Getting information about selected hosts in the LoadLeveler cluster
2. Getting information about jobs of selected classes
3. Getting floating consumable resource information in the LoadLeveler cluster

```
#include <stdio.h>
#include "llapi.h"

main(int argc, char *argv[])
{
    LL_element *queryObject, *machine, *resource, *cluster;
    LL_element *job, *step, *node, *task, *credential, *resource_req;
    int rc, obj_count, err_code, value;
    double load_avg;
    enum StepState step_state;
    char **host_list, **class_list;
    char *name, *res_name, *step_id, *job_class, *node_req;
    char *task_exec, *ex_args, *startd_state;

    /* Step 1: Display information of selected machines in the LL cluster */

    /* Initialize the query: Machine query */
    queryObject = ll_query(MACHINES);
    if (!queryObject) {
        printf("Query MACHINES: ll_query() returns NULL.\n"); exit(1);
    }

    /* Set query parameters: query specific machines by name */
    host_list = (char **)malloc(3*sizeof(char *));
    host_list[0] = "c163n12.ppd.pok.ibm.com";
    host_list[1] = "c163n11.ppd.pok.ibm.com";
    host_list[2] = NULL;
    rc = ll_set_request(queryObject, QUERY_HOST, host_list, ALL_DATA);
    if (rc) {
        printf("Query MACHINES: ll_set_request() return code is non-zero.\n"); exit(1);
    }

    /* Get the machine objects from the LoadL_negotiator (central manager) daemon */
    machine = ll_get_objs(queryObject, LL_CM, NULL, &obj_count, &err_code);
    if (machine == NULL) {
        printf("Query MACHINES: ll_get_objs() returns NULL. Error code = %d\n", err_code);
    }
    printf("Number of machines objects returned = %d\n", obj_count);

    /* Process the machine objects */
    while(machine) {
        rc = ll_get_data(machine, LL_MachineName, &name);
        if (!rc) {
            printf("Machine name: %s -----\n", name); free(name);
        }
        rc = ll_get_data(machine, LL_MachineStartdState, &startd_state);
        if (rc) {
            printf("Query MACHINES: ll_get_data() return code is non-zero.\n"); exit(1);
        }
    }
}
```

Figure 53. Obtaining machine, job, and cluster information with the data access API (Part 1 of 4)

```

printf("Startd State: %s\n", startd_state);
if (strcmp(startd_state, "Down") != 0) {
    rc = ll_get_data(machine, LL_MachineRealMemory, &value);
    if (!rc) printf("Total Real Memory: %d\n", value);
    rc = ll_get_data(machine, LL_MachineVirtualMemory, &value);
    if (!rc) printf("Free Swap Space: %d\n", value);
    rc = ll_get_data(machine, LL_MachineLoadAverage, &load_avg);
    if (!rc) printf("Load Average: %f\n", load_avg);
}
free(startd_state);
/* Consumable Resources associated with this machine */
resource = NULL;
ll_get_data(machine, LL_MachineGetFirstResource, &resource);
while(resource) {
    rc = ll_get_data(resource, LL_ResourceName, &res_name);
    if (!rc) {printf("Resource Name = %s\n", res_name); free (res_name);}
    rc = ll_get_data(resource, LL_ResourceInitialValue, &value);
    if (!rc) printf("    Total: %d\n", value);
    rc = ll_get_data(resource, LL_ResourceAvailableValue, &value);
    if (!rc) printf("    Available: %d\n", value);
    resource = NULL;
    ll_get_data(machine, LL_MachineGetNextResource, &resource);
}
machine = ll_next_obj(queryObject);
}

/* Free objects obtained from Negotiator */
ll_free_objs(queryObject);
/* Free query element */
ll_deallocate(queryObject);

/* Step 2: Display information of selected jobs */

/* Initialize the query: Job query */
queryObject = ll_query(JOBS);
if (!queryObject) {
    printf("Query JOBS: ll_query() returns NULL.\n");
    exit(1);
}

/* Query all class "Parallel" and "No_Class" jobs submitted to c163n11, c163n12 */
class_list = (char **)malloc(3*sizeof(char *));
class_list[0] = "Parallel";
class_list[1] = "No_Class";
class_list[2] = NULL;
rc = ll_set_request(queryObject, QUERY_HOST, host_list, ALL_DATA);
if (rc) {printf("Query JOBS: ll_set_request() return code is non-zero.\n"); exit(1);}
rc = ll_set_request(queryObject, QUERY_CLASS, class_list, ALL_DATA);
if (rc) {printf("Query JOBS: ll_set_request() return code is non-zero.\n"); exit(1);}

/* Get the requested job objects from the Central Manager */
job = ll_get_objs(queryObject, LL_CM, NULL, &obj_count, &err_code);
if (job == NULL) {
    printf("Query JOBS: ll_get_objs() returns NULL. Error code = %d\n", err_code);
}
printf("Number of job objects returned = %d\n", obj_count);

/* Process the job objects and display selected information of each job step.

```

Figure 53. Obtaining machine, job, and cluster information with the data access API (Part 2 of 4)

Examples of using the data access API

```
*
* Notes:
* 1. Since LL_element is defined as "void" in llapi.h, when using
* ll_get_data it is important that a valid "specification"
* parameter be used for a given "element" argument.
* 2. Checking of return code is not always made in the following
* loop to minimize the length of the listing.
*/

while(job) {
    rc = ll_get_data(job, LL_JobName, &name);
    if (!rc) {printf("Job name: %s\n", name); free(name);}

    rc = ll_get_data(job, LL_JobCredential, &credential);
    if (!rc) {
        rc = ll_get_data(credential, LL_CredentialUserName, &name);
        if (!rc) {printf("Job owner: %s\n", name); free(name);}
        rc = ll_get_data(credential, LL_CredentialGroupName, &name);
        if (!rc) {printf("Unix Group: %s\n", name); free(name);}
    }
    step = NULL;
    ll_get_data(job, LL_JobGetFirstStep, &step);
    while(step) {
        rc = ll_get_data(step, LL_StepID, &step_id);
        if (!rc) {printf(" Step ID: %s\n", step_id); free(step_id);}
        rc = ll_get_data(step, LL_StepJobClass, &job_class);
        if (!rc) {printf(" Step Job Class: %s\n", job_class); free(job_class);}
        rc = ll_get_data(step, LL_StepState, &step_state);
        if (!rc) {
            if (step_state == STATE_RUNNING) {
                printf(" Step Status: Running\n");
                printf(" Allocated Hosts:\n");
                machine = NULL;
                ll_get_data(step, LL_StepGetFirstMachine, &machine);
                while(machine) {
                    rc = ll_get_data(machine, LL_MachineName, &name);
                    if (!rc) { printf(" %s\n", name); free(name); }
                    machine = NULL;
                    ll_get_data(step, LL_StepGetNextMachine, &machine);
                }
            }
            else {
                printf(" Step Status: Not Running\n");
            }
        }
    }
    node = NULL;
    ll_get_data(step, LL_StepGetFirstNode, &node);
    while(node) {
        rc = ll_get_data(node, LL_NodeRequirements, &node_req);
        if (!rc) {printf(" Node Requirements: %s\n", node_req); free(node_req);}
        task = NULL;
        ll_get_data(node, LL_NodeGetFirstTask, &task);
        while(task) {
```

Figure 53. Obtaining machine, job, and cluster information with the data access API (Part 3 of 4)

```

rc = ll_get_data(task, LL_TaskExecutable, &task_exec);
if (!rc) {printf("    Task Executable: %s\n", task_exec); free(task_exec);}
rc = ll_get_data(task, LL_TaskExecutableArguments, &ex_args);
if (!rc) {printf("    Task Executable Arguments: %s\n", ex_args);}
free(ex_args);}
resource_req = NULL;
ll_get_data(task, LL_TaskGetFirstResourceRequirement, &resource_req);
while(resource_req) {
    rc = ll_get_data(resource_req, LL_ResourceRequirementName, &name);
    if (!rc) {printf("        Resource Req Name: %s\n", name); free(name);}
    rc = ll_get_data(resource_req, LL_ResourceRequirementValue, &value);
    if (!rc) {printf("        Resource Req Value: %d\n", value);}
    resource_req = NULL;
    ll_get_data(task, LL_TaskGetNextResourceRequirement, &resource_req);
}
task = NULL;
ll_get_data(node, LL_NodeGetNextTask, &task);
}
node = NULL;
ll_get_data(step, LL_StepGetNextNode, &node);
}
step = NULL;
ll_get_data(job, LL_JobGetNextStep, &step);
}
job = ll_next_obj(queryObject);
}
ll_free_objs(queryObject);
ll_deallocate(queryObject);

/* Step 3: Display Floating Consumable Resources information of LL cluster. */

/* Initialize the query: Cluster query */
queryObject = ll_query(CLUSTERS);
if (!queryObject) {
    printf("Query CLUSTERS: ll_query() returns NULL.\n");
    exit(1);
}
ll_set_request(queryObject, QUERY_ALL, NULL, ALL_DATA);
cluster = ll_get_objs(queryObject, LL_CM, NULL, &obj_count, &err_code);
if (!cluster) {
    printf("Query CLUSTERS: ll_get_objs() returns NULL. Error code = %d\n", err_code);
}
printf("Number of Cluster objects = %d\n", obj_count);
while(cluster) {
    resource = NULL;
    ll_get_data(cluster, LL_ClusterGetFirstResource, &resource);
    while(resource) {
        rc = ll_get_data(resource, LL_ResourceName, &res_name);
        if (!rc) {printf("Resource Name = %s\n", res_name); free(res_name);}
        rc = ll_get_data(resource, LL_ResourceInitialValue, &value);
        if (!rc) {printf("Resource Initial Value = %d\n", value);}
        rc = ll_get_data(resource, LL_ResourceAvailableValue, &value);
        if (!rc) {printf("Resource Available Value = %d\n", value);}
        resource = NULL;
        ll_get_data(cluster, LL_ClusterGetNextResource, &resource);
    }
    cluster = ll_next_obj(queryObject);
}
ll_free_objs(queryObject);
ll_deallocate(queryObject);
}

```

Figure 53. Obtaining machine, job, and cluster information with the data access API (Part 4 of 4)

Example 2: The following example shows how LoadLeveler's data access API can be used to extract job accounting information saved in a history file.

Examples of using the data access API

```
#include <stdio.h>
#include "llapi.h"
#define STR_NULL(ptr) (ptr ? ptr : "")

main(int argc, char *argv[])
{
    LL_element *queryObject, *job = NULL, *step = NULL;
    LL_element *mach_usage = NULL, *disp_usage = NULL, *event_usage = NULL;
    int64_t int64_data;
    int rc, obj_count, err_code, job_count, step_count, int_data;
    char *str_data;
    char *start_dates[] = { "01/23/2006", "01/25/2006", NULL };
    char *end_dates[] = { "01/23/2006", "02/01/2006", NULL };
    int mach_usage_count, disp_usage_count, event_usage_count;

    /* Initialize the query: Job query */
    queryObject = ll_query(JOBS);
    if (!queryObject) { printf("Query JOBS: ll_query() returns NULL.\n"); exit(1); }

    /* Request information of job steps started/ended between certain dates. */
    rc = ll_set_request(queryObject, QUERY_STARTDATE, start_dates, ALL_DATA);
    if (rc) { printf("ll_set_request() - QUERY_STARTDATE - RC = %d\n", rc); exit(1); }
    rc = ll_set_request(queryObject, QUERY_ENDDATE, end_dates, ALL_DATA);
    if (rc) { printf("ll_set_request() - QUERY_ENDDATE - RC = %d\n", rc); exit(1); }

    /* Get the requested job objects from the specified history file. */
    job = ll_get_objs(queryObject, LL_HISTORY_FILE,
        "/tmp/spool/c209f1n05/history", &obj_count, &err_code);
    if (!job) { printf("ll_get_objs() returns NULL. Error code = %d\n", err_code); exit(1); }

    printf("*****\n");
    printf("Number of job objects returned = %d\n", obj_count);
    printf("*****\n");

    /* Loop through the job objects. */
    job_count = 0;
    while (job) {
        job_count++;
        printf("=====\n");
        printf("Job number = %d\n", job_count);

        /* Loop through the job step objects. */
    }
}
```

Figure 54. Extracting job accounting information from a history file (Part 1 of 3)

```

ll_get_data(job, LL_JobGetFirstStep, &step);
step_count = 0;
while (step) {
    step_count++;
    printf("=====\n");
    printf("    Step number = %d\n", step_count);
    ll_get_data(step, LL_StepID, &str_data);
    printf("    LL_StepID          = %s\n", STR_NULL(str_data));
    ll_get_data(step, LL_StepImageSize, &int_data);
    printf("    LL_StepImageSize     = %d\n", int_data);
    ll_get_data(step, LL_StepImageSize64, &int64_data);
    printf("    LL_StepImageSize64   = %lld\n", int64_data);

    /* Process CPU limit */
    ll_get_data(step, LL_StepCpuLimitHard, &int_data);
    printf("    LL_StepCpuLimitHard   = %d\n", int_data);
    ll_get_data(step, LL_StepCpuLimitHard64, &int64_data);
    printf("    LL_StepCpuLimitHard64 = %lld\n", int64_data);
    ll_get_data(step, LL_StepCpuLimitSoft, &int_data);
    printf("    LL_StepCpuLimitSoft   = %d\n", int_data);
    ll_get_data(step, LL_StepCpuLimitSoft64, &int64_data);
    printf("    LL_StepCpuLimitSoft64 = %lld\n", int64_data);

    /* Job Step CPU limit */
    ll_get_data(step, LL_StepCpuStepLimitHard64, &int64_data);
    printf("    LL_StepCpuStepLimitHard64 = %lld\n", int64_data);
    ll_get_data(step, LL_StepCpuStepLimitSoft64, &int64_data);
    printf("    LL_StepCpuStepLimitSoft64 = %lld\n", int64_data);

    /* Process Data Limit */
    ll_get_data(step, LL_StepDataLimitHard64, &int64_data);
    printf("    LL_StepDataLimitHard64   = %lld\n", int64_data);
    ll_get_data(step, LL_StepDataLimitSoft64, &int64_data);
    printf("    LL_StepDataLimitSoft64   = %lld\n", int64_data);

    /* CPU time used by the job step. */
    ll_get_data(step, LL_StepStepUserTime64, &int64_data);
    printf("    LL_StepStepUserTime64     = %lld (microsecs)\n", int64_data);
    ll_get_data(step, LL_StepStepSystemTime64, &int64_data);
    printf("    LL_StepStepSystemTime64   = %lld (microsecs)\n", int64_data);

    /* Loop through the machine usage objects. */
    /* A parallel job step run on 3 machines typically has 3 machine usage objects. */
    mach_usage_count = 0;
    rc = ll_get_data(step, LL_StepGetFirstMachUsage, &mach_usage);
    while (mach_usage) {
        mach_usage_count++;

```

Figure 54. Extracting job accounting information from a history file (Part 2 of 3)

Examples of using the data access API

```
printf("-----\n");
printf("    Machine Usage number      = %d\n", mach_usage_count);
ll_get_data(mach_usage, LL_MachUsageMachineName, &str_data);
printf("    Machine name                = %s\n", STR_NULL(str_data));

/* Loop through the dispatch usage objects. */
disp_usage_count = 0;
ll_get_data(mach_usage, LL_MachUsageGetFirstDispUsage, &disp_usage);
while (disp_usage) {
    disp_usage_count++;
    printf("-----\n");
    printf("    Dispatch Usage number          = %d\n", disp_usage_count);

    ll_get_data(disp_usage, LL_DispatchUsageStepUserTime64, &int64_data);
    printf("    LL_DispatchUsageStepUserTime64 = %lld (microsecs)\n", int64_data);
    ll_get_data(disp_usage, LL_DispatchUsageStepSystemTime64, &int64_data);
    printf("    LL_DispatchUsageStepSystemTime64 = %lld (microsecs)\n", int64_data);

    /* Loop through the event usage objects. */
    /* Each dispatch typically has 2 events: "started" and "completed". */
    /* There may be other events if the LL administrator executes the command */
    /* "llctl -g capture <user event name>" while the job is running. */
    event_usage_count = 0;
    ll_get_data(disp_usage, LL_DispatchUsageGetFirstEventUsage, &event_usage);
    while (event_usage) {
        event_usage_count++;
        printf("-----\n");
        printf("    Event Usage number              = %d\n", event_usage_count);
        ll_get_data(event_usage, LL_EventUsageEventName, &str_data);
        printf("    LL_EventUsageEventName          = %s\n", STR_NULL(str_data));
        ll_get_data(event_usage, LL_EventUsageStepUserTime64, &int64_data);
        printf("    LL_EventUsageStepUserTime64     = %lld (microsecs)\n", int64_data);
        ll_get_data(event_usage, LL_EventUsageStepSystemTime64, &int64_data);
        printf("    LL_EventUsageStepSystemTime64   = %lld (microsecs)\n", int64_data);
        ll_get_data(disp_usage, LL_DispatchUsageGetNextEventUsage, &event_usage);
    }
    ll_get_data(mach_usage, LL_MachUsageGetNextDispUsage, &disp_usage);
}
rc = ll_get_data(step, LL_StepGetNextMachUsage, &mach_usage);
}
ll_get_data(job, LL_JobGetNextStep, &step);
}
job = ll_next_obj(queryObject);
}
exit(0);
}
```

Figure 54. Extracting job accounting information from a history file (Part 3 of 3)

Error handling API

This API allows you to gather the information contained in the LoadLeveler error object and output that information as an error message.

ll_error subroutine

Purpose

ll_error – Converts a LoadLeveler error object to an error message string. As an option, you can print the error message string to stdout or stderr.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
char *ll_error (LL_element **errObj, int print_to);
```

Parameters

errObj

This is the address of a pointer to a LoadLeveler error object. A NULL *errObj* pointer indicates that any internal error objects will be printed.

print_to

- 1 - print error message to stdout
- 2 - print error message to stderr
- Any other value - no error message printed

Description

This subroutine also prints any error messages generated by APIs that do not support an error object as a parameter. When called with a NULL *errObj*, any error messages previously stored by such APIs, will be printed as directed by the *print_to* parameter and then deleted.

It is the caller's responsibility to free the storage associated with the error message string.

The LoadLeveler error object pointed to by **errObj* is deleted upon exit and NULL is assigned to **errObj*.

Return Values

The **ll_error** API returns a NULL return code if there is no error object to print.

Fair share scheduling API

This API allows operations on fair share scheduling.

ll_fair_share subroutine

Purpose

ll_fair_share – Allows LoadLeveler administrators to perform certain fair share scheduling operations.

Library

LoadLeveler API library **libllapi.a**

Syntax

```
#include "llapi.h"
```

```
int ll_fair_share (int version, LL_element **errObj,
                  LL_fair_share_param *param);
```

Parameters

version

Is an input parameter that indicates the LoadLeveler API version (this should be **LL_API_VERSION**).

errObj

Is the address of a pointer to an **LL_element**. The pointer to **LL_element** must be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed through the **ll_error** function. The caller must free the error object storage before reusing the pointer.

param

Is an input parameter of the address of a **LL_fair_share_param** structure defined in **llapi.h**.

In the **LL_fair_share_param** structure, the fields are defined as follows:

int operation

Specifies an intended operation on fair share scheduling with one of the following values:

FAIR_SHARE_RESET

Indicates that fair share scheduling will be reset either to start from scratch again by erasing all previous historic fair share data, or to start from a previously saved point by reading from a saved data file.

FAIR_SHARE_SAVE

Indicates that a snapshot of the historic fair share data for fair share scheduling will be saved to a file in a specified directory.

char *savedir

Specifies the directory for saving a snapshot of the historic fair share data for fair share scheduling. The directory should be initialized to NULL if not used. A valid directory writable by the central manager must be specified for the **FAIR_SHARE_SAVE** operation.

char *saved_file

Indicates that the previously saved file will be used in the **FAIR_SHARE_RESET** operation, if specified, to reset fair share scheduling to a known point. The file should be initialized to NULL if not used.

ll_fair_share subroutine

Description

The `ll_fair_share()` subroutine is the API for the `llfs` command used to perform certain fair share scheduling operations.

The `ll_fair_share` subroutine can be used to reset fair share scheduling or to save a snapshot of the historic fair share data to a file.

This function is for LoadLeveler administrators and the BACKFILL scheduler only.

Return Values

API_OK

Request successfully sent to LoadLeveler.

API_CANT_CONNECT

Failed to connect to a LoadLeveler daemon.

API_SCHEDD_DOWN

One or more of the LoadLeveler Schedd daemons are down.

API_INPUT_NOT_VALID

Input data is not valid.

API_NOT_SUPPORTED

Fair share scheduling is not supported by the scheduler.

API_NO_PERMISSION

The user does not have permission to perform this operation.

API_CANT_READ_FILE

The specified file cannot be read.

API_CANT_WRITE_FILE

Failed to write data to a file.

API_NOT_ENABLED

Fair share scheduling is not enabled.

API_CANT_TRANSMIT

A data transmission failure occurred.

API_CONFIG_ERR

Errors were encountered while processing configuration files.

Related Information

Commands: `llfs`

Subroutines: `ll_error`

Query API

This API provides information about the jobs and machines in the LoadLeveler cluster. You can use this together with the workload management API, since the workload management API requires you to know which machines are available and which jobs need to be scheduled. See “Workload management API” on page 619 for more information. These APIs exist for backward compatibility. It is recommended that you use the data access API when possible.

LoadLeveler for Linux does not support the query API.

The query API consists of the following subroutines:

- **ll_free_jobs** subroutine
- **ll_free_nodes** subroutine
- **ll_get_jobs** subroutine
- **ll_get_nodes** subroutine

ll_free_jobs subroutine

Purpose

ll_free_jobs – Frees storage that was allocated by ll_get_jobs.

Library

LoadLeveler API library libllapi.a

Syntax

```
#include "llapi.h"

int ll_free_jobs (LL_get_jobs_info *ptr);
```

Parameters

ptr Specifies the address of the LL_get_jobs_info structure to be freed.

Description

This subroutine, available to any user, frees the storage pointed to by the LL_get_jobs_info pointer.

Return Values

This subroutine returns a value of zero when successful. Otherwise, it returns an integer value defined in the llapi.h file.

Error Values

-8 The *version_num* member of the LL_get_jobs_info structure did not match the current version.

Examples

Makefiles and examples which use this subroutine are located in the samples/llsch subdirectory of the release directory.

Related Information

Subroutines: ll_free_nodes, ll_get_jobs, ll_get_nodes

ll_free_nodes subroutine

Purpose

ll_free_nodes – Frees storage that was allocated by ll_get_nodes.

Library

LoadLeveler API library libllapi.a

Syntax

```
#include "llapi.h"

int ll_free_nodes (LL_get_nodes_info *ptr);
```

Parameters

ptr Specifies the address of the LL_get_nodes_info structure to be freed.

Description

This subroutine, available to any user, frees the storage pointed to by the LL_get_nodes_info pointer.

Return Values

This subroutines returns a value of zero when successful.

Error Values

-8 The *version_num* member of the LL_get_jobs_info structure did not match the current version.

Examples

Makefiles and examples which use this subroutine are located in the **samples/llsch** subdirectory of the release directory.

Related Information

Subroutines: ll_free_nodes, ll_get_jobs, ll_get_nodes

ll_get_jobs subroutine

Purpose

ll_get_jobs – Returns information about all jobs in the LoadLeveler job queue.

Note: This is an obsolete API and is supported for backward compatibility only.

Library

LoadLeveler API library **libllapi.a**

Syntax

```
#include "llapi.h"

int ll_get_jobs (LL_get_jobs_info *);
```

Parameters

ptr Specifies the pointer to the **LL_get_jobs_info** structure that was allocated by the caller. The **LL_get_jobs_info** members are:

int *version_num*

Represents the version number of the **LL_start_job_info** structure. This should be set to **LL_PROC_VERSION**.

int *numJobs*

Represents the number of entries in the array.

LL_job ***JobList*

Represents the pointer to an array of **LL_job** structures. The **LL_job** structure is defined in **llapi.h**.

Description

This subroutine, available to any user, contains an array of **LL_job** structures indicating each job in the LoadLeveler system.

Some job information, such as the start time of the job, is not available to this API. (It is recommended that you use the dispatch time, which is available, in place of the start time.) Also, some accounting information is not available to this API.

Return Values

This subroutine returns a value of zero when successful. Otherwise, it returns an integer value defined in the **llapi.h** file.

Error Values

- 1 There is an error in the input parameter.
- 2 The API cannot connect to the central manager.
- 3 The API cannot allocate memory.
- 4 A configuration error occurred.

Examples

Makefiles and examples which use this subroutine are located in the **samples/llsch** subdirectory of the release directory.

Related Information

Subroutines: **ll_free_jobs**, **ll_free_nodes**, **ll_get_nodes**.

ll_get_nodes subroutine

Purpose

ll_get_nodes – Returns information about all of nodes known by the negotiator daemon.

Note: This is an obsolete API and is supported for backward compatibility only.

Library

LoadLeveler API library **libllapi.a**

Syntax

```
#include "llapi.h"

int ll_get_nodes(LL_get_nodes_info *ptr);
```

Parameters

ptr Specifies the pointer to the **LL_get_nodes_info** structure that was allocated by the caller. The **LL_get_nodes_info** members are:

int *version_num*
Represents the version number of the **LL_start_job_info** structure.

int *numNodes*
Represents the number of entries in the *NodeList* array.

LL_node ***NodeList*
Represents the pointer to an array of **LL_node** structures. The **LL_node** structure is defined in **llapi.h**.

Description

This subroutine, available to any user, contains an array of **LL_job** structures indicating each node in the LoadLeveler system.

Return Values

This subroutine returns a value of zero when successful.

Error Values

- 1 There is an error in the input parameter.
- 2 The API cannot connect to the central manager.
- 3 The API cannot allocate memory.
- 4 A configuration error occurred.

Examples

Makefiles and examples which use this subroutine are located in the **samples/llsch** subdirectory of the release directory.

Related Information

Subroutines: **ll_free_jobs**, **ll_free_nodes**, **ll_get_jobs**

Reservation API

This API allows you to make, change, and remove reservations. In addition, it provides the ability to bind job steps to a reservation and unbind job steps from a reservation. General users should refer to “Working with reservations” on page 197 for additional information. Additional information for LoadLeveler administrators is in “Configuring LoadLeveler to support reservations” on page 121.

This API consists of the following subroutines:

- **ll_bind subroutine**
- **ll_change_reservation subroutine**
- **ll_init_reservation_param subroutine**
- **ll_make_reservation subroutine**
- **ll_remove_reservation subroutine**

II_bind subroutine

Purpose

II_bind – Enables you to bind job steps to a reservation. The **II_bind** subroutine can also be used to unbind a list of job steps from the reservations to which they are currently bound to, or whichever reservation they have requested to bind to in the event that the bind has not yet occurred.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int ll_bind (int version, LL_element **errObj, LL_bind_param **param);
```

Parameters

version

Is an input parameter that indicates the LoadLeveler API version (this should be the same value as **LL_API_VERSION** in **llapi.h**).

errObj

Provides the address of a pointer to an **LL_element** that points to an error object if this function fails.

The caller must free the error object storage before reusing the pointer. You can also use the **II_error** subroutine to display error messages stored in the error object. If you are going to use the **II_error** subroutine, the pointer must be initialized to NULL to ensure that a valid pointer is passed to the **II_error** subroutine.

param

Provides the address of a pointer to a **LL_bind_param** structure as defined in **llapi.h**.

In the **LL_bind_param** structure, the fields are defined as follows:

char **jobsteplist

A NULL-terminated array of job or step identifiers. When a job identifier is specified, the action of the API is taken for all steps of the job. At least one job or step identifier must be specified.

The format of a job identifier is *host.jobid*. The format of a step identifier is *host.jobid.stepid*.

where:

- *host* is the name of the machine that assigned the job and step identifiers.
- *jobid* is the job number assigned to the job when it was submitted.
- *stepid* is the job step number assigned to the job step when it was submitted.

The job or step identifier may be specified in an abbreviated form, *jobid* or *jobid.stepid*, when the API is invoked on the same machine that assigned the job and step identifiers. In this case, LoadLeveler will use the local machine's *hostname* to construct the full job or step identifier.

ll_bind subroutine

| **Note:** For coscheduled jobs, even if all coscheduled job steps are not in
| the list of targeted job steps, the requested operation is
| performed on all coscheduled job steps.

char *ID

The reservation identifier. The format of a full LoadLeveler reservation identifier is *[host.]rid[.r]*.

where:

- *host* is the name of the machine that assigned the reservation identifier.
- *rid* is the number assigned to the reservation when it was created. An *rid* is required.
- *r* indicates that this is a reservation ID (*r* is optional).

The reservation identifier may be specified in an abbreviated form, *rid[.r]*, when the API is invoked on the same machine that assigned the reservation identifier. In this case, LoadLeveler will use the local machine's host name to construct the full reservation identifier.

int unbind

Indicates that a value of 1 means that the job steps in *jobsteplist* are to be unbound from the reservations to which they are currently bound. A value of 0 indicates that the job steps in the job step list are to be bound to the reservation specified by the ID.

Description

The `ll_bind()` subroutine is the API for the `llbind` command.

This function is for the BACKFILL scheduler only and only jobs in an idle-like state can be bound to a reservation.

LoadLeveler administrators can bind any job step to a reservation. If a job step is already bound to a reservation, it will first be unbound from the current reservation before being bound to the requested reservation. Nonadministrators must be the owner of the job steps to be bound or unbound, and either be the owner of the reservation or one of the users specified by the owner as having permission to use the reservation.

Return Values

RESERVATION_OK

Request successfully sent to LoadLeveler.

RESERVATION_REQUEST_DATA_NOT_VALID

Input is not valid.

RESERVATION_CONFIG_ERR

Errors were encountered while processing configuration files.

RESERVATION_NO_STORAGE

The system cannot allocate memory.

RESERVATION_CANT_TRANSMIT

A data transmission failure occurred.

RESERVATION_API_CANT_CONNECT

The subroutine cannot connect to the Schedd or central manager.

RESERVATION_NOT_SUPPORTED

The scheduler in use does not support reservations.

RESERVATION_NOT_EXIST

The reservation does not exist.

RESERVATION_NO_PERMISSION

Permission cannot be granted.

RESERVATION_WRONG_STATE

The reservation is not in the correct state.

Related Information

Commands: **llbind**

Subroutines: **ll_error**, **ll_make_reservation**

ll_change_reservation subroutine

Purpose

ll_change_reservation – Enables you to change the attributes of a reservation.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int ll_change_reservation (int version, LL_element **errObj, char **ID,  
                          LL_reservation_change_param **param);
```

Parameters

version

Is an input parameter that indicates the LoadLeveler API version (this should be the same value as **LL_API_VERSION** in **llapi.h**).

errObj

Provides the address of a pointer to an **LL_element** that points to an error object if this function fails.

The caller must free the error object storage before reusing the pointer. You can also use the **ll_error** subroutine to display error messages stored in the error object. If you are going to use the **ll_error** subroutine, the pointer must be initialized to NULL to ensure that a valid pointer is passed to the **ll_error** subroutine.

When using a job command file to change a reservation once the job is successfully submitted, an informational message will be returned in *errObj* that contains the name of the job.

ID Provides the reservation identifier. The format of a full LoadLeveler reservation identifier is *[host.]rid[.r]*.

where:

- *host* is the name of the machine that assigned the reservation identifier.
- *rid* is the number assigned to the reservation when it was created. An *rid* is required.
- *r* indicates that this is a reservation ID (*r* is optional).

The reservation identifier may be specified in an abbreviated form, *rid[.r]*, when the API is invoked on the same machine that assigned the reservation identifier. In this case, LoadLeveler will use the local machine's host name to construct the full reservation identifier.

param

Provides the address of a pointer to a NULL-terminated array of **LL_reservation_change_param** structures as defined in **llapi.h**. The caller must allocate and free storage for the **LL_reservation_change** structures.

In the **LL_reservation_change_param** structure, the fields are defined as follows:

enum LL_reservation_data type

Contains the type of the data to modify as shown in Table 99 on page 605.

Table 99. enum LL_reservation_data type

To modify	Specify	Type of new data
start_time	RESERVATION_START_TIME	char *
start_time	RESERVATION_ADD_START_TIME	int *
duration	RESERVATION_DURATION	int *
duration	RESERVATION_ADD_DURATION	int *
number_of_nodes	RESERVATION_BY_NODE	int *
number_of_nodes	RESERVATION_ADD_NUM_NODE	int *
hostlist	RESERVATION_BY_HOSTLIST	char **, NULL terminated
hostlist	RESERVATION_ADD_HOSTS	char **, NULL terminated
hostlist	RESERVATION_DEL_HOSTS	char **, NULL terminated
jobstep	RESERVATION_BY_JOBSTEP	char *
job_command_file	RESERVATION_BY_JCF	char *
userlist	RESERVATION_USERLIST	char **, NULL terminated
userlist	RESERVATION_ADD_USERS	char **, NULL terminated
userlist	RESERVATION_DEL_USERS	char **, NULL terminated
grouplist	RESERVATION_GROUPLIST	char **, NULL terminated
grouplist	RESERVATION_ADD_GROUPS	char **, NULL terminated
grouplist	RESERVATION_DEL_GROUPS	char **, NULL terminated
shared mode	RESERVATION_MODE_SHARED	int *; *data = 0: Not Shared *data = 1: Share
remove on idle mode	RESERVATION_MODE_REMOVE_ON_IDLE	int *; *data = 0: Do not remove on Idle *data = 1: Remove on Idle
owner	RESERVATION_OWNER	char *
group	RESERVATION_GROUP	char *
number_of_bg_cnodes	RESERVATION_BY_BG_CNODE	int *

If several options are available to modify the same type of data, only one is allowed. For example, RESERVATION_DURATION and RESERVATION_ADD_DURATION are mutually exclusive. *number_of_nodes*, *hostlist*, *jobstep*, *job_command_file*, and *number_bg_cnodes* are all used to modify the reserved nodes and, therefore, the associated enums are all mutually exclusive.

The duration of a reservation can be decreased (corresponding to the **-d -nn** option on **llchres**) by specifying the data type RESERVATION_ADD_DURATION and providing a negative value.

ll_change_reservation subroutine

The same is true for RESERVATION_ADD_NUM_NODE and RESERVATION_ADD_START_TIME.

void *data

Specifies the new data for the modification corresponding to *type*.

Description

The **ll_change_reservation()** subroutine is the API for the **llchres** command. The "Notes on changing a reservation" listed in the **llchres** command also apply to the **ll_change_reservation** subroutine.

More than one attribute of a reservation can be changed with a single call. Either all of the changes can and will be made, or none of the changes will be made. If the changes cannot be made, *errObj* will contain a message indicating a reason for the failure. The message may not contain all of the reasons the request cannot be satisfied.

A coscheduled job step cannot be used for changing a reservation.

This function is for the BACKFILL scheduler only.

Only LoadLeveler administrators and the owner of the reservation can use this function.

Return Values

RESERVATION_OK

Request successfully sent to LoadLeveler.

RESERVATION_TOO_CLOSE

Reservation is being made within the minimum advance time.

RESERVATION_NO_STORAGE

The system cannot allocate memory.

RESERVATION_CONFIG_ERR

Errors were encountered while processing configuration files.

RESERVATION_USER_LIMIT_EXCEEDED

Exceeds the maximum number of reservations for the user.

RESERVATION_GROUP_LIMIT_EXCEEDED

Exceeds the maximum number of reservations for the group.

RESERVATION_NO_PERMISSION

Permission cannot be granted.

RESERVATION_WRONG_STATE

The reservation is not in the correct state.

RESERVATION_API_CANT_CONNECT

The subroutine cannot connect to the Schedd or central manager.

RESERVATION_JOB_SUBMIT_FAILED

Submit of the job command file failed.

RESERVATION_NO_MACHINE

One or more machines in the host list are not in the LoadLeveler cluster.

RESERVATION_WRONG_MACHINE

Reservations are not permitted on one or more machines in the host list.

RESERVATION_NO_RESOURCE

Insufficient resources in the LoadLeveler cluster.

RESERVATION_NO_JOBSTEP

The job step used for node selection does not exist.

RESERVATION_WRONG_JOBSTEP

The job step used for node selection is not in the right state.

RESERVATION_NOT_SUPPORTED

The scheduler in use does not support reservations.

RESERVATION_NOT_EXIST

The reservation does not exist.

RESERVATION_REQUEST_DATA_NOT_VALID

Input is not valid.

RESERVATION_CANT_TRANSMIT

A data transmission failure occurred.

RESERVATION_TOO_LONG

The duration exceeds the maximum reservation duration.

Related Information

Commands: llchres

Subroutines: ll_error, ll_make_reservation

ll_init_reservation_param subroutine

Purpose

ll_init_reservation_param – Initializes the optional fields in the **LL_reservation_param** structure to default values prior to passing that structure to the **ll_make_reservation** subroutine.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int ll_init_reservation_param (int version, LL_element **errObj,  
                             LL_reservation_param **param);
```

Parameters

version

Is an input parameter that indicates the LoadLeveler API version (this should be the same value as **LL_API_VERSION** in **llapi.h**).

errObj

Provides the address of a pointer to an **LL_element** that points to an error object if this function fails.

The caller must free the error object storage before reusing the pointer. You can also use the **ll_error** subroutine to display error messages stored in the error object. If you are going to use the **ll_error** subroutine, the pointer must be initialized to **NULL** to ensure that a valid pointer is passed to the **ll_error** subroutine.

param

Provides the address of a pointer to a **LL_reservation_param** structure. The **LL_reservation_param** structure will be initialized.

Description

The **ll_init_reservation_param()** subroutine is used in conjunction with the **ll_make_reservation** subroutine. A program using this function would only have to set the required fields and any optional fields where the default value is not applicable.

Return Values

This subroutine returns a zero to indicate success.

Related Information

Subroutines: **ll_make_reservation**

ll_make_reservation subroutine

Purpose

ll_make_reservation – Enables you to create a LoadLeveler reservation.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int ll_make_reservation (int version, LL_element **errObj,
                        LL_reservation_param **param);
```

Parameters

version

Is an input parameter that indicates the LoadLeveler API version (this should be **LL_API_VERSION**).

errObj

Provides the address of a pointer to an **LL_element** that points to an error object if this function fails.

The caller must free the error object storage before reusing the pointer. You can also use the **ll_error** subroutine to display error messages stored in the error object. If you are going to use the **ll_error** subroutine, the pointer must be initialized to NULL to ensure that a valid pointer is passed to the **ll_error** subroutine.

If a job command file is used to create a reservation and the job is successfully submitted, an informational message will be returned.

param

Provides the address of a pointer to a **LL_reservation_param** structure defined in **llapi.h**. The caller must allocate and free storage for this structure. It is suggested that the caller use the **ll_init_reservation_param** call to initialize the structure.

In the **LL_reservation_param** structure, the fields are defined as follows:

char **ID

Contains the address where the reservation ID is to be returned.

char *start_time

Contains a string specifying the start time of a reservation in the format of "[mm/dd[/[cc]yy]]HH:MM".

int duration

Specifies how long the reservation lasts in the unit of minutes.

enum LL_reservation_data data_type

Indicates how the nodes should be reserved. The valid values are:

RESERVATION_BY_NODE	by number of nodes
RESERVATION_BY_HOSTLIST	by specifying a hostlist
RESERVATION_BY_JOBSTEP	by specifying a jobstep
RESERVATION_BY_JCF	by job command file
RESERVATION_BY_BG_CNODE	by number of Blue Gene c-nodes

void *data

Contains the pointer to the actual data specifying what nodes to reserve:

ll_make_reservation subroutine

	data_type	data is a pointer of the type
	RESERVATION_BY_NODE	int *
	RESERVATION_BY_HOSTLIST	char **, a NULL terminated array of machine names
	RESERVATION_BY_JOBSTEP	char *, a jobstep name in the format of host.jobid.stepid
	RESERVATION_BY_JCF	char *, the full pathname to a LoadLeveler Job Command File
	RESERVATION_BY_BG_CNODE	int * for number of Blue Gene c-nodes

int options

Specifies options that control characteristics of the reservation. The follow values can be OR'ed together to set this parameter:

RESERVATION_SHARED

Selects the SHARED option for the reservation. For a SHARED reservation, after all bound job steps which can run on the reserved nodes are scheduled to run, the remaining resources can be used to run job steps not bound to the reservation. Only bound job steps can be scheduled to run on a reservation that is not shared.

RESERVATION_REMOVE_ON_IDLE

Selects the REMOVE_ON_IDLE option for the reservation. For a REMOVE_ON_IDLE reservation, if all bound job steps are finished or if all bound job steps are Idle and none can run on the reserved nodes, the reservation will be removed (canceled) automatically by LoadLeveler. If this option is not set, the reservation will remain, regardless of being used or not.

char **users

Contains the list of users who can use the reservation. This pointer should be set to NULL so only the reservation owner and the LoadLeveler administrator can use the reservation.

char **groups

Contains the list of LoadLeveler groups whose users can use the reservation. This pointer should be set to NULL so only the list of LoadLeveler groups and the LoadLeveler administrator can use the reservation.

char *group

Contains a string of a LoadLeveler group that will own the reservation.

Description

The **ll_make_reservation ()** subroutine is the API for the **llmkres** command used to create a new reservation.

The **ll_init_reservation_param** subroutine can be used to initialize the **LL_reservation_param** structure.

A coscheduled job step cannot be used for making a reservation.

This function is for the BACKFILL scheduler only.

The “ll_init_reservation_param subroutine” on page 608 is a convenience function that can be used to initialize the **LL_reservation_param** structure. You must set the ID, start_time, duration, data_type, and data fields of the **LL_reservation_param** structure. If any of these fields are NULL or 0 (as appropriate), the call will fail.

Only users authorized by LoadLeveler administrators to make reservations can use this function.

Return Values

RESERVATION_OK

Request successfully sent to LoadLeveler.

RESERVATION_LIMIT_EXCEEDED

Exceeds the maximum number of reservations allowed for the cluster.

RESERVATION_TOO_CLOSE

Reservation is being made within the minimum advance time.

RESERVATION_NO_STORAGE

The system cannot allocate memory.

RESERVATION_CONFIG_ERR

Errors were encountered while processing configuration files.

RESERVATION_USER_LIMIT_EXCEEDED

Exceeds the maximum number of reservations for the user.

RESERVATION_GROUP_LIMIT_EXCEEDED

Exceeds the maximum number of reservations for the group.

RESERVATION_NO_PERMISSION

Permission cannot be granted.

RESERVATION_SCHEDD_CANT_CONNECT

The Schedd cannot connect to the central manager.

RESERVATION_API_CANT_CONNECT

The subroutine cannot connect to the Schedd or central manager.

RESERVATION_JOB_SUBMIT_FAILED

Submit of the job command file failed.

RESERVATION_NO_MACHINE

One or more machines in the host list are not in the LoadLeveler cluster.

RESERVATION_WRONG_MACHINE

Reservations are not permitted on one or more machines in the host list.

RESERVATION_NO_RESOURCE

Insufficient resources in the LoadLeveler cluster.

RESERVATION_NO_JOBSTEP

The job step used for node selection does not exist.

RESERVATION_WRONG_JOBSTEP

The job step used for node selection is not in the right state.

RESERVATION_NOT_SUPPORTED

The scheduler in use does not support reservations.

RESERVATION_REQUEST_DATA_NOT_VALID

Input is not valid.

RESERVATION_CANT_TRANSMIT

A data transmission failure occurred.

RESERVATION_TOO_LONG

The duration exceeds the maximum reservation duration.

Related Information

Commands: **llmkres**

Subroutines: **ll_error**

A sample program called **res.c** is provided in the **samples/llres** subdirectory of the release directory.

ll_remove_reservation subroutine

Purpose

ll_remove_reservation – Enables you to cancel one or more reservations.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"

int ll_remove_reservation (int version, LL_element **errObj, char **IDs,
                           char **user_list, char **host_list, char **group_list,
                           char **base_partition_list = NULL);
```

Parameters

version

Is an input parameter that indicates the LoadLeveler API version (this should be the same value as **LL_API_VERSION** in **llapi.h**).

errObj

Provides the address of a pointer to an **LL_element** that points to an error object if this function fails.

The caller must free the error object storage before reusing the pointer. You can also use the **ll_error** subroutine to display error messages stored in the error object. If you are going to use the **ll_error** subroutine, the pointer must be initialized to **NULL** to ensure that a valid pointer is passed to the **ll_error** subroutine.

IDs

Specifies a **NULL**-terminated array of reservation identifiers. The format of a full LoadLeveler reservation identifier is *[host.]rid[.r]*.

where:

- *host* is the name of the machine that assigned the reservation identifier.
- *rid* is the number assigned to the reservation when it was created. An *rid* is required.
- *r* indicates that this is a reservation ID (*r* is optional).

The reservation identifier may be specified in an abbreviated form, *rid[.r]*, when the API is invoked on the same machine that assigned the reservation identifier. In this case, LoadLeveler will use the local machine's host name to construct the full reservation identifier.

user_list

Specifies a **NULL**-terminated array of user IDs that own reservations.

host_list

Specifies a **NULL**-terminated array of machine names. One member of **all** indicates all Blue Gene base partitions.

group_list

Specifies a **NULL**-terminated array of LoadLeveler groups.

base_partition_list

Specifies a **NULL**-terminated array of Blue Gene base partitions. One member of **all** indicates **all** Blue Gene base partitions.

Description

The `ll_remove_reservation()` subroutine is the API for the `llrmres` command.

A list of reservation IDs cannot be specified when *user_list*, *host_list*, *base_partition_list*, or *group_list* is specified. If reservation IDs is non-NULL when *user_list*, *host_list*, *base_partition_list*, or *group_list* is also non-NULL, the return value will be `RESERVATION_REQUEST_DATA_NOT_VALID`. Input for the *user_list*, *host_list*, *group_list*, or *base_partition_list* parameters can be provided in any combination. The *host_list* and *base_partition_list* parameters are mutually exclusive.

This function is for the BACKFILL scheduler only.

Only LoadLeveler administrators and the owner of the reservation can use this function.

Return Values

RESERVATION_OK

Request successfully sent to LoadLeveler.

RESERVATION_REQUEST_DATA_NOT_VALID

Input is not valid.

RESERVATION_CONFIG_ERR

Errors were encountered while processing configuration files.

RESERVATION_NO_STORAGE

The system cannot allocate memory.

RESERVATION_CANT_TRANSMIT

A data transmission failure occurred.

RESERVATION_API_CANT_CONNECT

The subroutine cannot connect to the Schedd or central manager.

RESERVATION_NOT_SUPPORTED

The scheduler in use does not support reservations.

Related Information

Commands: `llrmres`

Subroutines: `ll_error`, `ll_make_reservation`

Submit API

This API allows you to submit jobs to LoadLeveler. The submit API consists of the following subroutines and a user exit for monitoring programs:

- **llfree_job_info subroutine**
- **llsubmit subroutine**
- **monitor_program user exit**

In LoadLeveler for Linux only, **llsubmit** returns an error value of `-1` and writes the error messages to `stderr` when **SEC_ENABLEMENT** is **CTSEC**.

llfree_job_info subroutine

Purpose

llfree_job_info – Frees space for the array and the job step information used by **llsubmit**.

Syntax

```
void llfree_job_info (LL_job *job_info, int job_version);
```

Parameters

job_info

Is a pointer to a **LL_job** structure. Upon return, the space pointed to by the **step_list** variable and the space associated with the **LL_job** step structures pointed to by the **step_list** array are freed. All fields in the **LL_job** structure are set to zero.

job_version

Is an integer indicating the version of **llfree_job_info** being used. This argument should be set to **LL_JOB_VERSION** which is defined in the **llapi.h** header file.

llsubmit subroutine

Purpose

llsubmit – Submits jobs to LoadLeveler for scheduling. It is both the name of a LoadLeveler command used to submit jobs as well as the subroutine described here.

Syntax

```
int llsubmit (char *job_cmd_file, char *monitor_program, char *monitor_arg,  
             LL_job *job_info, int job_version);
```

Parameters

job_cmd_file

Is a pointer to a string containing the name of the job command file.

monitor_program

Is a pointer to a string containing the name of the monitor program to be invoked when the state of the job is changed. Set to NULL if a monitoring program is not provided.

monitor_arg

Is a pointer to a string which is stored in the job object and is passed to the monitor program. The maximum length of the string is 1023 bytes. If the length exceeds this value, it is truncated to 1023 bytes. Set to NULL if an argument is not provided.

job_info

Is a pointer to a structure defined in the **llapi.h** header file. No fields are required to be filled in. Upon return, the structure will contain the number of job steps in the job command file and a pointer to an array of pointers to information about each job step. Space for the array and the job step information is allocated by **llsubmit**. The caller should free this space using the **llfree_job_info** subroutine.

job_version

Is an integer indicating the version of **llsubmit** being used. This argument should be set to **LL_JOB_VERSION** which is defined in the **llapi.h** include file.

Description

LoadLeveler must be installed and configured correctly on the machine on which the submit application is run.

The uid and gid in effect when **llsubmit** is invoked are the uid and gid used when the job is run.

To submit a job to a remote cluster, call the **ll_cluster** API to define the cluster prior to calling the **llsubmit** API. If the job being submitted has the **cluster_list** keyword defined in the job command file, the cluster specified by the **ll_cluster** API takes precedence over the **cluster_list** keyword.

Return Values

0 The job was submitted successfully.

Error Values

-1 Error, error messages written to stderr.

Related Information

Subroutines: `ll_cluster`

monitor_program user exit

Purpose

Using the **monitor_program** user exit, you can create a program that monitors jobs submitted using the **llsubmit** subroutine. The Schedd daemon invokes this monitor program if the **monitor_program** argument to **llsubmit** is not null. The monitor program is invoked each time a job step changes state. This means that the monitor program will be informed when the job step is started, completed, vacated, removed, or rejected. If you suspect the monitor program encountered problems or did not run, you should check the listing in the Schedd log. In the event of a monitor program failure, the job is still run.

Syntax

```
monitor_program job_id user_arg state exit_status
```

Parameters

monitor_program

Is the name of the program supplied in the `monitor_program` argument passed to the **llsubmit** function.

job_id

Is the full ID for the job step.

user_arg

The string supplied to the `monitor_arg` argument that is passed to the **llsubmit** function.

state

Is the current state of the job step. Possible values for the state are:

JOB_STARTED

The job step has started.

JOB_COMPLETED

The job step has completed.

JOB_VACATED

The job step has been vacated. The job step will be rescheduled if the job step is restartable or if it is checkpointable.

JOB_REJECTED

A **startd** daemon has rejected the job. The job will be rescheduled to another machine if possible.

JOB_REMOVED

The job step was canceled or could not be started.

JOB_NOTRUN

The job step cannot be run because a dependency cannot be met.

exit_status

Is the exit status from the job step. The argument is meaningful only if the state is **JOB_COMPLETED**.

Workload management API

The workload management API consists of the following subroutines:

- **ll_cluster** subroutine
- **ll_cluster_auth** subroutine
- **ll_control** subroutine
- **ll_modify** subroutine
- **ll_move_job** subroutine
- **ll_move_spool** subroutine
- **ll_preempt** subroutine
- **ll_preempt_jobs** subroutine
- **ll_run_scheduler** subroutine
- **ll_start_job** subroutine
- **ll_start_job_ext** subroutine
- **ll_terminate_job** subroutine

The **ll_control** subroutine can be used to perform most of the LoadLeveler control operations and is designed for general use.

The **ll_cluster**, **ll_cluster_auth**, and **ll_move_job** subroutines are for use in the multicluster environment.

The **ll_preempt** subroutine is not available in LoadLeveler for Linux. For LoadLeveler 3.3, the **ll_preempt** subroutine was replaced with the **ll_preempt_jobs** subroutine.

The **ll_start_job_ext** and **ll_terminate_job** subroutines are intended to be used together with an external scheduler.

Note that the **ll_start_job_ext** and **ll_terminate_job** subroutines automatically connect to an alternate central manager if they cannot contact the primary central manager. You should use **ll_start_job_ext** and **ll_terminate_job** in conjunction with the query API. The query API collects information regarding which machines are available and which jobs need to be scheduled. See “Query API” on page 595 for more information.

Note: The AIX Workload Manager (WLM) and the LoadLeveler workload management API are two distinct and unrelated components.

ll_cluster subroutine

Purpose

ll_cluster – Enables other APIs to operate on a remote cluster.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int ll_cluster (int version, LL_element **errObj, LL_cluster_param *param);
```

Parameters

version

Input parameter that indicates the LoadLeveler API version (should have the same value as **LL_API_VERSION** in **llapi.h**).

errObj

Provides the address of a pointer to **LL_element** that points to an error object if this function fails.

param

Provides a pointer to an **LL_cluster_param** structure.

```
typedef enum LL_cluster_op {
    CLUSTER_SET,
    CLUSTER_UNSET
} ClusterOp_t;

typedef struct {
    ClusterOp_t action;
    char **cluster_list;
}
LL_cluster_param;
```

In the **enum LL_Cluster_op** structure, valid values are:

CLUSTER_SET

Sets the multicluster environment to *cluster_list*.

CLUSTER_UNSET

Unsets the multicluster environment.

In the **LL_cluster_param** structure, the fields are defined as follows:

action

Determines whether the cluster environment should be set or unset. The set action must have a corresponding *cluster_list*. The unset action ignores the *cluster_list* data.

cluster_list

Is a NULL-terminated array of cluster stanza names. Only one cluster name is allowed. The reserved words **any** and **all** are not allowed.

Description

This API is called to enable other APIs to run on remote clusters in a multicluster environment and can be invoked by all users. To issue an API multiple times for different clusters, the **ll_cluster** and corresponding APIs must be issued for each cluster.

Return Values**CLUSTER_SUCCESS**

Cluster name successfully set.

CLUSTER_SYSTEM_ERROR

LoadLeveler internal system error.

CLUSTER_INVALID_CLUSTER_PARAM

An input parameter that is not valid was specified. Possible causes are that the *cluster_list* parameter is NULL or that the reserved words **any** or **all** were specified.

CLUSTER_INVALID_ACTION_PARAM

An *action* parameter that is not valid was specified. Valid values are **CLUSTER_UNSET** and **CLUSTER_SET**.

Related Information

Subroutines: ll_ckpt, ll_deallocate, ll_free_objs, ll_get_data, ll_get_objs, ll_modify, ll_next_obj, ll_query, ll_reset_request, ll_set_request, llsubmit

ll_cluster_auth subroutine

Purpose

ll_cluster_auth – Creates a public key, a private key, and a security certificate.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int ll_cluster_auth (int version, LL_cluster_auth_param **param);
```

Parameters

version

Input parameter that indicates the LoadLeveler API version (should have the same value as **LL_API_VERSION** in **llapi.h**).

param

Provides the address of a pointer to an array of pointers to **LL_cluster_auth_param** structures. The last element of the array must be **NULL**.

```
typedef enum LL_cluster_auth_op {  
    CLUSTER_AUTH_GENKEY  
} ClusterAuthOp_t;  
  
typedef struct {  
    ClusterAuthOp_t type;  
}  
LL_cluster_auth_param;
```

In the **enum LL_cluster_auth_op** structure, valid values are:

CLUSTER_AUTH_GENKEY
Generates required keys.

In the **enum LL_cluster_auth_op** structure, the fields are defined as follows:

type

Indicates the requested operation.

Description

The **ll_cluster_auth()** subroutine is the API for the **llclusterauth** command. Refer to the **llclusterauth** command for information about other available command options.

This function must be run from a process with **root** authority.

The **ll_cluster_auth()** subroutine creates a public key, a private key, a security certificate, and a directory for authorized keys. The keys and certificate are created in the **/var/LoadL/ssl** directory for AIX and in the **/var/opt/LoadL/ssl** directory for Linux.

- The private key is stored in **id_rsa**
- The public key is stored in **id_rsa.pub**
- The security certificate is stored in **id_rsa.cert**
- The authorized keys are stored in **authorized_keys**

In order for a connection to be accepted, the public key for the node requesting the connection must be stored in the authorized keys file on the node being connected to. Only a process with **root** authority can run this subroutine.

Return Values

The following return values are defined in llapi.h:

API_OK

Request successfully sent to LoadLeveler.

API_INVALID_INPUT

An input parameter that is not valid was specified.

API_CONFIG_ERR

Errors encountered while processing configuration files.

API_CANT_AUTH

Caller not authorized.

API_CANT_LISTEN

API cannot create listen socket.

API_CANT_CONNECT

Failed to connect to LoadLeveler.

API_TIMEOUT

Timed out waiting for a response.

API_SSL_ERR

Timed out waiting for a response.

API_STEP_NOT_IDLE

Request failed, the job step is not in the IDLE state.

Related Information

Commands: **llclusterauth**

ll_control subroutine

Purpose

ll_control – Allows an application program to perform most of the functions that are currently available through the standalone commands: **llctl**, **llfavorjob**, **llfavoruser**, **llhold**, and **llprio**.

In LoadLeveler for Linux only, **ll_control** returns an error condition when **SEC_ENABLEMENT** is **CTSEC**.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int ll_control (int control_version, enum LL_control_op control_op,
               char **host_list, char ** user_list, char **job_list,
               char **class_list, int priority);
```

Parameters

int *control_version*

An integer indicating the version of **ll_control** being used. This argument should be set to **LL_CONTROL_VERSION**.

enum *LL_control_op*

The control operation to be performed. The enum **LL_control_op** is defined in **llapi.h** as:

```
enum LL_control_op {
  LL_CONTROL_RECYCLE, LL_CONTROL_RECONFIG, LL_CONTROL_START, LL_CONTROL_STOP,
  LL_CONTROL_DRAIN, LL_CONTROL_DRAIN_STARTD, LL_CONTROL_DRAIN_SCHDED,
  LL_CONTROL_PURGE_SCHDED, LL_CONTROL_FLUSH, LL_CONTROL_SUSPEND,
  LL_CONTROL_RESUME, LL_CONTROL_RESUME_STARTD, LL_CONTROL_RESUME_SCHDED,
  LL_CONTROL_FAVOR_JOB, LL_CONTROL_UNFAVOR_JOB, LL_CONTROL_FAVOR_USER,
  LL_CONTROL_UNFAVOR_USER, LL_CONTROL_HOLD_USER, LL_CONTROL_HOLD_SYSTEM,
  LL_CONTROL_HOLD_RELEASE, LL_CONTROL_PPIO_ABS, LL_CONTROL_PPIO_ADJ,
  LL_CONTROL_START_DRAINED,};
```

char ***host_list*

A NULL-terminated array of host names.

char ***user_list*

A NULL-terminated array of user names.

char ***job_list*

A NULL-terminated array of job or step identifiers. When a job identifier is specified, the action of the API is taken for all steps of the job. At least one job or step identifier must be specified.

The format of a job identifier is *host.jobid*. The format of a step identifier is *host.jobid.stepid*.

where:

- *host* is the name of the machine that assigned the job and step identifiers.
- *jobid* is the job number assigned to the job when it was submitted.
- *stepid* is the job step number assigned to the job step when it was submitted.

The job or step identifier may be specified in an abbreviated form, *jobid* or *jobid.stepid*, when the API is invoked on the same machine that assigned the job

and step identifiers. In this case, LoadLeveler will use the local machine's host name to construct the full job or step identifier.

Note: For coscheduled jobs, even if all coscheduled job steps are not in the list of targeted job steps, the requested operation is performed on all coscheduled job steps only when holding or releasing jobs.

char ***class_list*

A NULL-terminated array of class names.

int *priority*

An integer representing the new absolute value of user priority or adjustment to the current user priority of job steps.

Description

The **ll_control** subroutine performs operations that are essentially equivalent to those performed by the standalone commands: **llctl**, **llfavorjob**, **llfavoruser**, **llhold**, and **llprio**. Because of this similarity, descriptions of the **ll_control** operations are grouped according to the standalone command they resemble.

llctl type of operations: These are the **ll_control** operations which mirror operations performed by the **llctl** command. This summary includes a brief description of each of the allowed **llctl** types of operations. For more information about the **llctl** command, see “llctl - Control LoadLeveler daemons” on page 409.

LL_CONTROL_START:

Starts the LoadLeveler daemons on the specified machines. The calling program must have rsh privileges to start LoadLeveler daemons on remote machines.

Note: LoadLeveler will fail to start if any value has been set for the **MALLOCTYPE** environment variable.

LL_CONTROL_START_DRAINED:

Starts the LoadLeveler in the drained state.

LL_CONTROL_STOP:

Stops the LoadLeveler daemons on the specified machines.

LL_CONTROL_RECYCLE:

Stops, and then restarts, all of the LoadLeveler daemons on the specified machines.

LL_CONTROL_RECONFIG:

Forces all of the LoadLeveler daemons on the specified machines to reread the configuration files.

LL_CONTROL_DRAIN:

When this operation is selected, the following happens: (1) No LoadLeveler jobs can start running on the specified machines, and (2) No LoadLeveler jobs can be submitted to the specified machines.

LL_CONTROL_DRAIN_SCHEDD:

No LoadLeveler jobs can be submitted to the specified machines.

LL_CONTROL_DRAIN_STARTD:

Keeps LoadLeveler jobs from starting on the specified machines. If a *class_list* is specified, then the classes specified will be drained (made unavailable). The literal string “**allclasses**” can be used as an abbreviation for all of the classes.

ll_control subroutine

LL_CONTROL_FLUSH:

Terminates running jobs on the specified machines and send them back to the negotiator to await redispach (if restart=yes).

LL_CONTROL_PURGE_SCHEDD:

Purges the specified Schedd host's job queue; a *host_list* consisting of one host name must be specified.

LL_CONTROL_SUSPEND:

Suspends all jobs on the specified machines. This operation is not supported for parallel jobs.

LL_CONTROL_RESUME:

Resumes job submission to, and job execution on, the specified machines.

LL_CONTROL_RESUME_STARTD:

Resumes job execution on the specified machines; if a *class_list* is specified, then execution of jobs associated with these classes is resumed.

LL_CONTROL_RESUME_SCHEDD:

Resumes job submission to the specified machines.

For these **llctl** type of operations, the *user_list*, *job_list*, and *priority* arguments are not used and should be set to **NULL** or zero. The *class_list* argument is meaningful only if the operation is **LL_CONTROL_DRAIN_STARTD**, or **LL_CONTROL_RESUME_STARTD**. If *class_list* is not being used, then it should be set to **NULL**. If *host_list* is **NULL**, then the scope of the operation is all machines in the LoadLeveler cluster. Unlike the standalone **llctl** command, where the scope of the operation is either global or one host, **ll_control** operations allow the user to specify a list of hosts (through the *host_list* argument). To perform these operations, the calling program must have LoadLeveler administrator authority. The only exception to this rule is the **LL_CONTROL_START** operation.

llfavorjob type of operations: The **llfavorjob** type of control operations are: **LL_CONTROL_FAVOR_JOB**, and **LL_CONTROL_UNFAVOR_JOB**. For these operations, the *user_list*, *host_list*, *class_list*, and *priority* arguments are not used and should be set to **NULL** or zero. **LL_CONTROL_FAVOR_JOB** is used to set specified job steps to a higher system priority than all job steps that are not favored. **LL_CONTROL_UNFAVOR_JOB** is used to unfavor previously favored job steps, restoring the original priorities. The calling program must have LoadLeveler administrator authority to perform these operations.

llfavoruser type of operations: The **llfavoruser** type of control operations are: **LL_CONTROL_FAVOR_USER**, and **LL_CONTROL_UNFAVOR_USER**. For these operations, the *host_list*, *job_list*, *class_list*, and *priority* arguments are not used and should be set to **NULL** or zero. **LL_CONTROL_FAVOR_USER** sets jobs of one or more users to the highest priority in the system, regardless of the current setting. Jobs already running are not affected. **LL_CONTROL_UNFAVOR_USER** is used to unfavor previously favored user's jobs, restoring the original priorities. The calling program must have LoadLeveler administrator authority to perform these operations.

llhold type of operations: The **llhold** type of control operations are: **LL_CONTROL_HOLD_USER**, **LL_CONTROL_HOLD_SYSTEM**, and **LL_CONTROL_HOLD_RELEASE**. For these operations, the *class_list* and *priority* arguments are not used, and should be set to **NULL** or zero. **LL_CONTROL_HOLD_USER** and **LL_CONTROL_HOLD_SYSTEM** place jobs in user hold and system hold, respectively. **LL_CONTROL_HOLD_RELEASE** is used

to release jobs from both types of hold. The calling program must have LoadLeveler administrator authority to put jobs into system hold, and to release jobs from system hold. If a job is in both user and system holds then the LL_CONTROL_HOLD_RELEASE operation must be performed twice to release the job from both types of hold. If the user is not a LoadLeveler administrator then the llhold types of operations have no effect on jobs that do not belong to that user.

llprio type of operations: The llprio type of control operations are: LL_CONTROL_PRIO_ABS, and LL_CONTROL_PRIO_ADJ. For these operations, the *user_list*, *host_list*, and *class_list* arguments are not used, and should be set to NULL. llprio type of operations change the user priority of one or more job steps in the LoadLeveler queue. LL_CONTROL_PRIO_ABS specifies a new absolute value of the user priority, and LL_CONTROL_PRIO_ADJ specifies an adjustment to the current user priority. The valid range of LoadLeveler user priorities is 0–100 (inclusive); 0 is the lowest possible priority, and 100 is the highest. The llprio type of operations have no effect on a running job step unless this job step returns to Idle state. If the user is not a LoadLeveler administrator, then an llprio type of operation has no effect on jobs that do not belong to that user.

Return Values

- 0 The specified command has been sent to the appropriate LoadLeveler daemon.
- 2 The specified command cannot be sent to the central manager.
- 3 The specified command cannot be sent to one of the **LoadL_master** daemons.
- 4 ll_control encountered an error while processing the administration or configuration file.
- 6 A data transmission failure has occurred.
- 7 The calling program does not have LoadLeveler administrator authority.
- 19 An incorrect **ll_control** version has been specified.
- 20 A system error has occurred.
- 21 The system cannot allocate memory.
- 22 A **control_op** operation that is not valid has been specified.
- 23 The **job_list** argument contains one or more errors.
- 24 The **host_list** argument contains one or more errors.
- 25 The **user_list** argument contains one or more errors.
- 26 Incompatible arguments have been specified for **HOLD** operation.
- 27 Incompatible arguments have been specified for **PRIORITY** operation.
- 28 Incompatible arguments have been specified for **FAVORJOB** operation.
- 29 Incompatible arguments have been specified for **FAVORUSER** operation.
- 30 An error occurred while ll_control tried to start a child process.
- 31 An error occurred while ll_control tried to start the **LoadL_master** daemon.
- 32 An error occurred while ll_control tried to execute the **llpurgeschedd** command.
- 33 The **class_list** argument contains incompatible information.
- 34 ll_control cannot create a file in the **/tmp** directory.
- 35 LoadLeveler has encountered miscellaneous incompatible input specifications.
- 41 This release of LoadLeveler for Linux does not support CTSEC.

Related Information

Commands: llctl, llfavorjob, llfavoruser, llhold, llprio

ll_modify subroutine

Purpose

ll_modify – Modifies the attributes of a submitted job step.

In LoadLeveler for Linux only, **ll_modify** returns an error condition when **SEC_ENABLEMENT** is **CTSEC**.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int ll_modify (int version, void *errObj, LL_modify_param **param,  
              char **steplist);
```

Parameters

version

Input parameter that indicates the LoadLeveler API version (should have the same value as **LL_API_VERSION** in **llapi.h**).

errObj

Provides the address of a pointer to **LL_element** that points to an error object if this function fails.

The caller must free the error object storage before reusing the pointer. You can also use the **ll_error** subroutine to display error messages stored in the error object. If you are going to do so, the pointer should be initialized to **NULL** to avoid a segmentation fault when the pointer is passed to the **ll_error** subroutine.

param

Provides the address of an array of 2 pointers to the **LL_modify_param** structure defined in **llapi.h**. The first pointer should point to an **LL_modify_param** structure already filled out by the caller. The second pointer should be assigned **NULL**.

In the **LL_modify_param** structure:

type Describes the attribute to be modified.

data Is a pointer to the new attribute value.

All job step attributes types that can be modified through **ll_modify()** are listed in **enum LL_modify_op** in **llapi.h**.

The **LL_modify_op** structure stores user inputs to the function, where:

type Is the type of the command option.

data Is a pointer to the data value associated with the command option.

name Is a **NULL** terminated array of job step names. Only a single job step is allowed in the current implementation.

steplist

A **NULL** terminated array of job or step identifiers. When a job identifier is specified, the action of the API is taken for all steps of the job. At least one job or step identifier must be specified.

The format of a job identifier is *host.jobid*. The format of a step identifier is *host.jobid.stepid*.

where:

- *host* is the name of the machine that assigned the job and step identifiers.
- *jobid* is the job number assigned to the job when it was submitted.
- *stepid* is the job step number assigned to the job step when it was submitted.

The job or step identifier may be specified in an abbreviated form, *jobid* or *jobid.stepid*, when the API is invoked on the same machine that assigned the job and step identifiers. In this case, LoadLeveler will use the local machine's host name to construct the full job or step identifier.

Description

ll_modify() is the API for the **llmodify** command. Refer to the **llmodify** command for information about other available command options.

In **enum LL_modify_op**:

- The system priority option will be ignored for any job step not in an idle state. The data field for the SYSPRIO option is an integer. The system priority for a job step set with the SYSPRIO option will not be changed when LoadLeveler recalculates system priorities.
- The **keyword** value is to be used to specify a string in the form, **keyword=value**, where keyword is the attribute of the job step to modify and *value* is the new value for the attribute. For a list of supported keywords and their descriptions, see "llmodify - Change attributes of a submitted job step" on page 435.
- The attributes that can be modified by the **enums** are described in "llmodify - Change attributes of a submitted job step" on page 435.

Return Values

The following return values are defined in llapi.h:

MODIFY_BAD_BG_CONNECTION

The Blue Gene connection request was not recognized.

MODIFY_BAD_BG_SHAPE

The Blue Gene partition shape is bad.

MODIFY_BAD_BG_SIZE

The Blue Gene partition size request was not positive.

MODIFY__CANT_MODIFY

The Blue Gene partition exists, but you cannot modify this value.

MODIFY__EMPTY_BG_PARTITION

The Blue Gene requested partition name was blank.

MODIFY_SUCCESS

Request successfully sent to LoadLeveler.

MODIFY_INVALID_PARAM

An input parameter that is not valid was specified.

MODIFY_CONFIG_ERROR

Errors encountered while processing config files.

MODIFY_NOT_IDLE

Request failed, job step not in IDLE state.

MODIFY_WRONG_JOB_TYPE

The modify operation is not valid for the job type.

MODIFY_WRONG_STATE

Request failed, job step in wrong state.

ll_modify subroutine

MODIFY_NOT_AUTH

Caller not authorized.

MODIFY_SYSTEM_ERROR

LoadLeveler internal system error.

MODIFY_CANT_TRANSMIT

Communication error while sending request.

MODIFY_CANT_CONNECT

Failed to connect to LoadLeveler.

MODIFY_INVALID_PARAM

The specified keyword is not supported.

MODIFY_NO_CTSEC_SUPPORT_ERR

SEC_ENABLEMENT was set to CTSEC. LoadLeveler for Linux does not support CTSEC.

MODIFY_NO_PVM_SUPPORT_ERR

pvm_root was specified. LoadLeveler for Linux does not support PVM.

MODIFY_OVERLAP_RESERVATION

Request failed. The requested change would cause the job step to overlap with a reservation.

Related Information

Commands: **llmodify**

Subroutines: **ll_cluster**, **ll_error**

Examples

```
/* mymodify.c - make a job step non-preemptable */
#include <stdio.h>
#include <string.h>
#include "llapi.h"

int main(int argc, char *argv[])
{
    int rc, preempt = 0;
    LL_modify_param mycmd, *cmdp[2];
    char *step_list[2];
    LL_element *errObj = NULL;
    char *errmsg;

    if (argc < 2) {
        printf("Usage: %s job_step_name \n", argv[0]); exit(1);
    }

    step_list[0] = argv[1];
    step_list[1] = NULL;
    printf("\n*** Make Job Step %s non-preemptable ***\n\n",
        step_list[0]);

    /* Initialize the LL_modify_param structure */
    mycmd.type = STEP_PREEMPTABLE;
    mycmd.data = &preempt;

    cmdp[0] = &mycmd
    cmdp[1] = NULL;

    /* Modify job step to be non preemptable */
    printf("Modify job step to be non-preemptable\n");
    rc = ll_modify(LL_API_VERSION, &errObj, cmdp, step_list);
    if (rc) {
```

```
|         errmsg = ll_error(&errObj, 0);  
|         printf("ll_modify() return code: %d\n%s\n", rc, errmsg);  
|         free(errmsg);  
|         exit(1);  
|     }  
|     return 0;  
| }
```

ll_move_job subroutine

Purpose

ll_move_job – Moves an idle job from one cluster to another.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"

int ll_move_job (int version, LL_element **errObj, LL_move_job_param **param);
```

Parameters

version

Input parameter that indicates the LoadLeveler API version (should have the same value as **LL_API_VERSION** in **llapi.h**).

errObj

Provides the address of a pointer to **LL_element** that points to an error object if this function fails.

param

Provides the address of a pointer to an **LL_move_job_param** structure.

```
typedef struct {
    char *cluster_name;
    char *job_id;
}
LL_move_job_param;
```

In the **LL_move_job_param** structure, the fields are defined as follows:

cluster_name

The name of the cluster to which the job will be moved.

job_id

The job ID of the job to be moved.

Description

The **ll_move_job()** subroutine is the API for the **llmovejob** command. Refer to the **llmovejob** command for information about other available command options.

The **ll_move_job** API moves a single idle job from one cluster to another. If any steps within the job are not idle, the transfer request is rejected and the API returns **API_STEP_NOT_IDLE**. The remote job retains the original *job_ID* from the local cluster. Upon transfer, the remote cluster performs any user mapping and remote job filtering necessary for the job.

Any changes made to the idle job in the local cluster by the **llmodify** command will not be carried forward to the remote cluster. Any jobs submitted when the local cluster was not configured as a part of a multicluster cannot be moved if the cluster's configuration is changed to a multicluster environment.

Only administrators can issue the **ll_move_job()** subroutine. In a mixed operating multicluster environment, administrators must ensure the binary compatibility of the job being transferred.

Return Values

The following return values are defined in llapi.h:

API_OK

Request successfully sent to LoadLeveler.

API_INVALID_INPUT

An input parameter that is not valid was specified.

API_CONFIG_ERR

Errors encountered while processing configuration files.

API_CANT_AUTH

Caller not authorized.

API_CANT_LISTEN

API cannot create listen socket.

API_CANT_CONNECT

Failed to connect to LoadLeveler.

API_TIMEOUT

Timed out waiting for a response.

API_STEP_NOT_IDLE

Request failed, the job step is not in the IDLE state.

Related Information

Commands: **llmovejob**

ll_move_spool subroutine

Purpose

ll_move_spool – Moves the job records from the spool directory of one managing Schedd to another managing Schedd in the local cluster.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int ll_move_spool (int version, LL_move_spool_param ** param,  
                  int (*func) (char *jobid,  
                                int rc, LL_element **messageObj),  
                  LL_element **errObj);
```

Parameters

version

Input parameter that indicates the LoadLeveler API version (should have the same value as **LL_API_VERSION** in **llapi.h**).

param

Provides the address of a pointer to an **LL_move_spool_param** structure.

```
typedef struct {  
    char **schedd_host;  
    char *spool_directory;  
    SpoolData_t data;  
} LL_move_spool_param;  
  
typedef enum move_spool_data {  
    LL_MOVE_SPOOL_JOBS  
} SpoolData_t;
```

*(*func)(char *jobid, int rc, LL_element **messageObj)*

Specifies the user-supplied function to be called after every job is processed. The function must return an integer and must accept as input:

- A char pointer representing the job ID processed
- An integer representing the return code of the job processing
- The address to a pointer of an **LL_element** data type representing any messages generated

The **LL_element** data type is defined in the **llapi.h** file. If a callback is not desired, the function pointer should be set to **NULL**.

errObj

Provides the address of a pointer to **LL_element** that points to an error object if this subroutine returns an unsuccessful return code.

Description

The **ll_move_spool()** subroutine is the API for the **llmovespool** command.

The **ll_move_spool** subroutine is intended for recovery purposes only. This subroutine moves the job records from the *spool_directory* of one managing *schedd_host* to another managing *schedd_host* in the local cluster. The **ll_move_spool** subroutine must be run from a machine that has read and write access to the specified *spool_directory* containing the job records being moved. This machine must also have network connectivity to the machine running the Schedd daemon that will receive the job records. Jobs to be moved can be in any state.

The *schedd_host* that created the job records to be moved must not be running during the move operation. Jobs within the job queue database will be unrecoverable if the job queue is updated during the move by any process other than the **ll_move_spool** subroutine. The *schedd_host* that created the job records to be moved must have the **schedd_fenced** machine stanza keyword set to **true** prior to the **ll_move_spool** subroutine being issued.

All moved jobs retain their original job identifiers. The **ll_move_spool** subroutine invokes the function specified by the (**func*) parameter after each job is processed. When the job records for a job are successfully transferred, the *schedd_host* of the job is updated to represent the new managing *schedd_host* and the job records in the specified *spool_directory* are deleted. If the transfer for any step within a job fails, the job records for that step remain in the specified *spool_directory*. If for some reason a job fails, the **ll_move_spool** subroutine should be reissued against the specified *spool_directory* to reprocess the job.

The user-supplied function enables messages to be displayed as each job is processed without having to wait for all of the jobs to be completed. If a callback is not desired, the function pointer should be set to NULL. The callback function must call the **ll_error** API passing in the *messageObj* in order to free the memory allocated to it.

The **ll_move_spool** subroutine does not move the reservation queue or fair share scheduling data found within the specified *spool_directory*.

Only administrators can issue the **ll_move_spool** subroutine.

A sample is available in:

{\$RELEASEDIR}/samples/llmovespool

Return Values

The following return values are defined in **llapi.h** and can be returned by the **ll_move_spool** API:

API_OK

Request successfully sent to LoadLeveler.

API_INPUT_NOT_VALID

An input parameter that is not valid was specified.

API_CONFIG_ERR

Errors encountered while processing configuration files.

API_CANT_AUTH

Caller not authorized.

API_CANT_LISTEN

API cannot create listen socket.

API_CANT_CONNECT

Failed to connect to LoadLeveler.

API_TIMEOUT

Timed out waiting for a response.

API_JOBQ_ERR

Error occurred reading source job queue or writing to destination job queue.

ll_move_spool subroutine

API_SCHEDD_DOWN

The *schedd_host* is not accepting jobs.

API_SCHEDD_NOT_FENCED

The original Schedd machine stanza does not have the **schedd_fenced** keyword set to **true**.

The following return values are defined in **llapi.h** and can be passed to the user-specified function:

API_CANT_TRANSMIT

schedd_host encountered an error while processing the executable or job command file job records.

API_OK

Request successfully sent to LoadLeveler.

API_CANT_CONNECT

Failed to connect to LoadLeveler.

API_TIMEOUT

Timed out waiting for a response.

API_JOBQ_ERR

Error occurred reading source job queue or writing to destination job queue.

API_JOB_NOT_FOUND

schedd_host encountered an error while processing the received job.

Related Information

Commands: **llmovespool**

ll_preempt subroutine

Purpose

ll_preempt – Enables you to preempt a running job step or to resume a job step that has already been preempted through the **llpreempt** command or the **ll_preempt** subroutine (user-initiated). The **ll_preempt** subroutine cannot resume a job step preempted through PREEMPT_CLASS rules (system-initiated).

The **ll_preempt** subroutine is not available in LoadLeveler for Linux.

For LoadLeveler 3.3, the **ll_preempt** subroutine was replaced with the **ll_preempt_jobs** subroutine.

Library

LoadLeveler API library, **libllapi.a**

Syntax

```
#include "llapi.h"
```

```
int ll_preempt (int version, LL_element **errObj, char *job_step,
               enum LL_preempt_op type);
```

Parameters

version

Input parameter that indicates the LoadLeveler API version (should have the same value as **LL_API_VERSION** in llapi.h).

errObj

Provides the address of a pointer to LL_element that points to an error object if this function fails.

The caller must free the error object storage before reusing the pointer. You can also use the ll_error subroutine to display error messages stored in the error object. If you are going to do so, the pointer should be initialized to NULL to avoid a segmentation fault when the pointer is passed to the ll_error subroutine.

jobstep

A string used to specify the name of a job step.

type

- Preempts job step if *type* equals PREEMPT_STEP
- Resumes job step if *type* equals RESUME_STEP

Description

ll_preempt() is the API for the **llpreempt** command.

- This function is for external schedulers
- Only LoadLeveler administrators have authority to use this function

Return Values

API_OK

Request successfully sent to LoadLeveler.

API_INVALID_INPUT

An input parameter that is not valid was specified.

API_CONFIG_ERR

Errors encountered while processing config files.

II_preempt subroutine

API_CANT_AUTH

Caller not authorized.

API_CANT_CONNECT

Failed to connect to LoadLeveler.

ll_preempt_jobs subroutine

Purpose

ll_preempt_jobs – Preempts a set of running job steps using the specified preempt method or resumes job steps that have already been preempted with the preempt method of suspend through the **llpreempt** command or the **ll_preempt_jobs** subroutine. The **ll_preempt_jobs** subroutine cannot resume a job step that was preempted through the PREEMPT_CLASS rules or a job step that was preempted with a preempt method other than suspend.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int ll_preempt_jobs (int version, void *errObj, LL_preempt_param **param);
```

Parameters

version

Is an input parameter that indicates the LoadLeveler API version (this should be the same value as **LL_API_VERSION** in **llapi.h**).

errObj

Provides the address of a pointer to an **LL_element** that points to an error object if this function fails.

param

Is a pointer to an array of structures. The structure specifies the parameters for a preempt operation.

```
typedef struct LL_preempt_param {
    enum LL_preempt_op type;
    enum LL_preempt_method method;
    char ** user_list;
    char ** host_list;
    char ** jobstep_list;
} LL_preempt_param;
```

```
enum LL_preempt_op {PREEMPT_STEP, RESUME_STEP};
```

```
enum LL_preempt_method {LL_PREEMPT_SUSPEND, LL_PREEMPT_VACATE,
    LL_PREEMPT_REMOVE, LL_PREEMPT_SYS_HOLD, LL_PREEMPT_USER_HOLD}
```

In the **LL_preempt_param** structure, the fields are defined as follows:

type Is the type of operation on the job steps, **preempt** or **resume**.

method Is the method to be used to preempt the specified job steps. This argument is ignored if the type argument is not set to **PREEMPT_STEP**. Valid values for this argument are:

- LL_PREEMPT_SUSPEND
- LL_PREEMPT_VACATE
- LL_PREEMPT_REMOVE
- LL_PREEMPT_SYS_HOLD
- LL_PREEMPT_USER_HOLD

LoadLeveler for Linux does not support preemption by the suspend method. Job steps running on Linux nodes will not be suspended when *method* in the **LL_preempt_param** structure

ll_preempt_jobs subroutine

has the value LL_PREEMPT_SUSPEND. The call to the **ll_preempt_jobs** subroutine is equivalent to a no-op in this case.

In the **enum LL_preempt_op** structure, valid values are:

- PREEMPT_STEP
- RESUME_STEP

In the **enum LL_preempt_method** structure, valid values are:

- LL_PREEMPT_SUSPEND
- LL_PREEMPT_VACATE
- LL_PREEMPT_REMOVE
- LL_PREEMPT_SYS_HOLD
- LL_PREEMPT_USER_HOLD

user_list

Is a pointer to a NULL-terminated array of pointers to strings containing user names. All running job steps belonging to all users in the list and monitored by the machine the subroutine is running on will be preempted or resumed. If a *host_list* is also specified, only the user's job steps monitored on the specified hosts will be preempted or resumed.

host_list

Is a pointer to a NULL-terminated array of pointers to strings containing host names. All job steps monitored by the hosts will be preempted or resumed. If a *user_list* is also provided, only the running job steps monitored by the hosts and owned by the specified users will be preempted or resumed.

jobstep_list

Is a pointer to a NULL-terminated array of job or step identifiers. When a job identifier is specified, the action of the API is taken for all steps of the job. At least one job or step identifier must be specified.

The format of a job identifier is *host.jobid*. The format of a step identifier is *host.jobid.stepid*.

where:

- *host* is the name of the machine that assigned the job and step identifiers.
- *jobid* is the job number assigned to the job when it was submitted.
- *stepid* is the job step number assigned to the job step when it was submitted.

The job or step identifier may be specified in an abbreviated form, *jobid* or *jobid.stepid*, when the API is invoked on the same machine that assigned the job and step identifiers. In this case, LoadLeveler will use the local machine's host name to construct the full job or step identifier.

Note: For coscheduled jobs, even if all coscheduled job steps are not in the list of targeted job steps, the requested operation is performed on all coscheduled job steps.

Description

The **ll_preempt_jobs()** subroutine is the API for the **llpreempt** command. See the **llpreempt** command for information about the available command options. In order to provide source code compatibility for applications using the current

version of the `preempt` function, the `ll_preempt()` subroutine will not be modified to support the new preemption options. The `ll_preempt()` subroutine will continue to be supported as is.

Return Values

- 0 Request successfully sent to the central manager.
- 1 An *ll_preempt_op* that is not valid was specified.
- 2 Cannot send request to central manager.
- 3 An incorrect version was specified.
- 4 Errors encountered while processing the LoadLeveler administration or configuration files.
- 5 A system error occurred.
- 6 A data transmission failure occurred.
- 7 The calling program does not have LoadLeveler administrator authority.

ll_run_scheduler subroutine

Purpose

ll_run_scheduler – Sends a request to the central manager to run the scheduling algorithm. It is used when the internal scheduling interval has been disabled so that an external program can control when the central manager attempts to schedule job steps.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int ll_run_scheduler (int version);
```

Parameters

version

Is an input parameter that indicates the LoadLeveler API version (this should be the same value as **LL_API_VERSION** in **llapi.h**).

Description

The **ll_run_scheduler()** subroutine sends a request to the central manager to run the LoadLeveler scheduling algorithm. The central manager's scheduling algorithm will run only once each time the **llrunscheduler** command is invoked. Each time the scheduling algorithm runs, the central manager will schedule as many job steps as the current available resources allow. The LoadLeveler scheduling interval must be disabled, that is the configuration keyword **NEGOTIATOR_INTERVAL** must be set to zero in order to use this algorithm. Only LoadLeveler administrators have authority to use this algorithm.

Return Values

RUN_SCHED_SUCCESS

Request successfully sent to the central manager.

RUN_SCHED_CONFIG_ERROR

Errors were encountered while processing configuration files.

RUN_SCHED_NOT_AUTH

Calling program does not have LoadLeveler administrator authority.

RUN_SCHED_NOT_AUTH

An internal system error occurred.

RUN_SCHED_CANT_TRANSMIT

A data transmission failure occurred.

RUN_SCHED_CANT_CONNECT

The Schedd cannot connect to the central manager.

ll_start_job subroutine

Purpose

ll_start_job – Tells the LoadLeveler negotiator to start a job on the specified nodes.

In LoadLeveler for Linux only, **ll_start_job** returns an error condition when **SEC_ENABLEMENT** is **CTSEC**.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int ll_start_job (LL_start_job_info *ptr);
```

Parameters

ptr Specifies the pointer to the **LL_start_job_info** structure that was allocated by the caller. The **LL_start_job_info** members are:

int *version_num*

Represents the version number of the **LL_start_job_info** structure. Should be set to **LL_PROC_VERSION**.

LL_STEP_ID *StepId*

Represents the step ID of the job step to be started.

char ***nodeList*

Is a pointer to an array of node names where the job will be started. The first member of the array is the parallel master node. The array must be ended with a **NULL**.

Description

This subroutine does not allow you to specify adapter usage information. Use the **ll_start_job_ext** subroutine instead.

You must set **SCHEDULER_TYPE = API** in the global configuration file to use this subroutine.

Only jobs steps currently in the Idle state are started.

Only processes having the LoadLeveler administrator user ID can invoke this subroutine.

An external scheduler uses this subroutine to start jobs that are in idle state. The list of jobs that are currently in the system is retrieved with the **ll_get_objs** API function, passing in a query element with type **JOBS**. The list of machines available to run jobs on is obtained with the **ll_get_objs** and a query element with type **MACHINES**. Additional data about both jobs and machines is obtained with the **ll_get_data** function call.

When this function is used to start a step, adapter resources are assigned to the step according to job command file network statements, if they are present. Adapter resources are assigned in the same manner as the **BACKFILL** scheduler assigns adapter resources, except that the **Communication Level** on the network statement is ignored and a value of **AVERAGE** is used. It is the responsibility of

ll_start_job subroutine

the external scheduler to ensure the machines to which the step is dispatched have sufficient adapter resources to run the step. Otherwise the step will not be started.

Return Values

This subroutine returns a value of zero to indicate the start job request was accepted by the negotiator. However, a return code of zero does not necessarily imply the job started. You can use the **llq** command to verify the job started. Otherwise, this subroutine returns an integer value defined in the **llapi.h** file.

Error Values

- 1 There is an error in the input parameter.
- 2 The subroutine cannot connect to the central manager.
- 4 An error occurred reading parameters from the administration or the configuration file.
- 5 The negotiator cannot find the specified *StepId* in the negotiator job queue.
- 6 A data transmission failure occurred.
- 7 The subroutine cannot authorize the action because you are not a LoadLeveler administrator.
- 8 The job object version number is incorrect.
- 9 The *StepId* is not in the Idle state.
- 10 One of the nodes specified is not available to run the job.
- 11 One of the nodes specified does not have an available initiator for the class of the job.
- 12 For one of the nodes specified, the requirements statement does not satisfy the job requirements.
- 13 The number of nodes specified was less than the minimum or more than the maximum requested by the job.
- 14 The LoadLeveler default scheduler is enabled.
- 15 The same node was specified twice in **ll_start_job** *nodelist*.

Examples

Makefiles and examples which use this subroutine are located in the **samples/llsch** subdirectory of the release directory. The examples include the executable **sch_api**, which invokes the query API and the job control API to start the second job in the list received from **ll_get_jobs** on two nodes. You should submit at least two jobs prior to running the sample. To compile **sch_api**, copy the sample to a writeable directory and update the **RELEASE_DIR** field to represent the current LoadLeveler release directory.

Related Information

Subroutines: **ll_get_jobs**, **ll_get_nodes**, **ll_start_job_ext**, **ll_terminate_job**

ll_start_job_ext subroutine

Purpose

ll_start_job_ext – Tells the LoadLeveler negotiator to start a job on the specified nodes, indicating which adapter and adapter resources to use.

In LoadLeveler for Linux only, **ll_start_job_ext** returns an error condition when **SEC_ENABLEMENT** is **CTSEC**.

An external scheduler uses this subroutine to start jobs that are in idle state. The list of jobs that are currently in the system is retrieved with the **ll_get_objs** API function, passing in a query element with type **JOBS**. The list of machines available to run jobs on is obtained with the **ll_get_objs** and a query element with type **MACHINES**. Additional data about both jobs and machines is obtained with the **ll_get_data** function call.

When this function is used to start a step, the external scheduler specifies the adapter resources that are assigned to the step and network statements in the job command file, if they are present, are ignored. It is the responsibility of the external scheduler to manage the availability of adapter resources and LoadLeveler does not prevent or detect the over commitment of adapter resources.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"
```

```
int ll_start_job_ext (LL_start_job_info_ext *ptr);
```

Parameters

ptr Specifies the pointer to the **LL_start_job_info_ext** structure that was allocated and populated by the caller. The **LL_start_job_info_ext** members are:

int *version_num*

Represents the version number of the **LL_start_job_info_ext** structure. Should be set to **LL_PROC_VERSION**.

LL_STEP_ID *StepId*

Represents the step ID of the job step to be started.

char ** *nodelist*

A pointer to an array of node names where the job will be started. The first member of the array is the parallel master node. The array must be ended with a **NULL**.

int *adapterUsageCount*

This is the size of the **adapterUsage** list. To determine what this number should be, add all the adapter usages for all protocols needed by one task and multiply the result by the number of tasks in the job.

LL_ADAPTER_USAGE * *adapterUsage*

This is a list of adapter information. The size of this list is given by **adapterUsageCount**. The members of this structure are:

char * *dev_name*

The device name of the adapter to be used.

ll_start_job_ext subroutine

char * *protocol*

A character string representing the communication protocol this usage supports. Valid values are **MPI**, **LAPI**, and **MPI_LAPI**.

char * *subsystem*

The communication subsystem this usage supports. Valid values are **IP** or **US**.

int *wid*

For **US** subsystem usages, this indicates which adapter window ID to use. For **IP** subsystem usages, this field is ignored.

uint64_t *mem*

For **US** subsystem usages, this is the amount of adapter memory to dedicate to the adapter usage. For **IP** subsystem usages, this field is ignored.

uint64_t *api_rcxtblocks*

For **US** subsystem usages, this is the number of user rCxt blocks to allocate for each adapter window. For **IP** subsystem usages, this field is ignored.

Notes:

1. This field should only be used on systems which contain `Switch_Network_Interface_For_HPS` adapters. When using this field, the *mem* field should not be used.
2. Previously existing applications which use `ll_start_job_ext` can be used unchanged to start jobs on systems with `Switch_Network_Interface_For_HPS` adapters, the value previously specified for the memory request will be ignored.

Each element in the adapterUsage list represents one communication channel for a task. If the subsystem is US (User Space), a communication channel will require a switch adapter window. Adapter windows, and User Space usages, must be specified on actual switch adapters that are only accessible if `AGGREGATE_ADAPTERS=False` is specified in the configuration file.

Description

You must set `SCHEDULER_TYPE = API` in the global configuration file to use this subroutine. In order to have access to the physical switch adapters in the LoadLeveler cluster (as opposed to virtual adapters representing all of the adapters on a network or adapters striping across multiple networks) you must specify `AGGREGATE_ADAPTERS = False` in the global configuration file.

Only jobs steps currently in the Idle state are started.

Only processes having the LoadLeveler administrator user ID can invoke this subroutine.

An external scheduler uses this subroutine in conjunction with the `ll_query` and `ll_get_data` subroutines of the query API. The query API returns information about which machines are available for scheduling and which jobs are currently in the job queue waiting to be scheduled.

The node list that is passed to the external scheduler API specifies the node on which each task of the job being started is to run. The distribution of tasks to nodes in the node list must be consistent with the node allocation specified by the job command file of the job being started. If it is not, the results are undefined and

the job may fail to start or may start with incorrect node assignments. Except when **BLOCKING** is specified, the entries for tasks that are running on the same node must all be specified sequentially in the node list. Table 100 describes how nodes should be arranged in the node list for the possible combinations of node and task specification in the job command file. In the table, '/' denotes integer division and (N mod M) is the remainder of the integer division of N by M.

Table 100. How nodes should be arranged in the node list

Job command file specification	Required nodelist structure
node=N tasks_per_node=T	There must be N different machine names specified, each specified T times.
node=N,M tasks_per_node=T	There must be between N and M different machine names specified, each specified T times.
node=N total_tasks=TT TT evenly divisible by N	There must be N different machine names specified, each specified TT/N times.
node=N total_tasks=TT TT not evenly divisible by N	There must be N different machine names specified. The first (TT mod N) unique machine names must each be specified (TT/N +1) times and the remaining machine names are each specified TT/N times.
total_tasks=TT BLOCKING=B TT evenly divisible by B	There must be TT/B sets of machine names specified, each set specifies a machine name B times. It is permissible for a machine name to be specified in more than one set.
total_tasks=TT BLOCKING=B TT not evenly divisible by B	There must be TT/B+1 sets of machine names specified. The first TT/B sets specify a machine name B times. The last set specifies a machine name (TT mod B) times. It is permissible for a machine name to be specified in more than one set.
total_tasks=TT BLOCKING=UNLIMITED	There must be TT entries in the node list. It is permissible for a machine name to be specified in the list more than once and it is not required that the specifications be contiguous.
task_geometry	There is a 1:1 correspondence between entries in the nodelist and task ids specified in the task geometry statement. Entries in the node list that correspond to task IDs in the same set must specify the same machine. Entries in the node list that correspond to task IDs in different sets must specify different machines.

Return Values

This routine returns a nonzero value to indicate the start request was not delivered to the negotiator. These values are defined in the **llapi.h** file and explained in "Error values." A return code of zero indicates the request was successfully delivered to the negotiator, but constraints on the negotiator may stop the job from starting. You can use the **llq** command to verify the job started.

Error Values

- 1 There is an error in the input parameter.
- 2 The subroutine cannot connect to the central manager.
- 4 An error occurred reading parameters from the administration or the configuration file.
- 5 The negotiator cannot find the specified StepId in the negotiator job queue.

ll_start_job_ext subroutine

- 6 A data transmission failure occurred.
- 7 The subroutine cannot authorize the action because you are not a LoadLeveler administrator.
- 8 The job object version number is incorrect.
- 9 The StepId is not in the Idle state.
- 10 One of the nodes specified is not available to run the job.
- 11 One of the nodes specified does not have an available initiator for the class of the job.
- 12 For one of the nodes specified, the requirements statement does not satisfy the job requirements.
- 13 The number of nodes specified was less than the minimum or more than the maximum requested by the job.
- 14 The LoadLeveler default scheduler is enabled.
- 15 The same node was specified twice in the ll_start_job nodelist.
- 20 Adapter usage information does not match job structure.
- 21 Adapter usage requested an adapter not on the machine.
- 22 Wrong number of entries on adapter usage list.
- 23 The adapter usage information did not specify the same protocol usage on each task.
- 24 An invalid protocol string was specified on an adapter usage.
- 25 The adapter usages specified incompatible protocols
- 26 An adapter usage specified a communication subsystem that was not IP or US

Examples

Makefiles and examples which use this subroutine are located in the **samples/llsch** subdirectory of the release directory. The examples include the executable **sch_api_ext**, which invokes the query API and the job control API to start the first job in the list received from **ll_query** on one node and to cancel the second job in the list. To compile **sch_api_ext**, copy the sample to a writeable directory and update the **RELEASE_DIR** field to represent the current LoadLeveler release directory.

Related Information

Subroutines: **ll_get_data**, **ll_query**, **ll_start_job**, **ll_terminate_job**

ll_terminate_job subroutine

Purpose

ll_terminate_job – Tells the negotiator to cancel the specified job step.

In LoadLeveler for Linux only, **ll_terminate_job** returns an error condition when **SEC_ENABLEMENT** is **CTSEC**.

Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

Syntax

```
#include "llapi.h"

int ll_terminate_job (LL_terminate_job_info *ptr);
```

Parameters

ptr Specifies the pointer to the **LL_terminate_info** structure that was allocated by the caller. The **LL_terminate_job_info** members are:

int *version_num*

Represents the version number of the **LL_terminate_job_info** structure. Should be set to **LL_PROC_VERSION**.

LL_STEP_ID *StepId*

Represents the step ID of the job step to be terminated.

Note: For coscheduled jobs, even if all coscheduled job steps are not in the list of targeted job steps, the requested operation is performed on all coscheduled job steps.

char **msg*

A pointer to a null terminated array of characters. If this pointer is null or points to a null string, a default message is used. This message will be available through **ll_get_data** to tell the process why a program was terminated.

Description

Only processes having the LoadLeveler administrator user ID can invoke this subroutine.

An external scheduler uses this subroutine in conjunction with the **ll_query** and **ll_get_data** subroutines of the query API. The external scheduler must use this subroutine to terminate interactive parallel jobs that cannot be run.

Return Values

This subroutine returns a value of zero when successful, to indicate the terminate job request was accepted by the negotiator. However, a return code of zero does not necessarily imply the negotiator canceled the job. Use the **llq** command to verify the job was canceled. Otherwise, this subroutine returns an integer value defined in the **llapi.h** file.

Error Values

- 1 There is an error in the input parameter.
- 4 An error occurred reading parameters from the administration or the configuration file.
- 6 A data transmission failure occurred.

ll_terminate_job subroutine

- 7 The subroutine cannot authorize the action because you are not a LoadLeveler administrator or you are not the user who submitted the job.
- 8 The job object version number is incorrect.

Examples

Makefiles and examples which use this subroutine are located in the **samples/llsch** subdirectory of the release directory. The examples include the executable **sch_api**, which invokes the query API and the job control API to terminate the first job reported by the **ll_get_jobs** subroutine. You should submit at least two jobs prior to running the sample. To compile **sch_api**, copy the sample to a writeable directory and update the **RELEASE_DIR** field to represent the current LoadLeveler release directory.

Related Information

Subroutines: **ll_get_jobs**, **ll_get_nodes**, **ll_start_job**

Appendix A. Troubleshooting LoadLeveler

This topic is divided into the following subtopics:

- “Frequently asked questions,” which contains answers to questions frequently asked by LoadLeveler customers. This topic focuses on answers that may help you get out of problem situations. The questions and answers are organized into the following categories:
 - LoadLeveler won't start. See “Why won't LoadLeveler start?” for more information.
 - **Jobs submitted to LoadLeveler do not run.** See “Why won't my job run?” on page 652 for more information.
 - **One or more of your machines goes down.** See “What happens to running jobs when a machine goes down?” on page 657 for more information.
 - **The central manager is not operating.** See “What happens if the central manager isn't operating?” on page 658 for more information.
 - **Resources need to be recovered from the Schedd machine.** See “How do I recover resources allocated by a Schedd machine?” on page 660 for more information.
 - **A core file needs to be found on Linux.** See “Why can't I find a core file on Linux?” on page 660 for more information.
 - **Inconsistencies are found in llfs output.** See “Why am I seeing inconsistencies in my llfs output?” on page 661 for more information.
 - **Configuration or administration file errors.** See “What happens if errors are found in my configuration or administration file?” on page 661 for more information.
 - **Miscellaneous questions.** See “Other questions” on page 661 for more information.
- “Troubleshooting in a multicluster environment” on page 663, which contains common questions and answers pertaining to operations within a multicluster environment.
- “Helpful hints” on page 666, which contains tips on running LoadLeveler, including some productivity aids.
- “Getting help from IBM” on page 670, which tells you how to contact IBM for assistance.

It is helpful to create error logs when you are diagnosing a problem. See to “Configuring recording activity and log files” on page 46 for information on setting up error logs.

Frequently asked questions

This topic contains answers to questions frequently asked by LoadLeveler customers.

Why won't LoadLeveler start?

If the **master** daemon will not run, go to the node where **LoadL_master** will not start and enter from the command line:

```
LoadL_master -t
```

Troubleshooting

This generates messages that might help to diagnose the problem. In addition, ensure the following are true:

1. The **Release** and **bin** directories are properly specified in the configuration files.
2. The administration file exists and is properly defined in the configuration file.
3. The central manager is correctly defined in the administration file.
4. The **log** directories are correctly defined in the configuration file.
5. The **spool**, **execute**, and **log** directories exist and permissions are set as follows:
 - The **spool** subdirectory is set to 700
 - The **execute** subdirectory is set to 1777
 - The **log** subdirectory is set to 775
6. The **LoadL_master** binary, in **/usr/lpp/LoadL/full/bin** for AIX or **/opt/ibmll/LoadL/full/bin** for Linux, is owned by **root** and has the setuid bit set.
7. The daemons are not already running. If they are already running, use the **ps** command to identify the processes, and then use the **kill** command to kill the daemons.
8. When cluster security services is enabled, all machines in the LoadLeveler cluster must list each other in their trusted hosts list for authentication.
9. When cluster security services is enabled, the **loadl** ID must be a member of the UNIX group identified by the **SEC_SERVICES_GROUP** configuration file keyword.

Note: LoadLeveler for Linux does not support cluster security services.

Why won't my job run?

If you submitted your job but it has not run, issue **llq -s** first to help diagnose the problem. If you need more help diagnosing the problem, refer to Table 101:

Table 101. Why your job might not be running

Why your job might not be running:	Possible solution
Job requires specific machine, operating system, or other resource.	Does the resource exist in the LoadLeveler cluster? If yes, wait until it becomes available. Check the GUI to compare the job requirements to the machine details, especially Arch , OpSys , and Class . Ensure that the spelling and capitalization matches.
Job requires specific job class	<ul style="list-style-type: none">• Is the class defined in the administration file? Use llclass to determine this. If yes,• Is there a machine in the cluster that supports that class? If yes, you need to wait until the machine becomes available to run your job.
The maximum number of jobs are already running on all the eligible machines	Wait until one of the machines finishes a job before scheduling your job.
The start expression evaluates to false.	Examine the configuration files (both LoadL_config and LoadL_config.local) to determine the START control function expression used by LoadLeveler to start a job. As a problem determination measure, set the START and SUSPEND values, as shown in this example: START: T SUSPEND: F

Table 101. Why your job might not be running (continued)

Why your job might not be running:	Possible solution
A job step is running on the node that your job requires, and that job step's preemption rules list your job's class as one that cannot share the node	The running job step is in a job class for which an administrator has defined preemption rules through the <code>PREEMPT_CLASS</code> keyword. When your job step's class is listed in the <code>ALL</code> clause of that keyword, your job step must wait until the running job step finishes.
The priority of your job is lower than the priority of other jobs.	You cannot affect the system priority given to this job by the negotiator daemon but you can try to change your user priority to move this job ahead of other jobs you previously submitted using the <code>llprio</code> command or the GUI.
The information the central manager has about machines and jobs may not be current.	Wait a few minutes for the central manager to be updated and then the job may be dispatched. This time limit (a few minutes) depends upon the polling frequency and polls per update set in the <code>LoadL_config</code> file. The default polling frequency is five seconds.
You do not have the same user ID on all the machines in the cluster.	To run jobs on any machine in the cluster, you have to have the same user ID and the same uid number on every machine in the pool. If you do not have a userid on one machine, your jobs will not be scheduled to that machine.
CtSec is enabled and the <code>.rhosts</code> file was not updated.	The <code>.rhosts</code> file should contain entries which specify all the host and user combinations allowed to submit jobs which will run as the local user. See "Steps for enabling CtSec services" on page 54 for more details.
Your job is not bound to a reservation under which nodes that your job requires to run are reserved	<p>When an unbound job requires nodes that are reserved under a reservation, LoadLeveler will not start the job unless one of the following conditions is true:</p> <ul style="list-style-type: none"> • The reservation was created with <code>SHARED</code> mode specified. If the reservation is using <code>SHARED</code> mode, your job will remain idle until the reservation state becomes <code>Active_Shared</code>. • The job's expected end time (current time plus the hard wall clock limit) indicates that the job will complete before the reservation starts. <p>If neither condition is true, but you have the authority to use the reservation, you may use the <code>llbind</code> command to bind your job to the reservation. Otherwise, your unbound job will remain idle until the reservation completes or is canceled.</p> <p>To check the reservation's status and attributes, use the <code>llqres</code> command. To find out which reservations you may use, check with your LoadLeveler administrator, or enter the command <code>llqres -l</code> and check the names in the Users or Groups fields (under the Modification time field) in the output listing. If your user name or a group name to which you belong appears in these output fields, you are authorized to use the reservation.</p>

Table 101. Why your job might not be running (continued)

Why your job might not be running:	Possible solution
Your job is bound to a reservation but the reservation is not active yet	LoadLeveler schedules bound job steps to run only when a reservation becomes active. Use the command llq -I to find the ID of the reservation to which the job is bound. Use the command llqres -I to find the start time of the reservation, and wait until that time to check the job status again.
Your job is bound to a reservation that does not reserve all of the resources that your job requires to run	<p>If a bound job requires specific resources that are not available during the reservation period, LoadLeveler will not dispatch the job to run under the reservation. This situation can occur if the job requires one or more of the following:</p> <ul style="list-style-type: none"> • Specific nodes that were not selected for the reservation. • More than the total number of reserved nodes. • Floating consumable resources, which cannot be reserved under a reservation. <p>If the LoadLeveler cluster has the resources that the job requires, use the command llbind -r, which unbinds the job from the reservation.</p>
Your job is bound to a reservation but the maximum number of jobs you may run has been reached already	If LoadLeveler detects that you currently are running the maximum number of jobs that you are allowed to run, it will not start your bound job even if the reservation is active.
Your job is bound to a reservation but the job's expected end time exceeds the reservation's end time	<p>LoadLeveler will dispatch your job only if its expected end time (current time plus the hard wall clock limit) does not exceed the end time of the reservation, or if both of the following conditions are true:</p> <ul style="list-style-type: none"> • This reservation is configured to allow jobs to continue running even when their expected end time exceeds the end of the reservation, and • The resources required to run your job are available. <p>Otherwise, this bound job will remain idle until either:</p> <ul style="list-style-type: none"> • The reservation completes or is canceled, or • You use the command llbind -r, which unbinds the job from the reservation.
Your job is bound to a reservation that does not exist	LoadLeveler puts your job in NotQueued state until the reservation is created. In that case, LoadLeveler will bind your job to the reservation. Otherwise, use the command llbind -r to unbind the job from the reservation.

You can use the **llq** command to query the status of your job or the **llstatus** command to query the status of machines in the cluster. Refer to Chapter 16, "Commands," on page 385 for information on these commands.

Why won't my parallel job run?

If you submitted your parallel job but it has not run, issue **llq -s** first to help diagnose the problem. If issuing this command does not help, refer to Table 101 on page 652 and to Table 102 on page 655 for more information:

Table 102. Why your job might not be running

Why your job might not be running:	Possible solution
The minimum number of processors requested by your job is not available.	Sufficient resources must be available. Specifying a smaller number of processors may help if your job can run with fewer resources.
The pool in your requirements statement specifies a pool which is invalid or not available.	The specified pool must be valid and available.
The adapter specified in the requirements statement or the network statement identifies an adapter which is invalid or not available.	<p>The specified adapter must be valid and available.</p> <p>Use llstatus -a to check the status of the adapters in the system. Switch adapters that show a state of 'NOT READY' or '-1' should be reported to the LoadLeveler administrator. Switch adapters with a state of '-1' indicate that the machine those adapters are on could not be queried for status.</p> <p>If the network statement specifies rcxtblocks, only Switch Network Interface for HPS adapters can be used for the step.</p>

Common set-up problems with parallel jobs

This topic presents a list of common problems found in setting up parallel jobs:

- If jobs appear to remain in a Pending or Starting state: check that the nameserver is consistent. Compare results of **host machine_name** and **host IP_address**
- For POE:
 - Specify the POE partition manager as the executable. Do *not* specify the parallel job as the executable.
 - Pass the parallel job as an argument to POE.
 - The parallel job must exist and must be specified as a full path name.
 - If the job runs in user space, specify the flag **-euilib us**.
 - Specify the correct adapter (when needed).
 - Specify a POE job only once in the job command file.
 - Compile only with the supported level of POE.
 - Specify only **parallel** as the *job_type*.

Why won't my checkpointed job restart?

If the job you submitted has the keyword **restart_from_ckpt = yes** and if the checkpoint file specified does not exist, the job will move to the Starting state and will then be removed from the queue. A mail message will be generated indicating the checkpoint file does not exist and a message will also appear in the StarterLog. Verify the values of the **ckpt_file** keyword in the Job Command File and the value of the **ckpt_dir** keyword in the Job Command or Administration File to ensure they resolve to the directory and file name of the desired checkpoint file.

Note: When a job is enabled for checkpoint, it is important to ensure the name of the checkpoint file is unique.

Why won't my submit-only job run?

If a job you submitted from a *submit-only* machine does not run, verify that you have defined the following statements in the machine stanza of the administration file of the submit-only machine:

```
submit_only = true
schedd_host = false
central_manager = false
```

Verify that another machine has set `schedd_host = true` and `schedd_runs_here = true`.

Why won't my job run on a cluster with both AIX and Linux machines?

The default shell on Linux (in both Red Hat Enterprise Linux and SUSE Linux Enterprise Server) is **bash** and **bash** may not be available on AIX. If a job step contains a **bash** script it will be rejected if it is run on an AIX node. The **ksh** is available on both AIX and Linux. You can specify which shell to use in the keyword **shell** in your job command file:

```
# @shell = /bin/ksh
```

Also, AIX and Linux are not binary compatible so jobs written in compiled languages such as C or Fortran must be compiled for the environment they will run on.

Does my Linux machine support CPU affinity?

If you are using the CPU affinity function that is available with LoadLeveler for Linux, you could have a problem with jobs that were submitted to a machine (or machines) being rejected. If this happens, you might receive an e-mail with a message that indicates that there are no available CPUs on the machine to run the job. This problem can occur if the **ALLOC_EXCLUSIVE_CPU_PER_JOB** keyword was set to **LOGICAL** or **PHYSICAL** in the local or global configuration files.

To resolve this problem, limit the number of initiators available on a node to the maximum available CPUs so that the number of jobs scheduled to a node never exceeds the available CPUs. This can be done by setting the **MAX_STARTERS** configuration keyword as follows:

```
MAX_STARTERS = number_of_available_CPUs
```

Administrators must be careful in setting up the maximum number of starters that can run on a compute node, if they turn on Linux CPU affinity support. Typically, **MAX_STARTERS** should be the same as the number of processors available on a compute node. To check to see if your kernel supports Linux CPU affinity, issue the following command:

```
root@c197blade4b07# uname -r
2.6.9-11.EL.mcr25smp
root@c197blade4b07#
```

The 2.6 kernel or higher is required. To get Linux kernel information about how many processors your node has, issue the following command to get a listing of available **LOGICAL** processors:

```
root@c197blade4b07# cat /proc/cpuinfo
```

Use the parameters provided by this listing, such as physical ID and siblings, to find the number of **PHYSICAL** processors available.

Why does a job stay in the Pending (or Starting) state?

If a job appears to stay in the Pending or Starting state, it is possible the job is continually being dispatched and rejected. Check the setting of the `MAX_JOB_REJECT` keyword. If it is set to -1 the job will be rejected an unlimited number of times. Try resetting this keyword to a small number, such as 10. Also, check the setting of the `ACTION_ON_MAX_REJECT` keyword. These keywords are described in Chapter 12, "Configuration file reference," on page 243.

What happens to running jobs when a machine goes down?

Both the `startd` daemon and the `Schedd` daemon maintain persistent states of all jobs. Both daemons use a specific protocol to ensure that the state of all jobs is consistent across `LoadLeveler`. In the event of a failure, the state can be recovered. Neither the `Schedd` nor the `startd` daemon discard the job state information until it is passed onto and accepted by another daemon in the process. Refer to Table 103 for more information.

Table 103. Troubleshooting running jobs when a machine goes down

If	Then
The network goes down but the machines are still running	If the network goes down but the machines are still running, when <code>LoadLeveler</code> is restarted, it looks for all jobs that were marked running when it went down. On the machine where the job is running, the <code>startd</code> daemon searches for the job and if it can verify that the job is still running, it continues to manage the job through completion. On the machine where <code>Schedd</code> is running, <code>Schedd</code> queues a transaction to the <code>startd</code> to reestablish the state of the job. This transaction stays queued until the state is established. Until that time, <code>LoadLeveler</code> assumes the state is the same as when the system went down.
The network partitions or goes down.	All transactions are left queued until the recipient has acknowledged them. Critical transactions such as those between the <code>Schedd</code> and <code>startd</code> are recorded on disk. This ensures complete delivery of messages and prevents incorrect decisions based on incomplete state information.
The machine with <code>startd</code> goes down.	Because job state is maintained on disk in <code>startd</code> , when <code>LoadLeveler</code> is restarted it can forward correct status to the rest of <code>LoadLeveler</code> . In the case of total machine failure, this is usually "JOB VACATED", which causes the job to be restarted elsewhere. In the case that only <code>LoadLeveler</code> failed, it is often possible to "find" the job if it is still running and resume management of it. In this case <code>LoadLeveler</code> sends <code>JOB RUNNING</code> to the <code>Schedd</code> and central manager, thereby permitting the job to run to completion.
The central manager machine goes down.	<p>All machines in the cluster send current status to the central manager on a regular basis. When the central manager restarts, it queries each machine that checks in, requesting the entire queue from each machine. Over the period of a few minutes the central manager restores itself to the state it was in before the failure. Each <code>Schedd</code> is responsible for maintaining the correct state of each job as it progressed while the central manager is down. Therefore, it is guaranteed that the central manager will correctly rebuild itself.</p> <p>All jobs started when the central manager was down will continue to run and complete normally with no loss of information. Users may continue to submit jobs. These new jobs will be forwarded correctly when the central manager is restarted.</p>

Table 103. Troubleshooting running jobs when a machine goes down (continued)

If	Then
The Schedd machine goes down	<p>When Schedd starts up again, it reads the queue of jobs and for every job which was in some sort of active state (i.e. PENDING, STARTING, RUNNING), it queries the machine where it is marked active.</p> <p>The running machine is required to return current status of the job. If the job completed while Schedd was down, JOB COMPLETE is returned with exit status and accounting information. If the job is running, JOB RUNNING is returned. If the job was vacated, JOB VACATED is returned. Because these messages are left queued until delivery is confirmed, no job will be lost or incorrectly dispatched due to Schedd failure.</p> <p>During the time the Schedd is down, the central manager will not be able to start new jobs that were submitted to that Schedd.</p> <p>To recover the resources allocated to jobs scheduled by a Schedd machine, see "How do I recover resources allocated by a Schedd machine?" on page 660.</p>
The lsubmit machine goes down	Schedd gets its own copy of the executable so it does not matter if the lsubmit machine goes down.

Why does lstatus indicate that a machine is down when llq indicates a job is running on the machine?

If a machine fails while a job is running on the machine, the central manager does not change the status of any job on the machine. When the machine comes back up the central manager will be updated.

What happens if the central manager isn't operating?

In one of your machine stanzas specified in the administration file, you specified a machine to serve as the central manager. It is possible for some problem to cause this central manager to become unusable such as network communication or software or hardware failures. In such cases, the other machines in the LoadLeveler cluster believe that the central manager machine is no longer operating. If you assigned one or more alternate central managers in the machine stanza, a new central manager will take control. The alternate central manager is chosen based upon the order in which its respective machine stanza appears in the administration file.

Once an alternate central manager takes control, it starts up its negotiator daemon and notifies all of the other machines in the LoadLeveler cluster that a new central manager has been selected. Figure 55 on page 659 illustrates how a machine can become the alternate central manager.

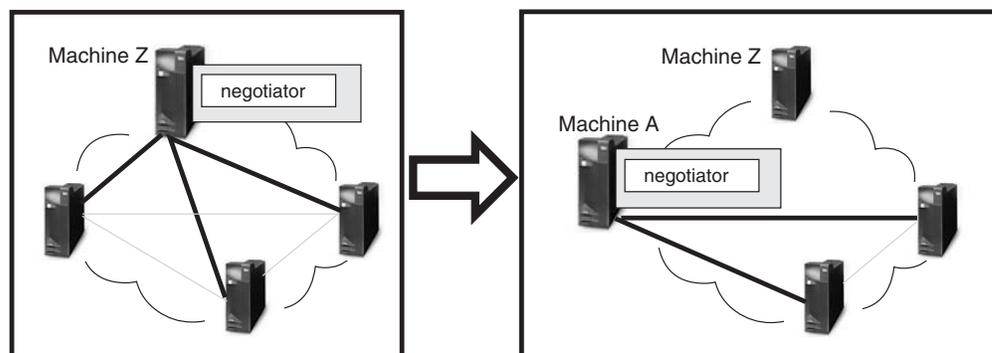


Figure 55. When the primary central manager is unavailable

The diagram illustrates that Machine Z is the primary central manager but Machine A took control of the LoadLeveler cluster by becoming the alternate central manager. Machine A remains in control as the alternate central manager until either:

- The primary central manager, Machine Z, resumes operation. In this case, Machine Z notifies Machine A that it is operating again and, therefore, Machine A terminates its negotiator daemon.
- Machine A also loses contact with the remaining machines in the pool. In this case, another machine authorized to serve as an alternate central manager takes control. Note that Machine A may remain as its own central manager.

Figure 56 illustrates how multiple central managers can function within the same LoadLeveler pool.

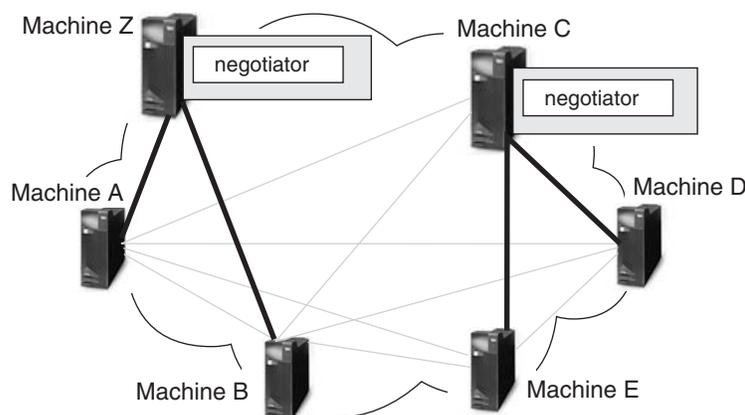


Figure 56. Multiple central managers

In this diagram, the primary central manager is serving Machines A and B. Due to some network failure, Machines C, D, and E have lost contact with the primary central manager machine and, therefore, Machine C which is authorized to serve as an alternate central manager, assumes that role. Machine C remains as the alternate central manager until either:

- The primary central manager is able to contact Machines C, D, and E. In this case, the primary central manager notifies the alternate central managers that it is operating again and, therefore, Machine C terminates its negotiator daemon. The negotiator daemon running on the primary central manager machine is refreshed to discard any old job status information and to pick up the new job status information from the newly rejoined machines.

- Machine C loses contact with Machines D and E. In this case, if machine D or E is authorized to act as an alternate central manager, it assumes that role. Otherwise, there will be no central manager serving these machines. Note that Machine C remains as its own central manager.

While LoadLeveler can handle this situation of two concurrent central managers without any loss of integrity, some installations may find administering it somewhat confusing. To avoid any confusion, you should specify all primary and alternate central managers on the same LAN segment.

For information on selecting alternate central managers, refer to “Defining machines” on page 78.

How do I recover resources allocated by a Schedd machine?

If a node running the Schedd daemon fails, resources allocated to jobs scheduled by this Schedd cannot be freed up until you restart the Schedd. Administrators must do the following to enable the recovery of Schedd resources:

1. Recognize that a node running the Schedd daemon is down and will be down long enough such that it is necessary for you to recover the Schedd resources.
2. Add the statement **schedd_fenced=true** to the machine stanza of the failed node. This statement specifies that the central manager ignores connections from the Schedd daemon running on this machine, and prevents conflicts from arising when a Schedd machine is restarted while a purge is taking place.
3. Reconfigure the central manager node so that it recognizes the “fenced” Schedd daemon. From the central manager machine issue **llctl reconfig**.
4. Issue the **llmovespool -d spool_directory -h target_schedd_hostname** command to move a job queue database to another Schedd within a local cluster.
5. Remove all files in the LoadLeveler spool directory of the failed node. Once the failed node is working again, you can remove the **schedd_fenced=true** statement.

For more information, see “ll_move_spool subroutine” on page 634 and “Procedure for recovering a job spool” on page 154.

Why can't I find a core file on Linux?

On Linux, when a LoadLeveler daemon terminates abnormally a core file is not generated. Why? Although a LoadLeveler daemon begins its existence as a root process, it uses the system functions **seteuid()** and **setegid()** to switch to effective user ID of **loadl** and effective group ID of **loadl** immediately after startup if the file **/etc/LoadL.cfg** is not defined. If this file is defined, the user ID associated with the **LoadLUserid** keyword and the group ID associated with the **LoadLGroupid** keyword are used instead of the default **loadl** user and group IDs.

On Linux systems, unless the default kernel runtime behavior is modified, the standard kernel action for a process that has successfully invoked **seteuid()** and **setegid()** to have a different effective user ID and effective group ID is not to dump a core file. So, if you want Linux to create a core file when a LoadLeveler daemon terminates abnormally you must use the file **/etc/LoadL.cfg** to set both **LoadLUserid** and **LoadLGroupid** to **root**.

On Red Hat Enterprise Linux 3.3 systems, the command **sysctl -w kernel.core_setuid_ok=1** can be used to change the default kernel core file creation behavior of **setuid** programs. If the **core_setuid_ok** option is enabled, the values of

`LoadLuserid` and `LoadLgroupid` in the `/etc/LoadL.cfg` file do not have to be `root` for the successful creation of LoadLeveler core files.

Why am I seeing inconsistencies in my `llfs` output?

Generally, the sum of the used shares by all users and the sum of the used shares by all LoadLeveler groups should have similar values. If there is a large (much larger than the number of entries in the `llfs` output) and increasing difference between the two sums, the cluster should be checked to make sure that all nodes have the same clock time. Large time differences among nodes in a cluster could lead to errors in the `llfs` output.

What happens if errors are found in my configuration or administration file?

When errors are found during administration file processing, processing continues in nearly all cases. Because processing continues, it is possible that even though the administration file has been read, it might not resemble the intended configuration. This is especially true in cases where opening and closing braces are mismatched. It is possible for the parser to interpret stanzas as substanzas of another stanza, and when this happens, those stanzas are effectively ignored. Consider these cases:

- A machine stanza is interpreted to be a stanza within a class stanza. Because class stanzas only support substanzas of the user type, the machine stanza is completely ignored.
- A user stanza is incorrectly interpreted to be a stanza within a class. Although this is valid, that user stanza will not exist on its own, but will instead be part of the class stanza, which was not the administrator's intent.

It is difficult to determine whether an error in the administration file will completely change the meaning of the file or if the error will effect only a single keyword value. Because it is not necessarily desirable to shutdown LoadLeveler daemons and commands for every possible error, and because the behavior should be consistent, processing will continue. It is important for the administrator to be aware of this behavior and to investigate and repair any configuration errors reported by the `llctl start` command (see “`llctl - Control LoadLeveler daemons`” on page 409 for more information).

Other questions

This topic contains answers to some miscellaneous questions asked by LoadLeveler customers.

Why do I have to `setuid = 0`?

The master daemon starts the `startd` daemon and the `startd` daemon starts the starter process. The starter process runs the job. The job needs to be run by the `userid` of the submitter. You either have to have a separate master daemon running for every ID on the system or the master daemon has to be able to `su` to every `userid` and the only user ID that can `su` any other `userid` is `root`.

Why doesn't LoadLeveler execute my `.profile` or `.login` script?

When you submit a batch job to LoadLeveler, the operating system will execute your `.profile` script before executing the batch job if your login shell is the Korn shell. On the other hand, if your login shell is the Bourne shell, on most operating systems (including AIX), the `.profile` script is not executed. Similarly, if your login shell is the C shell then AIX will execute your `.login` script before executing your LoadLeveler batch job but some other variants of UNIX may not invoke this script.

Troubleshooting

The reason for this discrepancy is due to the interactions of the shells and the operating system. To understand the nature of the problem, examine the following C program that attempts to open a login Korn shell and execute the "ls" command:

```
#include <stdio.h>
main()
{
    execl("/bin/ksh", "-", "-c", "ls", NULL);
}
```

UNIX documentations in general (SunOS, HP-UX, AIX, IRIX) give the impression that if the second argument is "-" then you get a login shell regardless of whether the first argument is /bin/ksh or /bin/csh or /bin/sh. In practice, this is not the case. Whether you get a login shell or not is implementation dependent and varies depending upon the UNIX version you are using. On AIX you get a login shell for /bin/ksh and /bin/csh but not the Bourne shell.

If your login shell is the Bourne shell and you would like the operating system to execute your **.profile** script before starting your batch job, add the following statement to your job command file:

```
# @ shell = /bin/ksh
```

LoadLeveler will open a login Korn shell to start your batch job which may be a shell script of any type (Bourne shell, C shell, or Korn shell) or just a simple executable.

What happens when a mksysb is created when LoadLeveler is running jobs?

When you create a mksysb (an image of the currently installed operating system) at a time when LoadLeveler is running jobs, the state of the jobs is saved as part of the mksysb. When the mksysb is restored on a node, those jobs will appear to be on the node, in the same state as when they were saved, even though the jobs are not actually there. To delete these phantom jobs, you must remove all files from the LoadLeveler **spool** and **execute** directories and then restart LoadLeveler.

What can I do when a reserved node is down?

If the reservation has not started yet, the node might become available before the reservation start time. If the node is still not available when the reservation starts, a LoadLeveler administrator may use the **llchres** command to remove the node and replace it with another.

How do I add or remove a node from the LoadLeveler administration file?

To add or remove a node from the LoadLeveler administration file, you must stop and restart LoadLeveler. Because stopping LoadLeveler will cause all running jobs to be vacated, you might want to drain the cluster before stopping LoadLeveler.

To add or remove nodes from the **LoadL_admin** file, do the following:

1. (**Optional**) Drain all nodes to allow all running jobs to complete by issuing the following command:

```
llctl -g drain startd allclasses
```
2. Stop LoadLeveler by issuing the following command:

```
llctl -g stop
```
3. Add and/or remove nodes from the **LoadL_admin** file.
4. Start LoadLeveler by issuing the following command:

```
llctl -g start
```

Troubleshooting in a multicluster environment

The following subtopics will help you troubleshoot your multicluster environment.

How do I determine if I am in a multicluster environment?

- Issue the `llstatus` command.
 - Output of command will display "Cluster name is *cluster_name*".

How do I determine how my multicluster environment is defined and what are the inbound and outbound hosts defined for each cluster?

- Issue `llstatus -C` command.
 - Output of command will display the local cluster's administration file cluster stanza information.
- Issue `llstatus -X all -C` command.
 - Output of command will display the administration file cluster stanza information for all clusters defined in the local cluster's configuration.

Why is my multicluster environment not enabled?

- Issue `llstatus -X all -C`.
 - The cluster stanzas defined for each cluster participating in the multicluster environment must have the same **outbound_hosts** and **inbound_hosts** defined.
 - Determine if any of the clusters are being started with **SCHEDD_STREAM_PORT** defined. The **inbound_schedd_port** keyword must be set for that cluster.
- Set the **D_MUSTER** debug flag for the **SCHEDD_DEBUG** configuration keyword on the machines defined as **inbound_hosts** and **outbound_hosts**, reconfigure LoadLeveler and examine the SchedLog on those machines for information about configuration errors.
- If the clusters are trying to enable OpenSSL, examine the SchedLog on the **inbound_hosts** and **outbound_hosts** for messages about SSL initialization errors and that multicluster is being disabled.

How do I find log messages from my multicluster defined installation exits?

- Determine which machine is executing the installation exit.
 - For **CLUSTER_METRIC**:
 - If the user specifies the reserved word **any** as the **cluster_list** during job submission, the job is sent to the first outbound Schedd defined for the first configured remote cluster. The **CLUSTER_METRIC** is executed on this Schedd to determine where the job will be distributed. If this Schedd is not the **outbound_hosts schedd** for the assigned cluster, the job will be forwarded to the correct **outbound_hosts schedd**. If the user specifies a list of clusters as the **cluster_list** during job submission, the job is sent to the first outbound Schedd defined for the first specified remote cluster. The **CLUSTER_METRIC** is executed on this Schedd to determine where the job will be distributed. If this Schedd is not the **outbound_hosts schedd** for the assigned cluster, the job will be forwarded to the correct **outbound_hosts schedd**.

- For **CLUSTER_USER_MAPPER**:
 - This installation exit is executed on the **inbound_hosts** of the local cluster when receiving a job submission, move job request or remote command.
- For **CLUSTER_REMOTE_JOB_FILTER**:
 - This installation exit is executed on the **inbound_hosts** of the local cluster when receiving a job submission or move job request.
- Set the **D_MUSTER** debug flag for the **SCHEDD_DEBUG** configuration keyword on the machines defined as **inbound_hosts** and **outbound_hosts**, reconfigure LoadLeveler and examine the SchedLog on those machines for information about configuration errors.

Why won't my remote job be submitted or moved?

- Determine if the remote job filter has changed the number of steps within the job.
 - If the local submission filter on the submitting cluster has added or deleted steps from the original user's job command file, the remote job filter must add or delete the same number of steps. The job command file statements returned by the remote job filter must contain the same number of steps as the job object received from the sending cluster.
- Determine if the job failed the assigned cluster's include and exclude rules for the cluster and/or class stanzas.
 - If the assigned cluster has **CLUSTER_USER_MAPPING** enabled, the mapped user ID is applied to the rules.
- Issue **llq** to determine if the job being moved has all of its steps in an idle-like state.
 - The **llmovejob** command should fail and report this situation.
- Issue **llq -x -d job_id** to determine if the job being moved has a job command file associated with it.
 - A job cannot be moved that was not submitted while in the multicluster environment.
- See "How do I find log messages from my multicluster defined installation exits?" on page 663 to determine if an installation exit has returned an error.
- Determine that the file system in the assigned cluster has the desired availability and permissions.
 - User may be mapped to another user ID thus another **\$HOME**.
 - User needs to have **initialdir** available.
 - **cluster_input_file** and **cluster_output_file** need requested file locations to be available.
 - If clusters share a common file system, users requesting **cluster_input_file** and **cluster_output_file** may have their remote location files removed if a local job is moved to another cluster. During a **llmovejob** operation, the files are copied from the remote location to the remote location instead of from the local location to the remote location. LoadLeveler only knows that the job being moved has access to the remote location because they were copied during the local submission. After the **llmovejob** is complete, LoadLeveler removes the files from the local cluster in the remote location, thus removing the files just copied.
- Determine if the job is an interactive jobs.
 - Interactive jobs may not be submitted to remote clusters.

- If the **llsubmit** or **llmovejob** command times out while waiting for a response from the remote cluster, LoadLeveler is not able to determine if the command was successful and it is recommended that the user issue **llq** to the remote cluster to determine if the job was submitted or moved.

Why did the **CLUSTER_REMOTE_JOB_FILTER** not update the job with all of the statements I defined?

- See the **CLUSTER_REMOTE_JOB_FILTER** configuration file keyword description for a list of keywords that are not changed by the filter.

How do I find my remote job?

- Capture the **stdout** of the **llsubmit** and **llmovejob** commands to see the **outbound_hosts** machine assigned to the job, the **inbound_hosts** machine assigned to the job, the cluster assigned to the job, and the job identifier assigned to the job.
 - The Schedd host represented in the job identifier for remote jobs does not represent the managing Schedd of the job. It represents the Schedd that assigned the job number.
- Issue the **llq -X all** command and search for the desired job identifier.
- Check for pertinent mail messages.
 - If a job has been moved by an administrator, the submitting user will receive mail notification.
 - The job may have completed already. If the user has **notify_user** and **notification** set, mail will indicate job status.

Why won't my remote job run?

If the remote job has been received by the central manager of the remote cluster:

- Follow the troubleshooting tips for local jobs in “Why won't my job run?” on page 652 or “Why won't my parallel job run?” on page 654.
- Use the information from the **llsubmit** and **llq** commands to determine the machines that have processed the job. Examine the Schedd logs on those machines for information relating to the specific job.
- Capture the **stdout** of the **llsubmit** and **llmovejob** commands to see the **outbound_hosts** machine assigned to the job, the **inbound_hosts** machine assigned to the job, the cluster assigned to the job, and the job identifier assigned to the job.

Note: The Schedd host represented in the job identifier for remote jobs does not represent the managing Schedd of the job. It represents the Schedd that assigned the job number.

- Issue **llq -X remote_cluster -l job_ID**.
- Check for the multicluster environment related keywords (see the **llq** command for detailed data descriptions):
 - Scheduling Cluster - what cluster is the job running in.
 - Submitting Cluster - what cluster was the job submitted from
 - Sending Cluster - during move job what cluster did the job come from
 - Requested Cluster - cluster_list specified by user.
 - Schedd History - history of managing Schedds
 - Outbound Schedds - history of outbound Schedds
 - Submitting User - user name that the job was submitted under

- Check for pertinent mail messages.

Why does `llq -X` all show no jobs running when there are jobs running?

- When not using `CLUSTER_USER_MAPPER`, check that the user's uid are the same between the local cluster and remote cluster.

Helpful hints

This topic contains tips on running LoadLeveler, including some productivity aids.

Scaling considerations

If you are running LoadLeveler on a large number of nodes (128 or more), network traffic between LoadLeveler daemons can become excessive to the point of overwhelming a receiving daemon. To reduce network traffic, consider the following daemon, keyword, and command recommendations for large installations.

- Set the `POLLS_PER_UPDATE*POLLING_FREQUENCY` interval to five minutes or more. This limits the volume of machine updates the startd daemons send to the negotiator. For example, set `POLLS_PER_UPDATE` to 10 and set `POLLING_FREQUENCY` to 30 seconds.
- If your installation's mix of jobs includes a high percentage of parallel jobs requiring many nodes, specify `schedd_host=yes` in the machine stanza of each Schedd machine. The Schedd daemons must communicate with hundreds of startd daemons every time a job runs. You can distribute this communication by activating many Schedd daemons. Typically, the number of Schedd machines in a LoadLeveler cluster ranges from 2 to 10, depending on the mix of workload and number of jobs in the system.
- If your installation allows jobs to be submitted from machines running the Schedd daemon, you should consider avoiding "Schedd affinity" by specifying `SCHEDD_SUBMIT_AFFINITY=FALSE` in the LoadLeveler configuration file. By default, the `llsubmit` command submits a job to the machine where the command was invoked provided the Schedd daemon is running on the machine. (This is called Schedd affinity.)
- You can decrease the amount of time the negotiator daemon spends running negotiation loops by increasing the `NEGOTIATOR_INTERVAL` and the `NEGOTIATOR_CYCLE_DELAY`. For example, set `NEGOTIATOR_INTERVAL` to 600, and set `NEGOTIATOR_CYCLE_DELAY` to 30.
- Make sure the machine update interval is not too short by setting the `MACHINE_UPDATE_INTERVAL` to a value larger than three times the polling interval (`POLLS_PER_UPDATE*POLLING_FREQUENCY`). This prevents the negotiator from prematurely marking a machine as "down" or prematurely cancelling jobs.
- In a large LoadLeveler cluster, issuing the `llctl` command with the `-g` can take minutes to complete. To speed this up, set up a working collective containing the machines in the cluster and use the `dsh` command; for example, `dsh llctl reconfig`. This command also allows you to limit your operation to a subset of machines by defining other working collectives.

Hints for running jobs

The following subtopics provide some helpful hints that are useful for running jobs.

Determining when your job started and stopped

By reading the notification mail you receive after submitting a job, you can determine the time the job was submitted, started, and stopped. Suppose you submit a job and receive the following mail when the job finishes:

```
Submitted at: Mon Oct 16 11:40:41 2006
Started   at: Mon Oct 16 11:45:00 2006
Exited    at: Mon Oct 16 12:49:10 2006
```

```
Real Time:  0 01:08:29
Job Step User Time:  0 00:30:15
Job Step System Time:  0 00:12:55
Total Job Step Time:  0 00:43:10
```

```
Starter User Time:  0 00:00:00
Starter System Time:  0 00:00:00
Total Starter Time:  0 00:00:00
```

This mail tells you the following:

Submitted at The time you issued the **llsubmit** command or the time you submitted the job with the graphical user interface.

Started at The time the starter process executed the job.

Exited at The actual time your job completed.

Real Time The wall clock time from submit to completion.

Job Step User Time
The CPU time the job consumed executing in user space.

Job Step System Time
The CPU time the system (AIX) consumed on behalf of the job.

Total Job Step Time
The sum of the **Job Step User Time** and **Job Step System Time** fields.

Starter User Time
The CPU time consumed by the LoadLeveler starter process for this job, executing in user space. Time consumed by the starter process is the only LoadLeveler overhead which can be directly attributed to a user's job.

Starter System Time
The CPU time the system (AIX) consumed on behalf of the LoadLeveler starter process running for this job.

Total Starter Time
The sum of the **Starter User Time** and **Starter System Time** fields.

You can also get the starting time by issuing **llsummary -l -x** and then issuing **awk /Date|Event/** against the resulting file. For this to work, you must have **ACCT = A_ON A_DETAIL** set in the **LoadL_config** file.

Running jobs at a specific time of day

Using a machine's local configuration file, you can set up the machine to run jobs at a certain time of day (sometimes called an *execution window*). The following coding in the local configuration file runs jobs between 5:00 PM and 8:00 AM daily, and suspends jobs the rest of the day:

```
START: (tm_hour >= 1700) || (tm_hour <= 0800)
SUSPEND: (tm_hour > 0800) && (tm_hour < 1700)
CONTINUE: (tm_hour >= 1700) || (tm_hour <= 0800)
```

Controlling the mix of idle and running jobs

The following keywords determine the mix of idle and running jobs for a user or group. These keywords, which are described in detail in “Defining users” on page 91, are:

maxqueued

Controls the number of jobs in any of these states: Idle, Pending, Starting, Running, Preempt Pending, Preempted, Resume Pending, and Checkpointing.

maxjobs

Controls the number of jobs in any of these states: Running, Pending, or Starting; thus it controls a subset of what **maxqueued** controls. The **maxjobs** keyword effectively controls the number of jobs in the Running state, since Pending and Starting are usually temporary states.

Note: The **maxjobs** keyword can also be configured in the class stanza to limit the total number of running job steps of a particular class.

maxidle

Controls the number of jobs in any of these states: Idle, Pending, or Starting; thus it controls a subset of what **maxqueued** controls. The **maxidle** keyword effectively controls the number of jobs in the Idle state, since Pending and Starting are usually temporary states.

Administrators can restrict the number of queued, idle, and running job steps on a per-class, per-user basis. The LoadLeveler administrator specifies the per-class, per-user constraints in the **LoadL_admin** file using user substanzas within each class stanza. For more information about substanzas, see “Defining user substanzas in class stanzas” on page 88.

What happens when you submit a job

For a user’s job to be allowed into the job queue and then dispatched:

- The total of other jobs (in the Idle, Pending, Starting, and Running states) for that user must be less than the **maxqueued** value for that user.
- The total idle jobs (those in the Idle, Pending, and Starting states) must be less than the **maxidle** value for the user.
- Constraints on the group’s jobs and the user’s jobs belonging to a particular class are considered.

Also, if the number of jobs exceeds the value specified by any of these **max** keywords, the job being considered is placed in the Not Queued state until one of the other jobs changes state. If the user is at the **maxqueued** limit, a job must complete, be canceled, or be held before the new job can enter the queue. If the user is at the **maxidle** limit, a job must start running, be canceled, or be held before the new job can enter the queue.

Once a job is in the queue, the job is not taken out of queue unless the user places a hold on the job, the job completes, or the job is canceled. This even applies to a job which is rejected or vacated and returned to the queue in the Idle state. (An exception to this, when you are running the default LoadLeveler scheduler, is parallel jobs which do not accumulate sufficient machines in a given time period. These jobs are moved to the Deferred state, meaning they must vie for the queue when their Deferred period expires.)

Once a job is in the queue, the job will run unless the **maxjobs** limit for the user is at a maximum.

Note the following restrictions for using these keywords:

- If **maxqueued** is greater than (**maxjobs** + **maxidle**), the **maxqueued** value will never be reached.
- If either **maxjobs** or **maxidle** is greater than **maxqueued**, then **maxqueued** will be the only restriction in effect, since **maxjobs** and **maxidle** will never be reached.

Sending output from several job steps to one output file

You can use dependencies in your job command file to send the output from many job steps to the same output file. For example:

```
# @ step_name = step1
# @ executable = ssba.job
# @ output = ssba.tmp
# @ ...
# @ queue
#
# @ step_name = append1
# @ dependency = (step1 != CC_REMOVED)
# @ executable = append.ksh
# @ output = /dev/null
# @ queue
# @
# @ step_name = step2
# @ dependency = (append1 == 0)
# @ executable = ssba.job
# @ output = ssba.tmp
# @ ...
# @ queue
# @
# @ step_name = append2
# @ dependency = (step2 != CC_REMOVED)
# @ executable = append.ksh
# @ output = /dev/null
# @ queue
#
# ...
```

Then, the file **append.ksh** could contain the line **cat ssba.tmp >> ssba.log**. All your output will reside in **ssba.log**. (Your dependencies can look for different return values, depending on what you need to accomplish.)

You can achieve the same result from within **ssba.job** by appending your output to an output file rather than writing it to **stdout**. Then your output statement for each step would be **/dev/null** and you wouldn't need the append steps.

Hints for using machines

The following subtopics provide some helpful hints for using machines.

Setting up a single machine to have multiple job classes

You can define a machine to have multiple job classes which are active at different times. For example, suppose you want a machine to run jobs of Class A any time, and you want the same machine to run Class B jobs between 6 p.m. and 8 a.m.

You can combine the **Class** keyword with a user-defined macro (called **Off_shift** in this example).

Troubleshooting

For example:

```
Off_Shift = ((tm_hour >= 18) || (tm_hour < 8))
```

Then define your START statement:

```
START : (Class == "A") || ((Class == "B") && $(Off_Shift))
```

Make sure you have the parenthesis around the **Off_Shift** macro, since the logical OR has a lower precedence than the logical AND in the START statement.

Also, to take weekends into account, code the following statements. Remember that Saturday is day 6 and Sunday is day 0.

```
Off_Shift = ((tm_wday == 6) || (tm_wday == 0) || (tm_hour >=18) \
|| (tm_hour < 8))
```

```
Prime_Shift = ((tm_wday != 6) && (tm_wday != 0) && (tm_hour >= 8) \
&& (tm_hour < 18))
```

Reporting the load average on machines

You can use the **/usr/bin/rup** command to report the load average on a machine. The **rup machine_name** command gives you a report that looks similar to the following:

```
localhost    up 23 days, 10:25,    load average: 1.72, 1.05, 1.17
```

You can use this command to report the load average of your local machine or of remote machines. Another command, **/usr/bin/uptime**, returns the load average information for only your local host.

History files and Schedd

The **Schedd** daemon writes to the spool/history file only when a job is completed or removed. Therefore, you can delete the history file and restart **Schedd** even when some jobs are scheduled to run on other hosts.

However, you should clean up the **spool/job_queue.dir** and **spool/job_queue.pag** files only when no jobs are being scheduled on the machine.

You should not delete these files if there are any jobs in the job queue that are being scheduled from this machine (for example, jobs with names such as *thismachine.clusterno.jobno*).

For fair share scheduling, **Schedd** daemons store historic CPU data for users and groups when their jobs terminate. Usually, a LoadLeveler cluster has more than one **Schedd** daemon. Each **Schedd** daemon only saves its own portion of the historic CPU data. The following database files in the directory specified by the **SPOOL** keyword on each Schedd machine contain the historic CPU data:

- fair_share_queue.dir
- fair_share_queue.pag

Getting help from IBM

Should you require help from IBM in resolving a LoadLeveler problem, you can get assistance by calling IBM Support. Before you call, be sure you have the following information:

1. Your access code (customer number).
2. The LoadLeveler product number.
3. The name and version of the operating system you are using.

4. A telephone number where you can be reached.

In addition, issue the following command:

```
11ct1 version
```

This command will provide you with code level information. Provide this information to the IBM representative.

The number for IBM support in the United States is 1-800-IBM-4YOU (426-4968).

The Facsimile number is 800-2IBM-FAX (2426-329).

Appendix B. Sample command output

The listings included in this topic are for the following commands:

- “llclass - Query class information” on page 403
- “llq - Query job status” on page 449
- “llstatus - Query machine status” on page 477
- “llsummary - Return job resource information for accounting” on page 496

llclass -l command output listing

The following listing shows **llclass -l** in a cluster with class stanzas configured in the **LoadL_admin** file for classes named high, medium, and low:

```
===== Class low =====
      Name: low
      Priority: 30
      Exclude_Users:
      Include_Users:
      Exclude_Groups:
      Include_Groups:
      Admin:
      Max_processors: -1
      Maxjobs: -1
      Resource_requirement: ConsumableVirtualMemory(5.000 mb)
      Class_comment:
      Class_ckpt_dir: /LL/ckptfiles
      Ckpt_limit: undefined, undefined
      Wall_clock_limit: 00:10:00, 00:15:00 (600 seconds, 900 seconds)
      Def_wall_clock_limit: 00:10:00, 00:15:00 (600 seconds, 900 seconds)
      Job_cpu_limit: undefined, undefined
      Cpu_limit: undefined, undefined
      Data_limit: undefined, undefined
      Core_limit: undefined, undefined
      File_limit: undefined, undefined
      Stack_limit: undefined, undefined
      Rss_limit: undefined, undefined
      Nice: 0
      Free_slots: 16
      Maximum_slots: 16
      Max_total_tasks: -1
      Max_proto_instances: 2
      Preempt_class:
      Start_class:
      User default: maxidle(-1) maxqueued(-1) maxjobs(-1) max_total_tasks(-1)
      User libld: maxidle(-1) maxqueued(30) maxjobs(10) max_total_tasks(15)
===== Class high =====
      Name: high
      Priority: 70
      Exclude_Users: loadl
      Include_Users:
      Exclude_Groups: No_Group
      Include_Groups:
      Admin: loadlst
      Max_processors: -1
      Maxjobs: -1
      Resource_requirement:
      Class_comment:
      Class_ckpt_dir:
      Ckpt_limit: undefined, undefined
      Wall_clock_limit: 00:24:00, 00:20:00 (1440 seconds, 1200 seconds)
      Def_wall_clock_limit: 00:24:00, 00:20:00 (1440 seconds, 1200 seconds)
      Job_cpu_limit: undefined, undefined
      Cpu_limit: 00:15:00, undefined (900 seconds, undefined)
      Data_limit: 10.000 mb, 8.000 mb (10485760 bytes, 8388608 bytes)
      Core_limit: unlimited, undefined
      File_limit: 5.000 gb, 4.000 gb (5368709120 bytes, 4294967296 bytes)
      Stack_limit: unlimited, unlimited
      Rss_limit: unlimited, unlimited
      Nice: 0
      Free_slots: 16
      Maximum_slots: 16
      Max_total_tasks: -1
```

Sample command output

```
Max_proto_instances: 2
  Preempt_class: ALL:VC{medium}
  Start_class:
  User default: maxidle(-1) maxqueued(-1) maxjobs(-1) max_total_tasks(-1)
----- Class medium -----
  Name: medium
  Priority: 50
  Exclude_Users:
  Include_Users: llbld
  Exclude_Groups:
  Include_Groups:
  Admin:
  Max_processors: -1
  Maxjobs: -1
Resource_requirement: donuts(10) widgets(5)
  Class_comment:
  Class_ckpt_dir:
  Ckpt_limit: undefined, undefined
  Wall_clock_limit: 12:00:00, 12:00:00 (43200 seconds, 43200 seconds)
  Def_wall_clock_limit: 12:00:00, 12:00:00 (43200 seconds, 43200 seconds)
  Job_cpu_limit: undefined, undefined
  Cpu_limit: 02:00:00, undefined (7200 seconds, undefined)
  Data_limit: undefined, undefined
  Core_limit: undefined, undefined
  File_limit: undefined, undefined
  Stack_limit: undefined, undefined
  Rss_limit: undefined, undefined
  Nice: 0
  Free_slots: 16
  Maximum_slots: 16
  Max_total_tasks: -1
Max_proto_instances: 2
  Preempt_class: ENOUGH:SU{low}
  Start_class:
  User default: maxidle(-1) maxqueued(40) maxjobs(-1) max_total_tasks(20)
-----
```

llq -l command output listing

The following listing shows the `llq -l` output for a POE Parallel non-checkpointing job step:

```
===== Job Step e189f2rp01.ppd.pok.ibm.com.66.0 =====
  Job Step Id: e189f2rp01.ppd.pok.ibm.com.66.0
  Job Name: multi_lapi_mpi
  Step Name: 0
  Structure Version: 10
  Owner: llbld
  Queue Date: Tue Aug 29 14:40:47 EDT 2006
  Status: Running
  Reservation ID: e189f2rp02.ppd.pok.ibm.com.12.r
  Requested Res. ID:
  Scheduling Cluster: CL2
  Submitting Cluster: CL1
  Sending Cluster: CL1
  Requested Cluster: CL2
  Schedd History: e189f2rp01.ppd.pok.ibm.com
  Outbound Schedds: e189f5rp04.ppd.pok.ibm.com
  Submitting User: llbld
  Dispatch Time: Tue Aug 29 14:40:47 EDT 2006
  Completion Date:
  Completion Code:
  Favored Job: No
  User Priority: 50
  user_sysprio: 0
  class_sysprio: 0
  group_sysprio: 0
  System Priority: -3115628
  q_sysprio: -3115628
  Previous q_sysprio: 0
  Notifications: Complete
  Virtual Image Size: 518 kb
  Large Page: N
  Coschedule: no
  SMT required: no
  Checkpointable: no
  Ckpt Start Time:
  Good Ckpt Time/Date:
  Ckpt Elapse Time: 0 seconds
  Fail Ckpt Time/Date:
  Ckpt Accum Time: 0 seconds
```

```

Checkpoint File:
Ckpt Execute Dir:
Restart From Ckpt: no
Restart Same Nodes: no
  Restart: yes
  Preemptable: yes
Preempt Wait Count: 0
Hold Job Until:
  RSet: RSET_NONE
Mcm Affinity Options:
  Env:
    In: /dev/null
    Out: multi.out
    Err: multi.err
Initial Working Dir: /u/11b1d/Checkpoint/cmd
Dependency:
Resources:
  Step Type: General Parallel
  Node Usage: shared
Submitting Host: e189f2rp01.ppd.pok.ibm.com
Schedd Host: e189f2rp01.ppd.pok.ibm.com
Job Queue Key:
  Notify User: 11b1d@e189f2rp01.ppd.pok.ibm.com
  Shell: /bin/ksh
LoadLeveler Group: No_Group
Class: medium
Ckpt Hard Limit: undefined
Ckpt Soft Limit: undefined
Cpu Hard Limit: undefined
Cpu Soft Limit: undefined
Data Hard Limit: undefined
Data Soft Limit: undefined
Core Hard Limit: undefined
Core Soft Limit: undefined
File Hard Limit: undefined
File Soft Limit: undefined
Stack Hard Limit: undefined
Stack Soft Limit: undefined
Rss Hard Limit: undefined
Rss Soft Limit: undefined
Step Cpu Hard Limit: undefined
Step Cpu Soft Limit: undefined
Wall Clk Hard Limit: 00:30:00 (1800 seconds)
Wall Clk Soft Limit: 00:30:00 (1800 seconds)
  Comment: "Parallel MPI and LAPI test"
Account:
  Unix Group: usr
gotiator Messages:
  Bulk Transfer: No
  Step rCxt Blocks: 0
Adapter Requirement: (sn_single,LAPI,US,shared,AVERAGE,instances=1,)\
  (sn_all,MPI,IP,shared,LOW,instances=1,)
  Step Cpus: 0
Step Virtual Memory: 0.000 mb
Step Real Memory: 0.000 mb
-----
Node
----
Name      :
Preferences :
Node minimum : 3
Node maximum : 3
Node actual  : 3
Allocated Hosts : e189f2rp02.ppd.pok.ibm.com::sn0(MPI,IP,-1,Shared,0 rCxt Blks), \
  sn1(MPI,IP,-1,Shared,0 rCxt Blks),sn0(LAPI,US,14,Shared,0 rCxt Blks), \
  sn0(MPI,IP,-1,Shared,0 rCxt Blks),sn1(MPI,IP,-1,Shared,0 rCxt Blks), \
  sn0(LAPI,US,15,Shared,0 rCxt Blks)
  + e189f2rp01.ppd.pok.ibm.com::sn0(MPI,IP,-1,Shared,0 rCxt Blks), \
  sn1(MPI,IP,-1,Shared,0 rCxt Blks),sn0(LAPI,US,34,Shared,0 rCxt Blks), \
  sn0(MPI,IP,-1,Shared,0 rCxt Blks),sn1(MPI,IP,-1,Shared,0 rCxt Blks), \
  sn0(LAPI,US,35,Shared,0 rCxt Blks)
  + e189f2rp03.ppd.pok.ibm.com::sn0(MPI,IP,-1,Shared,0 rCxt Blks), \
  sn1(MPI,IP,-1,Shared,0 rCxt Blks),sn0(LAPI,US,10,Shared,0 rCxt Blks), \
  sn0(MPI,IP,-1,Shared,0 rCxt Blks),sn1(MPI,IP,-1,Shared,0 rCxt Blks), \
  sn0(LAPI,US,11,Shared,0 rCxt Blks)

Master Task
-----
Executable : /usr/bin/poe
Exec Args  : /u/11b1d/bin/multi_lapi_mpi64 1 -ilevel 6 -labelio yes -pmdlog yes
Num Task Inst: 1

```

Sample command output

```
Task Instance: e189f2rp02:-1,
Task
----
Num Task Inst: 6
Task Instance: e189f2rp02:0:sn0(MPI,IP,-1,Shared,0 rCxt Blks),sn1(MPI,IP,-1,Shared,0 rCxt Blks),\
sn0(LAPI,US,14,Shared,0 rCxt Blks),
Task Instance: e189f2rp02:1:sn0(MPI,IP,-1,Shared,0 rCxt Blks),sn1(MPI,IP,-1,Shared,0 rCxt Blks),\
sn0(LAPI,US,15,Shared,0 rCxt Blks),
Task Instance: e189f2rp01:2:sn0(MPI,IP,-1,Shared,0 rCxt Blks),sn1(MPI,IP,-1,Shared,0 rCxt Blks),\
sn0(LAPI,US,34,Shared,0 rCxt Blks),
Task Instance: e189f2rp01:3:sn0(MPI,IP,-1,Shared,0 rCxt Blks),sn1(MPI,IP,-1,Shared,0 rCxt Blks),\
sn0(LAPI,US,35,Shared,0 rCxt Blks),
Task Instance: e189f2rp03:4:sn0(MPI,IP,-1,Shared,0 rCxt Blks),sn1(MPI,IP,-1,Shared,0 rCxt Blks),\
sn0(LAPI,US,10,Shared,0 rCxt Blks),
Task Instance: e189f2rp03:5:sn0(MPI,IP,-1,Shared,0 rCxt Blks),sn1(MPI,IP,-1,Shared,0 rCxt Blks),\
sn0(LAPI,US,11,Shared,0 rCxt Blks),

1 job step(s) in query, 0 waiting, 0 pending, 1 running, 0 held, 0 preempted
```

llq -l command output listing for a Blue Gene enabled system

The following listing is a fragment of the `llq -l` output for a system where Blue Gene support is enabled and Blue Gene is present:

```
===== Job Step bg1dd1.rchland.ibm.com.37.0 =====
Job Step Id: bg1dd1.rchland.ibm.com.37.0
Job Name: bg1dd1.rchland.ibm.com.37
Step Name: 0
....
Cmd: /usr/bin/mpirun
Args: -exe /test/com -cwd /test -args "-o 100 -b 64 -e 8388608 -n -i -r" -verbose 2
Env:
In: /dev/null
Out: /bg1scratch/vare11a/out.bg1dd1/bg1dd1.R010_J111_128.37.0.out
Err: /bg1scratch/vare11a/out.bg1dd1/bg1dd1.mpirun.R010_J111_128.37.0.out
Initial Working Dir: /test/jcf
Dependency:
Resources:
Preferences:
Step Type: Blue Gene
Size Requested:
Size Allocated:
Shape Requested:
Shape Allocated:
Wiring Requested: MESH
Wiring Allocated:
Rotate: True
Blue Gene Status:
Blue Gene Job Id:
BG Requirements: (Memory == 1024)
Partition Requested: R010_J111_128
Partition Allocated:
Error Text:
....
```

llq -l -x command output listing

The following listing shows the `llq -l -x` output for a POE Parallel non-checkpointing job step when the LoadLeveler cluster runs with Job Accounting enabled:

```
===== Job Step e189f5rp01.ppd.pok.ibm.com.3.0 =====
Job Step Id: e189f5rp01.ppd.pok.ibm.com.3.0
Job Name: btat_MPI/IP/sn_single
Step Name: Step_1
Structure Version: 10
Owner: llbld
Queue Date: Tue Aug 29 12:58:45 EDT 2006
Status: Running
Reservation ID:
Requested Res. ID:
Scheduling Cluster: CL2
Submitting Cluster: CL1
Sending Cluster: CL1
Requested Cluster: CL2
Schedd History: e189f5rp04.ppd.pok.ibm.com
```

```

Outbound Schedds: e189f5rp01.ppd.pok.ibm.com
Submitting User: llbld
Dispatch Time: Tue Aug 29 12:58:47 EDT 2006
Completion Date:
Completion Code:
  Favored Job: No
  User Priority: 50
  user_sysprio: 0
  class_sysprio: 30
  group_sysprio: 65
System Priority:
  q_sysprio:
Previous q_sysprio:
  Notifications: Always
Virtual Image Size: 518 kb
  Large Page: N
  Coschedule: no
  SMT required: no
  Checkpointable: yes
  Ckpt Start Time:
Good Ckpt Time/Date:
  Ckpt Elapse Time: 0 seconds
Fail Ckpt Time/Date:
  Ckpt Accum Time: 0 seconds
  Checkpoint File: /ckptfiles/llbld/btat_MPI/IP/sn_single.e189f5rp01.ppd.pok.ibm.com.3.0.ckpt
  Ckpt Execute Dir:
  Restart From Ckpt: no
  Restart Same Nodes: no
    Restart: yes
  Preemptable: yes
  Preempt Wait Count: 0
  Hold Job Until:
    RSet: RSET_NONE
Mcm Affinity Options:
  Env: LANG=en_US LOGIN=llbld PATH=./bin:/usr/bin:/etc:/usr/ucb...
  In: /dev/null
  Out: btat.e189f5rp04.3.0.out
  Err: btat.e189f5rp04.3.0.err
Initial Working Dir: /u/llbld/Checkpoint/cmd
  Dependency:
  Resources: ConsumableCpus(4)
  Step Type: General Parallel
  Node Usage: shared
  Submitting Host: e189f5rp01.ppd.pok.ibm.com
  Schedd Host: e189f5rp04.ppd.pok.ibm.com
  Job Queue Key: 000003
  Notify User: llbld
  Shell: /bin/ksh
LoadLeveler Group: No_Group
  Class: medium
  Ckpt Hard Limit: undefined
  Ckpt Soft Limit: undefined
  Cpu Hard Limit: 00:30:00 (1800 seconds)
  Cpu Soft Limit: 00:25:00 (1500 seconds)
  Data Hard Limit: 8.200 gb (8804682956 bytes)
  Data Soft Limit: 7.100 gb (7623566950 bytes)
  Core Hard Limit: 11.200 gb (12025908428 bytes)
  Core Soft Limit: 11.100 gb (11918534246 bytes)
  File Hard Limit: unlimited
  File Soft Limit: unlimited
  Stack Hard Limit: 400.000 mb (419430400 bytes)
  Stack Soft Limit: 300.000 mb (314572800 bytes)
  Rss Hard Limit: 15.200 gb (16320875724 bytes)
  Rss Soft Limit: 15.100 gb (16213501542 bytes)
Step Cpu Hard Limit: undefined
Step Cpu Soft Limit: undefined
Wall Clk Hard Limit: 00:20:00 (1200 seconds)
Wall Clk Soft Limit: 00:20:00 (1200 seconds)
  Comment: "BTAT test running MPI/IP over switch"
  Account:
  Unix Group: usr
  User Space Windows: 0
Negotiator Messages:
  Bulk Transfer: No
  Step rCxt Blocks: 0
Adapter Requirement: (sn_single,MPI,IP,shared,LOW,instances=1),(sn_single,LAPI,US,shared,LOW,\
  instances=1,)
  Step Cpus: 4
Step Virtual Memory: 0.000 mb
Step Real Memory: 0.000 mb
----- Detail for e189f5rp01.ppd.pok.ibm.com.3.0 -----
  Running Host: e189f5rp04.ppd.pok.ibm.com
  Machine Speed: 1.000000
  Starter User Time: 00:00:00.164480
  Starter System Time: 00:00:00.066182
  Starter Total Time: 00:00:00.230662

```

Sample command output

```
Starter maxrss: 1888
Starter ixrss: 7272
Starter idrss: 13868
Starter isrss: 0
Starter minflt: 0
Starter majflt: 0
Starter nswap: 0
Starter inblock: 0
Starter oublock: 0
Starter msgsnd: 0
Starter msgrcv: 0
Starter nsignals: 3
Starter nvcsw: 295
Starter nivcsw: 0
Step User Time: 00:03:02.246821
Step System Time: 00:00:59.628595
Step Total Time: 00:04:01.875416
Step maxrss: 6308
Step ixrss: 709692
Step idrss: 105216304
Step isrss: 0
Step minflt: 1858
Step majflt: 0
Step nswap: 0
Step inblock: 0
Step oublock: 0
Step msgsnd: 0
Step msgrcv: 0
Step nsignals: 0
Step nvcsw: 5200734
Step nivcsw: 3311517
-----
Node
----
Name          :
Preferences   :
Node minimum  : 1
Node maximum  : 1
Node actual   : 1
Allocated Hosts : e189f5rp04.ppd.pok.ibm.com:RUNNING:sn0(MPI,IP,-1,Shared,0 rCxt Blks), \
                  sn0(MPI,IP,-1,Shared,0 rCxt Blks),sn0(MPI,IP,-1,Shared,0 rCxt Blks), \
                  sn0(MPI,IP,-1,Shared,0 rCxt Blks)

Master Task
-----
Executable    : /bin/poe
Exec Args     : /u/11b1d/bin/btat -d 240 -v -ilevel 6 -labelio yes -pmdlog yes
Num Task Inst: 1
Task Instance: e189f5rp04:-1,

Task
----
Num Task Inst: 4
Task Instance: e189f5rp04:0:sn0(MPI,IP,-1,Shared,0 rCxt Blks),sn0(LAPI,US,-1,shared,0 rCxt Blks),
Task Instance: e189f5rp04:1:sn0(MPI,IP,-1,Shared,0 rCxt Blks),sn0(LAPI,US,-1,shared,0 rCxt Blks),
Task Instance: e189f5rp04:2:sn0(MPI,IP,-1,Shared,0 rCxt Blks),sn0(LAPI,US,-1,shared,0 rCxt Blks),
Task Instance: e189f5rp04:3:sn0(MPI,IP,-1,Shared,0 rCxt Blks),sn0(LAPI,US,-1,shared,0 rCxt Blks),

1 job step(s) in queue, 0 waiting, 0 pending, 1 running, 0 held, 0 preempted
```

llstatus -l command output listing

The following listing shows the output from `llstatus -l` on a machine connected to a switch network:

```
=====
Name          = e189f5rp01.ppd.pok.ibm.com
Machine       = e189f5rp01.ppd.pok.ibm.com
Arch          = R6000
OpSys        = AIX53
SYSPRIO      = 0
MACHPRIO     = (0 - LoadAvg)
VirtualMemory = 388700 kb
Disk         = 2416272 kb
KeyboardIdle  = 1
Tmp          = 1452264 kb
LoadAvg      = 0.016190
ConfiguredClasses = large(64) No_Class(120) preempt(16) preemptable(16) high(32) medium(32) low(32) super(16)
AvailableClasses = large(64) No_Class(116) preempt(16) preemptable(16) high(32) medium(32) low(32) super(16)
DrainingClasses =
DrainedClasses = super
Pool         = 1 3
```

```

FabricConnectivity = 2:1,1:1
Adapter            = networks(striped,,,-1,16/16,798/798 rCxt Blks,11,READY) network2 \
                  (aggregate,,,-1,16/16,798/798 rCxt Blks,1,READY) \
                  network1(aggregate,,,-1,16/16,798/798 rCxt Blks,1,READY) \
                  en0(ethernet,e189f5rp01.ppd.pok.ibm.com,9.114.170.97,) \
                  ml0(multilink,e189f5rp01ml0.ppd.pok.ibm.com,10.10.10.15,)
Feature            =
Max_Starters      = 480
Prestarted_Starters = 1
Total Memory      = 30720 mb
Memory            = 26624 mb
FreeRealMemory    = 19854 mb
LargePageSize     = 16.000 mb
LargePageMemory   = 4.000 gb
FreeLargePageMemory = 3.109 gb
PagesFreed        = 0
PagesScanned      = 0
PagesPagedIn     = 0
PagesPagedOut     = 0
ConsumableResources = ConsumableCpus(2,6) ConsumableMemory(25.219 gb,26.000 gb) RDMA*(4,4)+<
ConfigTimeStamp   = Tue Aug 29 09:47:50 EDT 2006
Cpus              = 6
RSetSupportType   = RSET_NONE
Speed             = 1.000000
Subnet            = 9.114.170
MasterMachPriority = 0.000000
CustomMetric      = 1
StartdAvail      = 1
State             = Running
EnteredCurrentState = Tue Aug 29 09:58:26 EDT 2006
START             = T
SUSPEND           = F
CONTINUE          = T
VACATE            = F
KILL              = F
Machine Mode      = general
Running           = 4
ScheddAvail      = 1
ScheddState       = Avail
ScheddRunning    = 1
Pending           = 0
Starting          = 0
Idle              = 0
Unexpanded        = 0
Held              = 0
Removed           = 0
RemovedPending   = 0
Completed         = 1
TotalJobs         = 2
Running Steps     = e189f5rp01.ppd.pok.ibm.com.186.0
ReservationPermitted= T
Reservations      = e189f5rp01.ppd.pok.ibm.com.185.r
SMT               = Not Supported
TimeStamp         = Tue Aug 29 09:58:30 EDT 2006

```

llstatus -l -b command output listing

The following listing shows the output from **llstatus -l -b** command:

```

Total Blue Gene Base Partitions  8
Total Blue Gene Compute Nodes    4096
Machine Size in Base Partitons   X=1 Y=2 Z=4
Machine Size in Compute Nodes    X=8 Y=16 Z=32

```

-- list of base partitions --

```
Z = 3
=====
```

```

+-----+
|      |
| 1    | R021
|      | <none>
|      | <none>
|-----|
| 0    | R001
|      | <none>
|      | <none>
+-----+

```

```
Z = 2
=====
```

Sample command output

```

+-----+
|      | R031 |
| 1    | <sdb> |
|      | *     |
+-----+
|      | R011 |
| 0    | <none>|
|      | <none>|
+-----+

```

Z = 1

=====

```

+-----+
|      | R030 |
| 1    | <none>|
|      | <none>|
+-----+
|      | R010 |
| 0    | <sdb> |
|      | *     |
+-----+

```

Z = 0

=====

```

+-----+
|      | R020 |
| 1    | <none>|
|      | <none>|
+-----+
|      | R000 |
| 0    | <none>|
|      | <none>|
+-----+

```

-- list of switches --

```

Switch ID: X_R000
Switch State: UP
Base Partition: R000
Switch Dimension: X
Switch Connections:
  FromPort=PORT_S0 ToPort=PORT_S1 PartitionState=FREE Partition=DD1FULL
Switch ID: X_R001
Switch State: UP
Base Partition: R001
Switch Dimension: X
Switch Connections:
  FromPort=PORT_S0 ToPort=PORT_S1 PartitionState=FREE Partition=DD1FULL
Switch ID: X_R010
Switch State: UP
Base Partition: R010
Switch Dimension: X
Switch Connections:
  FromPort=PORT_S0 ToPort=PORT_S1 PartitionState=FREE Partition=DD1FULL

```

.....

-- list of wires --

```

Wire Id: R000X_R000
Wire State: UP
FromComponent=R000 FromPort=MINUS_X

```

```

ToComponent=X_R000 ToPort=PORT_S0
PartitionState=FREE Partition=DD1FULL
Wire Id: R000Y_R000
Wire State: UP
FromComponent=R000 FromPort=MINUS_Y
ToComponent=Y_R000 ToPort=PORT_S0
PartitionState=FREE Partition=DD1FULL
Wire Id: R000Z_R000
Wire State: UP
FromComponent=R000 FromPort=MINUS_Z
ToComponent=Z_R000 ToPort=PORT_S0
PartitionState=FREE Partition=DD1FULL
Wire Id: R001X_R001
Wire State: UP
FromComponent=R001 FromPort=MINUS_X
ToComponent=X_R001 ToPort=PORT_S0
PartitionState=FREE Partition=DD1FULL
Wire Id: R001Y_R001
Wire State: UP
FromComponent=R001 FromPort=MINUS_Y
ToComponent=Y_R001 ToPort=PORT_S0
PartitionState=FREE Partition=DD1FULL
Wire Id: R001Z_R001
Wire State: UP
FromComponent=R001 FromPort=MINUS_Z
ToComponent=Z_R001 ToPort=PORT_S0
PartitionState=FREE Partition=DD1FULL

```

llsummary -l -x command output listing

The following listing is a fragment of the **llsummary -l -x** output for a POE Parallel job step submitted from LoadLeveler CLUSTER2 to LoadLeveler CLUSTER1:

```

===== Job e189f5rp01.ppd.pok.ibm.com.1 =====
      Job Id: e189f5rp01.ppd.pok.ibm.com.1
      Job Name: btat_MPI/IP/sn_single
      Structure Version: 210
      Owner: llbld
      Unix Group: usr
      Submitting Host: e189f5rp01.ppd.pok.ibm.com
      Submitting Userid: 602009
      Submitting Groupid: 100
      Scheduling Cluster: CLUSTER2
      Submitting Cluster: CLUSTER1
      Sending Cluster: CLUSTER1
      Submitting User: llbld
      Schedd History: e189f5rp04.ppd.pok.ibm.com
      Outbound Schedds: e189f5rp01.ppd.pok.ibm.com
      Number of Steps: 1
----- Step e189f5rp01.ppd.pok.ibm.com.1.0 -----
      Job Step Id: e189f5rp01.ppd.pok.ibm.com.1.0
      Step Name: Step_1
      Queue Date: Tue Aug 29 12:42:41 EDT 2006
      Job Accounting Key: 4968717789226364547
      Dependency:
      Status: Completed
      Dispatch Time: Tue Aug 29 12:42:43 EDT 2006
      Start Time: Tue Aug 29 12:42:43 EDT 2006
      Completion Date: Tue Aug 29 12:43:20 EDT 2006
      Completion Code: 0
      Start Count: 1
      User Priority: 50
      user_sysprio: 0
      class_sysprio: 40
      group_sysprio: 40
      Notifications: Always
      Virtual Image Size: 518 kb
      Checkpointable: yes

```

Sample command output

```
Good Ckpt Time/Date:
  Ckpt Accum Time: 0 seconds
  Checkpoint File: /ckptfiles/11bld/btat_MPI/IP/sn_single.e189f5rp01.ppd.pok.ibm.com.1.0.ckpt
  Restart From Ckpt: no
  Restart Same Nodes: no
    Restart: yes
  Hold Job Until:
    RSet: RSET_NONE
Mcm Affinity Options:
  Cmd: /bin/poe
  Args: /u/11bld/bin/btat -d 30 -v -ilevel 6 -labelio yes -pmdlog yes
  Env: LANG=en_US; LOGIN=11bld; PATH=./bin:/usr/bin:...
  In: /dev/null
  Out: btat.e189f5rp04.1.0.out
  Err: btat.e189f5rp04.1.0.err
Initial Working Dir: /u/11bld/Checkpoint/cmd
  Requirements: (Arch == "R6000") && (OpSys == "AIX53")
  Preferences:
    Step Type: General Parallel
  Min Processors: 1
  Max Processors: 1
  Allocated Host: e189f5rp04.ppd.pok.ibm.com
  Node Usage: shared
  Reservation ID:
  Notify User: 11bld
  Shell: /bin/ksh
LoadLeveler Group: No_Group
  Class: No_Class
  Ckpt Hard Limit: undefined
  Ckpt Soft Limit: undefined
  Cpu Hard Limit: 00:30:00 (1800 seconds)
  Cpu Soft Limit: 00:25:00 (1500 seconds)
  Data Hard Limit: 8.200 gb (8804682956 bytes)
  Data Soft Limit: 7.100 gb (7623566950 bytes)
  Core Hard Limit: 11.200 gb (12025908428 bytes)
  Core Soft Limit: 11.100 gb (11918534246 bytes)
  File Hard Limit: unlimited
  File Soft Limit: unlimited
  Stack Hard Limit: 400.000 mb (419430400 bytes)
  Stack Soft Limit: 300.000 mb (314572800 bytes)
  Rss Hard Limit: 15.200 gb (16320875724 bytes)
  Rss Soft Limit: 15.100 gb (16213501542 bytes)
  Step Cpu Hard Limit: undefined
  Step Cpu Soft Limit: undefined
  Wall Clk Hard Limit: 00:20:00 (1200 seconds)
  Wall Clk Soft Limit: 00:20:00 (1200 seconds)
  Comment: "BTAT test running MPI/IP over switch"
  Account:
  Job Tracking Exit:
  Job Tracking Args:
  Task_geometry:
  Resources:
    Blocking: UNSPECIFIED
Adapter Requirement:
  Step Cpus: 1
  Step Virtual Memory: 0.000 mb
  Step Real Memory: 0.000 mb
  Large Page: N
  Bulk Transfer: No
  Step rCxt Blocks: 0
----- Detail for e189f5rp01.ppd.pok.ibm.com.1.0 -----
  Running Host: e189f5rp04.ppd.pok.ibm.com
  Machine Speed: 1.000000
  Event: System
  Event Name: started
  Time of Event: Tue Aug 29 12:42:43 EDT 2006
  Starter User Time: 00:00:00.000000
  Starter System Time: 00:00:00.000000
  Starter Total Time: 00:00:00.000000
  Starter maxrss: 0
  Starter ixrss: 0
  Starter idrss: 0
  Starter isrss: 0
  Starter minflt: 0
  Starter majflt: 0
  Starter nswap: 0
```

```

Starter inblock: 0
Starter ouble: 0
Starter msgsnd: 0
Starter msgrcv: 0
Starter nsignals: 0
Starter nvcsw: 0
Starter nivcsw: 0
Step User Time: 00:00:00.000000
Step System Time: 00:00:00.000000
Step Total Time: 00:00:00.000000
Step maxrss: 0
Step ixrss: 0
Step idrss: 0
Step isrss: 0
Step minflt: 0
Step majflt: 0
Step nswap: 0
Step inblock: 0
Step ouble: 0
Step msgsnd: 0
Step msgrcv: 0
Step nsignals: 0
Step nvcsw: 0
Step nivcsw: 0
Event: System
Event Name: completed
Time of Event: Tue Aug 29 12:43:20 EDT 2006
Starter User Time: 00:00:00.159357
Starter System Time: 00:00:00.043124
Starter Total Time: 00:00:00.202481
Starter maxrss: 2300
Starter ixrss: 8976
Starter idrss: 15224
Starter isrss: 0
Starter minflt: 0
Starter majflt: 0
Starter nswap: 0
Starter inblock: 0
Starter ouble: 0
Starter msgsnd: 0
Starter msgrcv: 0
Starter nsignals: 4
Starter nvcsw: 69
Starter nivcsw: 3
Step User Time: 00:02:04.062897
Step System Time: 00:00:40.600541
Step Total Time: 00:02:44.663438
Step maxrss: 6328
Step ixrss: 478068
Step idrss: 70409520
Step isrss: 0
Step minflt: 20259
Step majflt: 83
Step nswap: 0
Step inblock: 0
Step ouble: 0
Step msgsnd: 0
Step msgrcv: 0
Step nsignals: 4
Step nvcsw: 3570827
Step nivcsw: 2228899

```

Node

```

Name :
Requirements : (Arch == "R6000") && (OpSys == "AIX53")
Preferences :
Node minimum : 1
Node maximum : 1
Node actual : 1
Allocated Hosts : e189f5rp04.ppd.pok.ibm.com:PENDING:sn0(MPI,IP,-1,Shared,0 rCxt Blks,) \
                  sn0(MPI,IP,-1,Shared,0 rCxt Blks,) \
                  sn0(MPI,IP,-1,Shared,0 rCxt Blks,) \
                  sn0(MPI,IP,-1,Shared,0 rCxt Blks,)

```

Sample command output

Master Task

Executable : /bin/poe
Exec Args : /u/llbld/bin/btat -d 30 -v -ilevel 6 -labelio yes -pmdlog yes
Num Task Inst: 1
Task Instance: e189f5rp04:-1

Task

Num Task Inst: 4
Task Instance: e189f5rp04:0:sn0(MPI,IP,-1,Shared,0 rCxt Blks,)
Task Instance: e189f5rp04:1:sn0(MPI,IP,-1,Shared,0 rCxt Blks,)
Task Instance: e189f5rp04:2:sn0(MPI,IP,-1,Shared,0 rCxt Blks,)
Task Instance: e189f5rp04:3:sn0(MPI,IP,-1,Shared,0 rCxt Blks,)

Appendix C. LoadLeveler port usage

A **port number** is an integer that specifies the port to use to connect to the specified daemon. You can define these port numbers in the configuration file or the `/etc/services` file or you can accept the defaults. LoadLeveler first looks in the configuration file for these port numbers. If LoadLeveler does not find the value in the configuration file, it looks in the `/etc/services` file. If the value is not found in this file, the default is used.

Note: See Table 104 for the configuration file keywords associated with the port numbers.

The first column on each line in Table 104 represents the name of a service. In most cases, these services are also the names of daemons with the following exceptions:

- **LoadL_negotiator_collector** is the service name for a second stream port that is used by the **LoadL_negotiator** daemon.
- **LoadL_schedd_status** is the service name for a second stream port used by the **LoadL_schedd** daemon.

For each LoadLeveler service definition shown in Table 104, the following information is shown:

Service name

Specifies the service name. The service names shown are examples of how the names might appear in the `/etc/services` file.

Port number

Specifies the port number used for the service.

Protocol name

Specifies the transport protocol used for the service.

Source port range

A range of port numbers used on either the client side or daemon (server) side of the service.

Required or optional

Whether or not the service is required.

Description/associated keywords

A short description of the service along with its associated configuration file keyword or keywords.

Table 104. LoadLeveler default port usage

Service name	Port number	Protocol name	Source port range*	Required or optional	Description/associated keywords
LoadL_master	9616	tcp	LB	Required	Master port number for stream port
	9617	udp	LB	Required	Master port number for dgram port
					Keywords: <ul style="list-style-type: none">• MASTER_STREAM_PORT (tcp)• MASTER_DGRAM_PORT (udp)

Table 104. LoadLeveler default port usage (continued)

Service name	Port number	Protocol name	Source port range*	Required or optional	Description/associated keywords
LoadL_negotiator	9614	tcp	LB	Required	Negotiator port number for stream port Keyword: NEGOTIATOR_STREAM_PORT
LoadL_negotiator_collector	9612	tcp	LB	Required	Second negotiator stream port
	9613	udp	LB	Required	Negotiator port number for dgram port Keywords: • CM_COLLECTOR_PORT (tcp) • COLLECTOR_DGRAM_PORT (udp)
LoadL_schedd	9605	tcp	LB	Required	Schedd port number for stream port Keyword: SCHEDD_STREAM_PORT
LoadL_schedd_status	9606	tcp	LB	Required	Schedd stream port for job status data Keyword: SCHEDD_STATUS_PORT
LoadL_startd	9611	tcp	LB	Required	Startd port number for stream port Keyword: STARTD_STREAM_PORT

Note: * A value of LB indicates that the source port range value should be left blank. In other words, no source port range value should be specified.

For more information about configuration file keyword syntax and configuring the LoadLeveler environment, see the following:

- Chapter 4, “Configuring the LoadLeveler environment,” on page 39
- Chapter 12, “Configuration file reference,” on page 243

Accessibility features for TWS LoadLeveler

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in IBM TWS LoadLeveler. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers.
- Keys that are tactilely discernible and do not activate just by touching them.
- Industry-standard devices for ports and connectors.
- The attachment of alternative input and output devices.

Note: The **IBM Cluster Information Center** and its related publications are accessibility-enabled for the IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

Keyboard navigation

This product uses standard Microsoft® Windows® navigation keys.

IBM and accessibility

See the *IBM Accessibility Center* at <http://www.ibm.com/able> for more information about the commitment that IBM has to accessibility.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Intellectual Property Law
2455 South Road, P386
Poughkeepsie, New York 12601-5400
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AFS	PowerPC
AIX	pSeries
AIX 5L	Redbooks
BladeCenter	RS/6000
Blue Gene	SP
DB2	System p
DFS	System p4
eServer	System p5
IBM	System x
IBMLink	Tivoli
LoadLeveler	xSeries
POWER	1350

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Intel, Intel logo, Intel Inside[®], Intel Inside logo, Intel Centrino[™], Intel Centrino logo, Celeron[®], Intel Xeon[™], Intel SpeedStep[®], Itanium[®], and Pentium[®] are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Red Hat, the Red Hat "Shadow Man" logo, and all Red Hat-based trademarks and logos are trademarks or registered trademarks of Red Hat, Inc., in the United States and other countries.

LoadLeveler incorporates Condor, which was developed at the University of Wisconsin-Madison, and uses it with the permission of its authors.

InfiniBand is a registered trademark and service mark of the InfiniBand Trade Association.

Other company, product, or service names may be trademarks or service marks of others.

Glossary

This glossary includes terms and definitions for IBM Tivoli Workload Scheduler (TWS) LoadLeveler. The following cross-references are used in this glossary:

- See** Refers the reader to (a) a term that is the expanded form of an abbreviation or acronym or (b) a synonym or more preferred term.
- See also** Refers the reader to a related term.
- Contrast with** Refers the reader to a term that has an opposed or substantively different meaning.

To view glossaries for other IBM products, go to <http://www-306.ibm.com/software/globalization/terminology/index.html>.

A

AFS. A distributed file system for large networks that is known for its ease of administration and expandability.

AIX. A UNIX operating system developed by IBM that is designed and optimized to run on POWER microprocessor-based hardware such as servers, workstations, and blades.

authentication. The process of validating the identity of a user or server.

authorization. The process of obtaining permission to perform specific actions.

B

Berkeley Load Average. The average number of processes on the operating system's ready-to-run queue.

C

C language. A language used to develop application programs in compact, efficient code that can be run on different types of computers with minimal change.

client. A system or process that is dependent on another system or process (usually called the *server*) to provide it with access to data, services, programs, or resources.

cluster. A collection of complete systems that work together to provide a single, unified computing capability.

D

daemon. A program that runs unattended to perform continuous or periodic functions, such as network control.

DCE. See *Distributed Computing Environment*.

default. Pertaining to an attribute, value, or option that is assumed when none is explicitly specified.

DFS. See *Distributed File System*.

Distributed Computing Environment (DCE). In network computing, a set of services and tools that supports the creation, use, and maintenance of distributed applications across heterogeneous operating systems and networks.

Distributed File Service (DFS). A component of a Distributed Computing Environment (DCE) that enables a single, integrated file system to be shared among all DCE users and host computers in a DCE cell. DFS prevents DCE users from simultaneously modifying the same information.

H

host. A computer that is connected to a network and provides an access point to that network. The host can be a client, a server, or both a client and server simultaneously.

L

LAPI. See *low-level application programming interface*.

low-level application programming interface (LAPI). An IBM message-passing interface that implements a one-sided communication model.

M

MCM. See *multiple chip module*.

memory affinity. A feature available in AIX to allocate memory attached to the same multiple chip module (MCM) on which the process runs. Memory affinity improves the performance of applications on IBM System p servers.

menu. A displayed list of items from which a user can make a selection.

Message Passing Interface (MPI). A library specification for message passing. MPI is a standard application programming interface (API) that can be used with parallel applications and that uses the best features of a number of existing message-passing systems.

Motif. User interface software, from Open Systems Foundation, for use with the X Window System.

MPI. See *Message Passing Interface*.

MPICH. A portable implementation of the Message Passing Interface (MPI).

MPICH-GM. A low-level message-passing system for Myrinet networks.

multiple chip module (MCM). The fundamental, processor, building block of IBM System p servers.

N

network. In data communication, a configuration in which two or more locations are physically connected for the purpose of exchanging data.

Network File System (NFS). A protocol, developed by Sun Microsystems, Incorporated, that enables a computer to access files over a network as if they were on its local disks.

NFS. See *Network File System*.

node. A computer location defined in a network.

P

parameter. A value or reference passed to a function, command, or program that serves as input or controls actions. The value is supplied by a user or by another program or process.

peer domain . A set of nodes configured for high availability by the configuration resource manager. Such a domain has no distinguished or master node. All nodes are aware of all other nodes, and administrative commands can be issued from any node in the domain. All nodes also have a consistent view of the domain membership.

process. A separately executable unit of work.

R

rCxt block. See *remote context blocks*.

RDMA. See *Remote Direct Memory Access*.

Reliable Scalable Cluster Technology (RSCT). A set of software components that together provide a comprehensive clustering environment for AIX and Linux. RSCT is the infrastructure used by a variety of IBM products to provide clusters with improved system availability, scalability, and ease of use.

remote context block (rCxt block). An interprocess communication buffer used by the low-level application programming interface (LAPI) for Remote Direct Memory Access (RDMA).

Remote Direct Memory Access (RDMA). A communication technique in which data is transmitted from the memory of one computer to that of another without passing through a processor. RDMA accommodates increased network speeds.

resource set (RSet). A data structure in AIX 5L used to represent physical resources such as processors and memory. AIX uses resource sets to restrict a set of processes to a subset of the system's physical resources.

RSCT. See *Reliable Scalable Cluster Technology*.

RSCT peer domain. See *peer domain*.

RSet. See *resource set*.

S

server. In a network, hardware or software that provides facilities to clients. Examples of a server are a file server, a printer server, or a mail server.

shell. A software interface between users and an operating system. Shells generally fall into one of two categories: a command line shell, which provides a command line interface to the operating system; and a graphical shell, which provides a graphical user interface (GUI).

SMT. See *symmetric multithreading*.

symmetric multithreading (SMT). Pertaining to a processor design that combines hardware multithreading with superscalar processor technology. Using SMT, a single physical processor emulates multiple processors by enabling multiple threads to issue instructions simultaneously during each cycle.

system administrator. The person who controls and manages a computer system.

T

TCP. See *Transmission Control Protocol*.

Transmission Control Protocol (TCP). A communication protocol used in the Internet and in any network that follows the Internet Engineering Task Force (IETF) standards for internetwork protocol. TCP

provides a reliable host-to-host protocol in packet-switched communication networks and in interconnected systems of such networks.

U

UDP. See *User Datagram Protocol*.

User Datagram Protocol (UDP). An Internet protocol that provides unreliable, connectionless datagram service. It enables an application program on one machine or process to send a datagram to an application program on another machine or process.

W

working directory. The active directory. When a file name is specified without a directory, the current directory is searched.

workstation. A configuration of input/output equipment at which an operator works. A workstation is a terminal or microcomputer at which a user can run applications and that is usually connected to a mainframe or a network.

Index

Special characters

- !var 349
- !var specification
 - on environment keyword 349
- /etc/LoadL.cfg file 40, 65
- .llrc script 8
- \$var specification
 - on environment keyword 349

Numerics

- 64-bit
 - keywords supported
 - administration file 305
 - configuration file 244
 - job command file 335
 - support for accounting functions 61
 - support for GUI 380
 - support for LoadLeveler APIs 505

A

- accessibility 687
 - keyboard 687
 - shortcut keys 687
- account keyword
 - detailed description 306
- account_no keyword
 - detailed description 336
- accounting
 - API 506
 - collecting data 57
 - based on events 59
 - based on machines 58, 126
 - based on user accounts 59
 - for serial or parallel jobs 58
 - correlating AIX and LoadLeveler records 61
 - in job command file 336
 - keywords
 - ACCT 57
 - ACCT_VALIDATION 57
 - GLOBAL_HISTORY 57
 - HISTORY_PERMISSION 57
 - JOB_ACCT_Q_POLICY 57
 - JOB_LIMIT_POLICY 57
 - llacmrg command 387
 - llacctl program 57
 - producing reports 60
 - storing data 60
 - using llsummary command 496
- accounting functions
 - 64-bit support 61
- accounting, job setup 61
- ACCT keyword
 - detailed description 244
- ACCT_VALIDATION keyword
 - detailed description 245
- ACTION_ON_MAX_REJECT keyword
 - detailed description 245

- ACTION_ON_SWITCH_TABLE_ERROR keyword
 - detailed description 245
- adapter
 - dedicated 359
 - shared 359
 - specifying in job command file 356, 362
- adapter stanza keywords
 - adapter_name 306
 - adapter_type 306
 - device_driver_name 313
 - interface_address 319
 - interface_name 319
 - logical_id 319
 - multilink_address 324
 - multilink_list 324
 - network_id 325
 - network_type 325
 - type 331
- adapter stanzas
 - examples 82
 - format 80
- adapter_name keyword
 - detailed description 306
- adapter_stanzas keyword
 - detailed description 306
- adapter_type keyword
 - detailed description 306
- add job to reservation 234
- admin keyword
 - detailed description 307
- ADMIN_FILE 45
- administering LoadLeveler
 - customizing the administration file 77
 - LoadL_admin file 301
 - stanzas 77
- administration file
 - account keyword 306
 - adapter_name keyword 306
 - adapter_stanzas keyword 306
 - adapter_type keyword 306
 - admin keyword 307
 - alias keyword 307
 - central_manager keyword 308
 - ckpt_dir keyword 308
 - ckpt_time_limit keyword 308
 - class_comment keyword 309
 - core_limit keyword 309
 - cpu_limit keyword 309
 - cpu_speed_scale keyword 310
 - customizing 77
 - data_limit keyword 310
 - default_class keyword 310
 - default_group keyword 311
 - default_interactive_class keyword 311
 - default_resources keyword 312
 - device_driver_name keyword 313
 - env_copy keyword 313
 - exclude_classes keyword 313
 - exclude_groups keyword 314
 - exclude_users keyword 315
 - fair_shares keyword 316

- administration file (*continued*)
 - file_limit keyword 316
 - inbound_hosts keyword 316
 - inbound_schedd_port keyword 316
 - include_classes keyword 316
 - include_groups keyword 317
 - include_users keyword 318
 - interface_address keyword 319
 - interface_name keyword 319
 - job_cpu_limit keyword 319
 - keyword descriptions 306
 - local keyword 319
 - logical_id keyword 319
 - machine_mode keyword 320
 - master_node_exclusive keyword 320
 - master_node_requirement keyword 320
 - max_jobs_scheduled keyword 320
 - max_node keyword 321
 - max_processors keyword 321
 - max_protocol_instances keyword 321
 - max_reservation_duration keyword 321
 - max_reservations keyword 322
 - max_top_dogs keyword 322
 - max_total_tasks keyword 322
 - maxidle keyword 323
 - maxjobs keyword 323
 - maxqueued keyword 323
 - multicluster_security keyword 324
 - multilink_address keyword 324
 - multilink_list keyword 324
 - multiple statements 119
 - name_server keyword 325
 - network_id keyword 325
 - network_type keyword 325
 - nice keyword 325
 - outbound_hosts keyword 326
 - pool_list keyword 326
 - port_number keyword 326
 - priority keyword 326
 - reservation_permitted keyword 327
 - resources keyword 327
 - rss_limit keyword 328
 - schedd_fenced keyword 328
 - schedd_host keyword 329
 - secure_schedd_port keyword 329
 - smt keyword 329
 - spacct_exclude_enable keyword 330
 - speed keyword 330
 - ssl_cipher_list keyword 330
 - stack_limit keyword 330
 - structure and syntax 301
 - submit_only keyword 330
 - total_tasks keyword 331
 - type keyword 331
 - wall_clock_limit keyword 331
- administrative actions
 - GUI 158
- advance reservation, Blue Gene support 147
- AFS authentication 246
- AFS authentication installation exit 69
- AFS_GETNEWTOKEN keyword
 - detailed description 246
- AGGREGATE_ADAPTERS keyword 43
 - detailed description 246
- AIX accounting
 - correlating AIX and LoadLeveler records 61

- AIX checkpoint
 - limitations 131
- AIX limitations
 - checkpoint and restart 131
- AIX restart
 - limitations 131
- AIX restrictions
 - checkpoint and restart 131
- alias keyword
 - detailed description 307
- ALLOC_EXCLUSIVE_CPU_PER_JOB keyword
 - detailed description 246
- API scheduler 43
- application programming interface (API)
 - summary 503
- application programming interfaces
 - accessing LoadLeveler objects 522
 - accounting 506
 - checkpointing serial jobs 510
 - job control 619
 - ll_error 590
 - ll_fair_share 592
 - ll_reservation 600
 - process configuration files 519
 - querying jobs and machines 595
 - scheduling 619
 - submitting jobs 614
 - workload management 619
- Arch
 - requirement in job command file 362
- ARCH keyword
 - detailed description 247
- Arch variable
 - detailed description 294
- arguments keyword
 - detailed description 337
- attributes
 - of job steps
 - changing 435
- authentication process, DCE 68
- authentication programs 68

B

- BACKFILL scheduler
 - advantages of using 42
- BACKFILL scheduling
 - avoiding circular preemption 117
 - implied START_CLASS values 118
 - releasing resources of preemptable jobs 119
 - selecting a preemption method 118
- BackgroundLoad keyword 293
- basics, LoadLeveler 3
- batch parallel jobs
 - naming files for checkpointing 135
- BG_ALLOW_LL_JOBS_ONLY keyword
 - detailed description 247
- BG_CACHE_PARTITIONS keyword
 - detailed description 247
- bg_connection keyword
 - detailed description 337
- BG_ENABLED keyword
 - detailed description 247
- BG_MIN_PARTITION_SIZE keyword
 - detailed description 248
- bg_partition keyword
 - detailed description 337

- bg_requirements keyword
 - detailed description 338
- bg_rotate keyword
 - detailed description 338
- bg_shape keyword
 - detailed description 339
- bg_size keyword
 - detailed description 339
- BIN 45
- BIN keyword
 - detailed description 248
- blocking 180
- blocking factor 180
- blocking keyword
 - detailed description 340
- Blue Gene advance reservation support 147
- Blue Gene fair share scheduling support 147
- Blue Gene heterogeneous memory support 147
- Blue Gene object
 - understanding 524
- building a job
 - using the GUI 217
- building jobs
 - using a job command file 165
- bulk data transfer
 - configuring 57
 - specifying for jobs 173
- bulkxfer keyword
 - detailed description 340

C

- Canceled job state
 - abbreviations 18
 - detailed description 18
- cancelling jobs
 - using llcancel 216
 - using the GUI 233
- capture data
 - GUI 160
- central manager 6, 237, 658
 - controlling scheduling cycle
 - example 67
 - local 139
 - querying fair share scheduling information 153
 - remote 139
 - specifying an alternate 44
- central_manager keyword
 - detailed description 308
- CENTRAL_MANAGER_HEARTBEAT_INTERVAL keyword
 - detailed description 248
- CENTRAL_MANAGER_TIMEOUT keyword
 - detailed description 248
- changing attributes of job steps
 - using llmodify command 435
- changing job priority
 - example 215
 - using llprio command 447
 - using the GUI 232
- changing scheduler types
 - example 115
 - reconfiguring 115
- checklist
 - parallel jobs 655
- checkpoint
 - file naming 131
 - removing old files 135
- checkpoint (*continued*)
 - restarting a job 655
- checkpoint and restart limitations, AIX 131
- checkpoint files, removing 135
- checkpoint keyword
 - detailed description 341
- checkpoint keywords
 - summary 129
- checkpoint, take 234
- checkpointing
 - API 510
 - how to checkpoint a job 214
 - naming files for interactive parallel jobs 135
 - naming serial and batch files 135
 - planning considerations 130
 - system-initiated 129, 341
 - user-initiated 129, 341
- checkpointing a job step
 - using llckpt command 400
- Checkpointing job state
 - abbreviations 18
 - detailed description 18
- choice button 221
- circular preemption
 - avoiding 117
- ckpt subroutine 511
- CKPT_CLEANUP_PROGRAM keyword
 - detailed description 248
- ckpt_dir keyword
 - detailed description 308, 341
- ckpt_execute_dir keyword
 - detailed description 342
- CKPT_EXECUTE_DIR keyword
 - detailed description 249
- ckpt_file keyword
 - detailed description 342
- ckpt_time_limit keyword
 - detailed description 308, 343
- class
 - multiple job classes 669
 - querying class information
 - using llclass command 403
- Class
 - defining for a machine 250
 - keyword 250
- class keyword
 - detailed description 343
- CLASS keyword
 - detailed description 249
- Class object
 - understanding 524
- class stanza keywords
 - admin 307
 - ckpt_dir 308
 - class_comment 309
 - core_limit 309
 - cpu_limit 309
 - data_limit 310
 - default_resources 312
 - env_copy_name 313
 - exclude_groups 314
 - exclude_users 315
 - file_limit 316
 - include_groups 317
 - include_users 318
 - job_cpu_limit 319
 - master_node_requirement 320

- class stanza keywords (*continued*)
 - max_node 321
 - max_processors 321
 - max_protocol_instances 321
 - max_top_dogs 322
 - max_total_tasks 322
 - maxjobs 323
 - nice 325
 - priority 326
 - rss_limit 328
 - stack_limit 330
 - total_tasks 331
 - type 331
 - wall_clock_limit 331
- class stanzas
 - defining substanzas 88
 - examples 86
 - format 92
- class_comment keyword
 - detailed description 309
- ClassSysprio variable
 - detailed description 294
 - use on SYSPRIO keyword 287
- CLIENT_TIMEOUT keyword
 - detailed description 250
- cluster
 - definition 3
 - local 139
 - querying multiple clusters 65
 - remote 139
 - submitting jobs to multiple clusters 65
 - various levels of POE 179
- Cluster object
 - understanding 525
- cluster stanza keywords
 - exclude_classes 313
 - exclude_groups 314
 - exclude_users 315
 - inbound_hosts 316
 - inbound_schedd_port 316
 - include_classes 316
 - include_groups 317
 - include_users 318
 - local 319
 - multicluster_security 324
 - outbound_hosts 326
 - secure_schedd_port 329
 - ssl_cipher_list 330
- cluster stanzas
 - examples 94
- cluster with both AIX and Linux machines
 - troubleshooting 656
- cluster_input_file keyword
 - detailed description 344
- cluster_list keyword
 - detailed description 344
- CLUSTER_METRIC keyword
 - detailed description 250
- cluster_output_file keyword
 - detailed description 345
- CLUSTER_REMOTE_JOB_FILTER keyword
 - detailed description 251
- CLUSTER_USER_MAPPER keyword
 - detailed description 252
- CM_COLLECTOR_PORT keyword
 - detailed description 253
- coexistence
 - POE software levels 179
- collect account data
 - GUI 160
- collect reservation data
 - GUI 160
- COMM keyword
 - detailed description 253
- command line interface
 - overview 385
- commands
 - llacmrg 127, 387
 - llbind 203, 389
 - llcancel 392
 - llchres 204, 395
 - llckpt 400
 - llclass 403
 - llclusterauth 408
 - llctl 409
 - lldbconvert 414
 - llexRPD 415
 - llfavorjob 419
 - llfavoruser 421
 - llfs 422
 - llhold 426
 - llinit 429
 - llmkres 200, 431
 - llmodify 435
 - llmovejob 440
 - llmovespool 442
 - llpreempt 444
 - llprio 447
 - llq 203, 204, 205, 449
 - llqres 202, 204, 205, 468
 - llrmres 205, 474
 - llrunscheduler 476
 - llstatus 477
 - llsubmit 203, 494
 - llsummary 496
 - sample output 673
 - summary 385
- commands and APIs, coscheduled job steps 172
- comment keyword
 - detailed description 345
- common name space 79
- communication level 356
- Complete Pending job state
 - abbreviations 18
 - detailed description 18
- Completed job state
 - abbreviations 18
 - detailed description 18
- configuration
 - API 519
- configuration file
 - ACCT keyword 244
 - ACCT_VALIDATION keyword 245
 - ACTION_ON_MAX_REJECT keyword 245
 - ACTION_ON_SWITCH_TABLE_ERROR keyword 245
 - AFS_GETNEWTOKEN keyword 246
 - AGGREGATE_ADAPTERS keyword 246
 - ALLOC_EXCLUSIVE_CPU_PER_JOB keyword 246
 - ARCH keyword 247
 - BG_ALLOW_LL_JOBS_ONLY keyword 247
 - BG_CACHE_PARTITIONS keyword 247
 - BG_ENABLED keyword 247
 - BG_MIN_PARTITION_SIZE keyword 248

configuration file (*continued*)

BIN keyword 248
 CENTRAL_MANAGER_HEARTBEAT_INTERVAL keyword 248
 CENTRAL_MANAGER_TIMEOUT keyword 248
 CKPT_CLEANUP_PROGRAM keyword 248
 ckpt_execute_dir keyword 342
 CKPT_EXECUTE_DIR keyword 249
 CLASS keyword 249
 CLIENT_TIMEOUT keyword 250
 CLUSTER_METRIC keyword 250
 CLUSTER_REMOTE_JOB_FILTER keyword 251
 CLUSTER_USER_MAPPER keyword 252
 CM_COLLECTOR_PORT keyword 253
 COMM keyword 253
 CONTINUE expression 253
 CUSTOM_METRIC keyword 253
 CUSTOM_METRIC_COMMAND keyword 253
 customizing 39
 DCE_AUTHENTICATION_PAIR keyword 254
 DEFAULT_PREEMPT_METHOD keyword 254
 defaults 39
 DRAIN_ON_SWITCH_TABLE_ERROR keyword 255
 ENFORCE_RESOURCE_MEMORY keyword 255
 ENFORCE_RESOURCE_POLICY keyword 255
 ENFORCE_RESOURCE_USAGE keyword 256
 EXECUTE keyword 256
 FAIR_SHARE_INTERVAL keyword 256
 FAIR_SHARE_TOTAL_SHARES keyword 256
 FEATURE keyword 256
 FLOATING_RESOURCES keyword 257
 FS_INTERVAL keyword 257
 FS_NOTIFY keyword 258
 FS_SUSPEND keyword 258
 FS_TERMINATE keyword 259
 GLOBAL_HISTORY keyword 259
 GSMONITOR keyword 259
 GSMONITOR_COREDUMP_DIR keyword 259
 GSMONITOR_DOMAIN keyword 260
 GSMONITOR_RUNS_HERE keyword 260
 HISTORY keyword 260
 HISTORY_PERMISSION keyword 260
 INODE_NOTIFY keyword 260
 INODE_SUSPEND keyword 261
 INODE_TERMINATE keyword 261
 JOB_ACCT_Q_POLICY keyword 262
 JOB_EPILOG keyword 262
 JOB_LIMIT_POLICY keyword 262
 JOB_PROLOG keyword 262
 JOB_USER_EPILOG keyword 262
 JOB_USER_PROLOG keyword 263
 KBDD keyword 263
 KBDD_COREDUMP_DIR keyword 263
 keyword descriptions 244
 KILL expression 263
 LIB keyword 263
 LL_RSH_COMMAND 263
 LOADL_ADMIN keyword 264
 LOCAL_CONFIG keyword 264
 LOG keyword 265
 MACHINE_AUTHENTICATE keyword 265
 MACHINE_UPDATE_INTERVAL keyword 265
 MACHPRIO keyword 265
 MAIL keyword 267
 MASTER keyword 267
 MASTER_COREDUMP_DIR keyword 267
 MASTER_DGRAM_PORT keyword 268

configuration file (*continued*)

MASTER_STREAM_PORT keyword 268
 MAX_CKPT_INTERVAL keyword 268
 MAX_JOB_REJECT keyword 268
 MAX_RESERVATIONS keyword 268
 MAX_STARTERS keyword 269
 MAX_TOP_DOGS keyword 269
 MIN_CKPT_INTERVAL keyword 269
 multiple statements 119
 NEGOTIATOR keyword 269
 NEGOTIATOR_COREDUMP_DIR keyword 270
 NEGOTIATOR_CYCLE_DELAY keyword 270
 NEGOTIATOR_CYCLE_TIME_LIMIT keyword 270
 NEGOTIATOR_INTERVAL keyword 270
 NEGOTIATOR_LOADAVG_INCREMENT keyword 271
 NEGOTIATOR_PARALLEL_DEFER keyword 271
 NEGOTIATOR_PARALLEL_HOLD keyword 271
 NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL keyword 271
 NEGOTIATOR_REJECT_DEFER keyword 272
 NEGOTIATOR_REMOVE_COMPLETED keyword 272
 NEGOTIATOR_RESCAN_QUEUE keyword 272
 NEGOTIATOR_STREAM_PORT keyword 272
 OBITUARY_LOG_LENGTH keyword 273
 POLLING_FREQUENCY keyword 273
 POLLS_PER_UPDATE keyword 273
 PREEMPT_CLASS keyword 274
 PREEMPTION_SUPPORT keyword 275
 PRESTARTED_STARTERS keyword 273
 PROCESS_TRACKING keyword 276
 PROCESS_TRACKING_EXTENSION keyword 276
 PUBLISH_OBITUARIES keyword 276
 REJECT_ON_RESTRICTED_LOGIN keyword 276
 RELEASDIR keyword 277
 RESERVATION_CAN_BE_EXCEEDED keyword 277
 RESERVATION_HISTORY keyword 277
 RESERVATION_MIN_ADVANCE_TIME keyword 277
 RESERVATION_PRIORITY keyword 278
 RESERVATION_SETUP_TIME keyword 278
 RESTARTS_PER_HOUR keyword 278
 RESUME_ON_SWITCH_TABLE_ERROR_CLEAR keyword 279
 RSET_SUPPORT keyword 279
 SAVELOGS keyword 279
 SCHEDD keyword 279
 SCHEDD_COREDUMP_DIR keyword 280
 SCHEDD_INTERVAL keyword 280
 SCHEDD_RUNS_HERE keyword 280
 SCHEDD_STATUS_PORT keyword 281
 SCHEDD_STREAM_PORT keyword 281
 SCHEDD_SUBMIT_AFFINITY keyword 280
 SCHEDULE_BY_RESOURCES keyword 281
 SCHEDULER_TYPE keyword 281
 SEC_ADMIN_GROUP keyword 282
 SEC_ENABLEMENT keyword 282
 SEC_IMPOSED_MECHS keyword 283
 SEC_SERVICES_GROUP keyword 283
 SPOOL keyword 283
 START expression 283
 START_CLASS keyword 284
 START_DAEMONS keyword 284
 STARTD keyword 285
 STARTD_COREDUMP_DIR keyword 285
 STARTD_DGRAM_PORT keyword 285
 STARTD_RUNS_HERE keyword 285
 STARTD_STREAM_PORT keyword 285
 STARTER keyword 286

- configuration file (*continued*)
 - STARTER_COREDUMP_DIR keyword 286
 - structure and syntax 243
 - SUBMIT_FILTER keyword 286
 - SUSPEND expression 287
 - syntax 243
 - SYSPRIO keyword 287
 - SYSPRIO_THRESHOLD_TO_IGNORE_STEP keyword 289
 - TRUNC_GSMONITOR_LOG_ON_OPEN keyword 289
 - TRUNC_KBDD_LOG_ON_OPEN keyword 289
 - TRUNC_MASTER_LOG_ON_OPEN keyword 290
 - TRUNC_NEGOTIATOR_LOG_ON_OPEN keyword 290
 - TRUNC_SCHEDD_LOG_ON_OPEN keyword 290
 - TRUNC_STARTD_LOG_ON_OPEN keyword 290
 - TRUNC_STARTER_LOG_ON_OPEN keyword 290
 - UPDATE_ON_POLL_INTERVAL_ONLY keyword 291
 - user-defined keywords 293
 - VACATE expression 291
 - VM_IMAGE_ALGORITHM keyword 291
 - WALLCLOCK_ENFORCE keyword 292
 - X_RUNS_HERE keyword 292
- configuration file keyword
 - LOADL_ADMIN 41
- configuration file keywords associated with port numbers 685
- configuration files
 - global and local 39
- configuration tasks
 - GUI 159
- configuration wizard
 - lltg 159
- configuring
 - cluster security services 53
 - MPICH jobs under LoadLeveler control 99
 - MPICH-GM jobs 100
 - MVAPICH jobs 100
 - security service 52
- Connectivity
 - requirement in job command file 363
- Connectivity variable
 - detailed description 294
 - use on MACHPRIO keyword 266
- considerations
 - checkpointing 130
 - parallel jobs 96
 - POE 97
 - POE software levels 179
- consumable resources 56
 - introduction 21
 - job scheduling 21
 - modifying using the GUI 233
 - Workload Manager 22
- ConsumableCpus variable
 - detailed description 294
 - use on MACHPRIO keyword 266
- ConsumableMemory variable
 - detailed description 294
 - use on MACHPRIO keyword 266
- ConsumableVirtualMemory variable
 - detailed description 295
 - use on MACHPRIO keyword 266
- CONTINUE expression
 - detailed description 253
- control functions 63
- copy 305, 335
- COPY_ALL specification
 - on environment keyword 349
- core file on Linux
 - troubleshooting 660
- core_limit keyword
 - detailed description 309, 345
- coschedule keyword
 - detailed description 346
- coscheduling job steps 171
 - commands and APIs 172
 - determining priority 171
 - preemption 172
 - submitting 171
 - termination 173
- CPU affinity support
 - overview 137
- CPU_Busy keyword 293
- CPU_Idle keyword 293
- cpu_limit keyword
 - detailed description 309, 346
- cpu_speed_scale keyword
 - detailed description 310
- Cpus variable
 - detailed description 295
 - use on MACHPRIO keyword 266
- create account report
 - GUI 160
- CtSec services 53
- CurrentTime variable
 - detailed description 295
- CUSTOM_METRIC keyword
 - detailed description 253
- CUSTOM_METRIC_COMMAND keyword
 - detailed description 253
- customizing
 - administration file 77
 - configuration file 39, 243
- CustomMetric variable
 - detailed description 295
 - use on MACHPRIO keyword 266

D

- daemon
 - master 8
 - overview 7
- daemons
 - control using llctl command 409
 - gsmonitor 13
 - kbdd 13
 - negotiator 12
 - Schedd 9
 - startd 10
- data access
 - API 522
 - Blue Gene objects
 - understanding 524
 - Class objects
 - understanding 524
 - Cluster objects
 - understanding 525
 - Fairshare objects
 - understanding 525
 - Job objects
 - understanding 525
 - Machine objects
 - understanding 527

- data access (*continued*)
 - MCluster objects
 - understanding 528
 - objects
 - understanding 523
 - Reservations objects
 - understanding 528
 - Wlmstat objects
 - understanding 529
 - data_limit keyword
 - detailed description 310, 346
 - DCE
 - authentication process 68
 - authentication programs 68
 - handling security credentials 68
 - DCE authentication 254
 - DCE_AUTHENTICATION_PAIR keyword
 - detailed description 254
 - debugging
 - controlling output 47
 - dedicated adapters 356
 - default scheduler
 - advantages of using 42
 - default_class keyword
 - detailed description 310
 - default_group keyword
 - detailed description 311
 - default_interactive_class keyword
 - detailed description 311
 - DEFAULT_PREEMPT_METHOD keyword
 - detailed description 254
 - default_resources keyword
 - detailed description 312
 - default_wall_clock_limit keyword
 - detailed description 312
 - Deferred job state
 - abbreviations 18
 - detailed description 18
 - defining classes 83
 - dependency 669
 - dependency keyword
 - detailed description 347
 - details
 - API scheduler 43
 - determining priority for coscheduled job steps 171
 - device_driver_name keyword
 - detailed description 313
 - diagnosing problems 651
 - directories
 - LoadLeveler location after installation 30
 - naming for checkpointing 134
 - submit-only LoadLeveler location after installation 31
 - disability 687
 - Disk
 - requirement in job command file 363
 - Disk variable
 - detailed description 295
 - use on MACHPRIO keyword 266
 - displaying job status
 - using the command llq 215
 - using the GUI 231
 - displaying machine status
 - adapter details 235
 - Blue Gene 236
 - cluster config 236
 - cluster status 235
 - details 235
 - displaying machine status (*continued*)
 - floating resources 235
 - machine resources 235
 - public submit machines 237
 - scheduler in use 237
 - using llstatus 216
 - using the GUI 235
 - domain variable
 - detailed description 295
 - drain
 - GUI 159
 - DRAIN_ON_SWITCH_TABLE_ERROR keyword
 - detailed description 255
 - dsh command 666
- ## E
- editing jobs 171, 229
 - ENFORCE_RESOURCE_MEMORY keyword
 - detailed description 255
 - ENFORCE_RESOURCE_POLICY keyword
 - detailed description 255
 - ENFORCE_RESOURCE_USAGE keyword
 - detailed description 256
 - EnteredCurrentState variable
 - detailed description 295
 - env_copy keyword
 - detailed description 313, 348
 - environment keyword
 - detailed description 349
 - specifications
 - !var 349
 - \$var 349
 - COPY_ALL 349
 - var=value 349
 - environment variable
 - MALLOCTYPE 29, 411, 625
 - environment variables
 - LOADL_JOB_CPU_LIMIT 69
 - LOADL_PROCESSOR_LIST 197
 - LOADL_STEP_CLASS 69
 - LOADL_STEP_COMMAND 69
 - LOADL_STEP_ID 69
 - LOADL_STEP_OWNER 69
 - LOADL_WALL_LIMIT 69
 - epilog programs 70
 - error keyword
 - detailed description 350
 - examples, fair share scheduling 151
 - exclude_classes keyword
 - detailed description 313
 - exclude_groups keyword
 - detailed description 314
 - exclude_users keyword
 - detailed description 315
 - executable 166
 - job command file 169
 - specified in a job command file 165
 - executable keyword
 - detailed description 351
 - EXECUTE 45
 - EXECUTE keyword
 - detailed description 256
 - executing machine 6
 - execution window for jobs 667
 - exit status 361, 494
 - prolog program 73

expressions
CONTINUE 63
KILL 63
START 63
SUSPEND 63
VACATE 63
external scheduler 619

F

fair share scheduling
central manager 153
configuring 148
GUI 161
keywords 148
overview 25
reconfiguring 150
when the Schedd daemons are down 151
when the Schedd daemons are up 151
resetting historic data 153
restoring historic data 153
saving historic data 153
fair share scheduling, Blue Gene support 147
FAIR_SHARE_INTERVAL keyword 148
detailed description 256
FAIR_SHARE_TOTAL_SHARES keyword 148
detailed description 256
fair_shares keyword 148
detailed description 316
Fairshare object
understanding 525
favor jobs 156
llfavorjob command 419
favor users 155
llfavoruser command 421
feature
requirement in job command file 363
FEATURE keyword
detailed description 256
file
customizing administration file 77
customizing configuration file 39
file structure and syntax
administration file 301
file system monitoring 50
file_limit keyword
detailed description 316, 351
files 134
naming checkpoint files 131
naming checkpointing files for interactive parallel jobs 135
naming checkpointing files for serial and batch jobs 135
filtering a job script 70
FLOATING_RESOURCES keyword
detailed description 257
flush
GUI 159
FreeRealMemory variable
detailed description 295
use on MACHPRIO keyword 266
FS_INTERVAL keyword
detailed description 257
FS_NOTIFY keyword
detailed description 258
FS_SUSPEND keyword
detailed description 258

FS_TERMINATE keyword
detailed description 259

G

GetHistory subroutine 507
getting a quick start using the default configuration 27
global configuration file
configuring 39
GLOBAL_HISTORY keyword
detailed description 259
graphical user interface
See GUI 217
group
default 311
UNIX 311
group keyword
detailed description 351
group stanza keywords
env_copy_name 313
exclude_users 315
fair_shares 316
include_users 318
max_node 321
max_processors 321
max_reservation_duration 321
max_reservations 322
max_total_tasks 322
maxidle 323
priority 326
total_tasks 331
type 331
group stanzas
examples 92
format 87
GroupBgSharesExceeded user-defined variable
use on SYSPRIO keyword 149
GroupHasBgShares user-defined variable
use on SYSPRIO keyword 149
GroupHasShares user-defined variable
use on SYSPRIO keyword 149
GroupIsBlueGene variable
use on SYSPRIO keyword 287
GroupQueuedJobs variable
detailed description 295
use on SYSPRIO keyword 287
GroupRemainingBgShares user-defined variable
use on SYSPRIO keyword 149
GroupRemainingShares user-defined variable
use on SYSPRIO keyword 149
GroupRunningJobs variable
detailed description 296
use on SYSPRIO keyword 287
GroupSharesExceeded user-defined variable
use on SYSPRIO keyword 149
GroupSysprio variable
detailed description 296
use on SYSPRIO keyword 287
GroupTotalJobs variable
detailed description 296
use on SYSPRIO keyword 287
GroupTotalShares variable
detailed description 296
GroupUsedBgShares variable
detailed description 296
use on SYSPRIO keyword 287

- GroupUsedShares variable
 - detailed description 296
- gsmonitor daemon 13
- GSMONITOR keyword
 - detailed description 259
- GSMONITOR_COREDUMP_DIR keyword
 - detailed description 259
- GSMONITOR_DOMAIN keyword
 - detailed description 260
- GSMONITOR_RUNS_HERE keyword
 - detailed description 260
- GUI
 - 64-bit support 380
 - configuration tasks 159
 - machine administration 158
 - capture data 160
 - collect account data 160
 - collect reservation data 160
 - configuration tasks 159
 - create account report 160
 - drain 159
 - fair share scheduling 161
 - flush 159
 - move spool 161
 - purge Schedd 160
 - reconfig 158
 - recycle 158
 - resume 159
 - start all 158
 - start Drained 158
 - start LoadLeveler 158
 - stop all 158
 - stop LoadLeveler 158
 - version 161
 - modifying consumable resources and other job attributes 233
- GUI (graphical user interface)
 - customizing 377
 - customizing for the GUI 380
 - help 379
 - menu bar 378
 - overview 377
 - pull-down menus 378
 - starting 377
 - tasks
 - summary 217
 - typographic conventions 379
 - Xloadl file 380
 - Xloadl_so file 380
- GUI (see graphical user interface) 155

H

- help
 - calling IBM 670
 - in the GUI 379
- heterogeneous memory support, Blue Gene 147
- HighLoad keyword 293
- hints for running LoadLeveler 666
- hints for using machines 669
- HISTORY 45
- history file
 - troubleshooting 670
- HISTORY keyword
 - detailed description 260
- HISTORY_PERMISSION keyword
 - detailed description 260

- hold keyword
 - detailed description 351
- holding jobs
 - using llhold 213, 216
 - using the GUI 233
- host variable
 - detailed description 296
- hostname variable
 - detailed description 296
- HOURL keyword 293
- how to checkpoint a job 214

I

- Idle job state
 - abbreviations 18
 - detailed description 18
- idle-like job states 438
- image_size keyword
 - detailed description 352
- implied START_CLASS values 118
- inbound_hosts keyword
 - detailed description 316
- inbound_schedd_port keyword
 - detailed description 316
- include_classes keyword
 - detailed description 316
- include_groups keyword
 - detailed description 317
- include_users keyword
 - detailed description 318
- InfiniBand adapters
 - support for 81
- initialdir keyword
 - detailed description 352
- initiators 269
- INODE_NOTIFY keyword
 - detailed description 260
- INODE_SUSPEND keyword
 - detailed description 261
- INODE_TERMINATE keyword
 - detailed description 261
- input keyword
 - detailed description 353
- instances 183
- integer blocking 180
- integrating LoadLeveler with WLM 127
- interactive jobs
 - planning considerations 97
- interactive parallel jobs
 - naming files for checkpointing 135
- interface
 - application programming (API)
 - summary 503
 - command line
 - overview 385
- interface_address keyword
 - detailed description 319
- interface_name keyword
 - detailed description 319

J

- job
 - accounting 57
 - based on events 59

- job (*continued*)
 - accounting (*continued*)
 - based on machines 58, 126
 - based on user accounts 59
 - for serial or parallel jobs 58
 - storing data 60
 - batch 5
 - building a job command file 165
 - building using the GUI 217
 - canceling 214
 - using llcancel command 392
 - cancelling 233
 - change priority
 - using llprio command 447
 - change step attributes
 - using llmodify command 435
 - class name 343
 - cluster_input_file name 344
 - cluster_list name 344
 - cluster_output_file name 345
 - definition 4
 - diagnosing problems with 652, 654, 656
 - editing 171, 229
 - environment variables 177
 - exit status 361, 494
 - filter 70
 - hold or release
 - using llhold command 426
 - holding 213, 233
 - interactive 97
 - parallel 178, 654
 - preempt a step
 - using llpreempt command 444
 - priority 212, 232, 326
 - query status
 - using llq command 449
 - releasing a hold 233
 - return resource information
 - using llsummary command 496
 - running 666
 - samples 215
 - serial 165, 211
 - status 211, 231, 453
 - submit
 - using llsubmit command 494
 - submit-only 656
 - submitting 165, 177, 211, 230
 - system priority 43
- job accounting setup procedure 61
- job command file
 - account_no keyword 336
 - arguments keyword 337
 - bg_connection keyword 337
 - bg_partition keyword 337
 - bg_requirements keyword 338
 - bg_rotate keyword 338
 - bg_shape keyword 339
 - bg_size keyword 339
 - blocking keyword 340
 - building 165
 - bulkxfer keyword 340
 - checkpoint keyword 341
 - ckpt_dir keyword 341
 - ckpt_file keyword 342
 - ckpt_time_limit keyword 343
 - class keyword 343
 - cluster_input_file keyword 344
 - cluster_list keyword 344
 - cluster_output_file keyword 345
 - comment keyword 345
 - core_limit keyword 345
 - coschedule 346
 - cpu_limit keyword 346
 - data_limit keyword 346
 - default_wall_clock_limit keyword 312
 - dependency keyword 347
 - env_copy keyword 348
 - environment keyword 349
 - error keyword 350
 - example 166, 167, 168, 333
 - executable example 169
 - executable keyword 351
 - file_limit keyword 351
 - group keyword 351
 - hold keyword 351
 - image_size keyword 352
 - initialdir keyword 352
 - input keyword 353
 - job_cpu_limit keyword 353
 - job_name keyword 354
 - job_type keyword 354
 - keyword descriptions 336
 - large_page keyword 354
 - LoadLeveler variables 372
 - max_processors keyword 355
 - mcm_affinity_options keyword 355
 - min_processors keyword 356
 - network keyword 356
 - node keyword 359
 - node_usage keyword 360
 - notification keyword 360
 - notify_user keyword 361
 - output keyword 361
 - parallel 334
 - preferences keyword 361
 - queue keyword 362
 - requirements keyword 362
 - resources keyword 365
 - restart keyword 366
 - restart_from_ckpt keyword 367
 - restart_on_same_nodes keyword 367
 - rset keyword 367
 - run-time environment variables 374
 - serial 333
 - shell keyword 368
 - smt 368
 - stack_limit keyword 369
 - startdate keyword 369
 - step_name keyword 369
 - submitting 177
 - syntax 333
 - task_geometry keyword 370
 - tasks_per_node keyword 370
 - total_tasks keyword 371
 - user_priority keyword 372
 - wall_clock_limit keyword 372
- job files
 - naming for checkpointing 135
 - naming for checkpointing interactive parallel jobs 135
- job object 9
- Job object
 - understanding 525

- job queue
 - definition 7
- job scheduling
 - consumable resources 21
- job spool recovery
 - procedure 154
- job state
 - abbreviations 18
 - descriptions 18
- job step
 - checkpointing
 - using llckpt command 400
- job steps, coscheduled 171
- JOB_ACCT_Q_POLICY keyword
 - detailed description 262
- job_cpu_limit keyword
 - detailed description 319, 353
- JOB_EPILOG keyword
 - detailed description 262
- JOB_LIMIT_POLICY keyword
 - detailed description 262
- job_name keyword
 - detailed description 354
- JOB_PROLOG keyword
 - detailed description 262
- job_type keyword
 - detailed description 354
- JOB_USER_EPILOG keyword
 - detailed description 262
- JOB_USER_PROLOG keyword
 - detailed description 263
- JobIsBlueGene variable
 - detailed description 296
- JobIsNotBlueGene user-defined variable
 - use on SYSPRIO keyword 149
- JobLoad keyword 293

K

- kbdd daemon 13
- KBDD keyword
 - detailed description 263
- KBDD_COREDUMP_DIR keyword
 - detailed description 263
- KeyboardBusy keyword 293
- KeyboardIdle variable
 - detailed description 297
- keywords
 - administration file 78, 306
 - 64-bit support 305
 - checkpoint 129
 - configuration file 44, 243, 244
 - 64-bit support 244
 - LoadLeveler variables 294
 - user-defined 293
 - fair share scheduling 148
 - job command file 336
 - 64-bit support 335
 - user-defined 293, 294
- KILL expression
 - detailed description 263

L

- LAPI 356

- large_page keyword
 - detailed description 354
- LargePageMemory
 - requirement in job command file 363
- LIB 45
- LIB keyword
 - detailed description 263
- libllapi.a library 503
- libllapi.so library 503
- limit keywords 83
- Linux CPU affinity support
 - assigning to application processes 138
 - overview 137
 - troubleshooting 656
- ll_bind subroutine 601
- ll_change_reservation (subroutine)
 - using 204
- ll_change_reservation subroutine 604
- ll_ckpt subroutine 512
- ll_cluster subroutine 620
- ll_cluster_auth subroutine 622
- ll_config_changed subroutine 520
- ll_control subroutine 624
- ll_deallocate subroutine 530
- ll_error 590
- ll_error subroutine 591
- ll_fair_share 592
- ll_fair_share subroutine 593
- ll_free_jobs subroutine 596
- ll_free_nodes subroutine 597
- ll_free_objs subroutine 531
- ll_get_data subroutine 532
- ll_get_jobs subroutine 598
- ll_get_nodes subroutine 599
- ll_get_objs subroutine 574
- ll_init_ckpt subroutine 515
- ll_init_reservation_param subroutine 608
- ll_make_reservation (subroutine)
 - using 201
- ll_make_reservation subroutine 609
- ll_modify subroutine 628
 - using 67
- ll_move_job subroutine 632
- ll_move_spool subroutine 634
- ll_next_obj subroutine 577
- ll_preempt subroutine 637
- ll_preempt_jobs subroutine 639
- ll_query subroutine 578
- ll_read_config subroutine 521
- ll_remove_reservation (subroutine)
 - using 205
- ll_remove_reservation subroutine 612
- ll_reservation 600
- ll_reset_request subroutine 579
- LL_RSH_COMMAND keyword
 - detailed description 263
- ll_run_scheduler subroutine 642
 - using 67
- ll_set_ckpt_callbacks subroutine 517
- ll_set_request subroutine 580
- ll_start_job subroutine 643
- ll_start_job_ext subroutine 645
- ll_terminate_job subroutine 649
- ll_unset_ckpt_callbacks subroutine 518
- LL_Version
 - requirement in job command file 363
- llacctmrg command 387

- llacctmrg command (*continued*)
 - using for reservations 127
- llacctval user exit 509
- llapi.h header file 503
- llbind command 389
 - using to remove a bound job 203
 - using to submit a job 203
- llcancel command 392
- llchres command 395
 - using 204
- llckpt command 400
- llclass -l command sample output listing 673
- llclass command 403
- llclusterauth command 408
- llctl command 409
- lldbconvert command 414
- llextrPD command 415
- llfavorjob command 419
- llfavoruser command 421
- llfree_job_info subroutine 615
- llfs command 422
- llhold command 426
- llinit command 429
- llmkres command 431
 - using 200
- llmodify command 435
 - using 67
- llmovejob command 440
- llmovespool command 442
- llpreempt command 444
- llprio command 447
- llq -l -x command sample output listing 676
- llq -l command sample output listing 674
- llq -l command sample output listing for a Blue Gene enabled system 676
- llq command 449
 - using for reservations 203, 204, 205
- llqres command 468
 - using 202, 204, 205
- llrmres command 474
 - using 205
- llrunscheduler command 476
 - using 67
- llstatus -l -b command sample output listing 679
- llstatus -l -x command sample output listing 681
- llstatus -l command sample output listing 678
- llstatus command 477
- llsubmit command 494
 - using for reservations 203
- llsubmit subroutine 616
- llsummary command 496
- lltg, configuration wizard 159
- load average 670
- LoadAvg variable
 - detailed description 297
 - use on MACHPRIO keyword 266
- loadl user ID 27, 39
- LoadL_admin file 301
- LOADL_ADMIN keyword 41
 - detailed description 264
- LOADL_CONFIG 65
- LoadL_config file 39
- LoadL_config.local file 39
- LOADL_INTERACTIVE_CLASS variable 311
- LOADL_JOB_CPU_LIMIT
 - environment variable 69
- LOADL_PROCESSOR_LIST
 - environment variable 197
- LOADL_STEP_CLASS
 - environment variable 69
- LOADL_STEP_COMMAND
 - environment variable 69
- LOADL_STEP_ID
 - environment variable 69
- LOADL_STEP_OWNER
 - environment variable 69
- LOADL_WALL_LIMIT
 - environment variable 69
- LoadLeveler
 - directory location after installation 30
 - job states 18
 - port usage information 685
 - starting 29
 - steps for integrating with WLM 127
- LoadLeveler APIs
 - 64-bit support 505
- LoadLeveler basics 3
- LoadLeveler cluster
 - initialize machines
 - using llinit command 429
- LoadLeveler commands
 - llacctmrg 387
 - sample output 673
- LoadLeveler daemon
 - overview 7
- LoadLeveler daemons
 - control using llctl command 409
- LoadLeveler for Linux quick installation and configuration 28
- LoadLeveler interfaces
 - application programming (API)
 - summary 503
 - command line
 - overview 385
- LoadLeveler multicluster support
 - local central manager 139
 - local cluster 139
 - local gateway Schedd 139
 - overview 139
 - remote central manager 139
 - remote cluster 139
 - remote gateway Schedd 139
- LoadLeveler support for checkpointing jobs 129
- LoadLeveler user ID 27
- LoadLeveler variables 294
 - Arch 294
 - ClassSysprio 294
 - Connectivity 294
 - ConsumableCpus 294
 - ConsumableMemory 294
 - ConsumableVirtualMemory 295
 - Cpus 295
 - CurrentTime 295
 - CustomMetric 295
 - Disk 295
 - domain 295
 - EnteredCurrentState 295
 - for setting dates
 - tm_mday 299
 - tm_mon 299
 - tm_wday 299
 - tm_yday 299
 - tm_year 299

LoadLeveler variables (*continued*)
 for setting dates (*continued*)
 tm4_year 299
 usage 299
 for setting time
 tm_isdst 300
 tm_min 300
 tm_sec 300
 tm4_year 299
 usage 299
 FreeRealMemory 295
 GroupQueuedJobs 295
 GroupRunningJobs 296
 GroupSysprio 296
 GroupTotalJobs 296
 GroupTotalShares 296
 GroupUsedBgShares 296
 GroupUsedShares 296
 host 296
 hostname 296
 in a job command file 372
 JobsBlueGene 296
 KeyboardIdle 297
 LoadAvg 297
 Machine 297
 MasterMachPriority 297
 Memory 297
 OpSys 297
 PagesFreed 297
 PagesScanned 297
 QDate 297
 Speed 298
 state 298
 tilde 298
 UserPrio 298
 UserQueuedJobs 298
 UserRunningJobs 298
 UserSysprio 298
 UserTotalJobs 298
 UserTotalShares 298
 UserUsedBgShares 299
 UserUsedShares 299
 VirtualMemory 299
 local central manager 139
 local cluster 139
 local configuration file
 configuring 39
 local gateway Schedd 139
 local keyword
 detailed description 319
 LOCAL_CONFIG 45
 LOCAL_CONFIG keyword
 detailed description 264
 LOG 45
 log files 45
 LOG keyword
 detailed description 265
 logical_id keyword
 detailed description 319
 LookAt message retrieval tool xv
 LowLoad keyword 294

M

machine
 administrative actions 158

machine (*continued*)
 collect history files
 using llactmrg command 387
 GUI 158
 initialize in LoadLeveler cluster
 using llnit command 429
 public scheduling 329
 query status
 using llstatus command 477
 scheduling 6
 Machine
 requirement in job command file 363
 Machine object
 understanding 527
 machine stanza keywords
 adapter_stanzas 306
 alias 307
 central_manager 308
 cpu_speed_scale 310
 machine_mode 320
 master_node_exclusive 320
 max_jobs_scheduled 320
 name_server 325
 pool_list 326
 port_number 326
 reservation_permitted 327
 resources 327
 schedd_fenced 328
 schedd_host 329
 spacct_exclude_enable 330
 speed 330
 submit_only 330
 type 331
 machine stanzas
 examples 80
 format 78
 Machine variable
 detailed description 297
 MACHINE_AUTHENTICATE keyword
 detailed description 265
 machine_mode keyword
 detailed description 320
 MACHINE_UPDATE_INTERVAL 666
 MACHINE_UPDATE_INTERVAL keyword
 detailed description 265
 MACHPRIO 43
 MACHPRIO keyword
 detailed description 265
 mail keyword 294
 MAIL keyword
 detailed description 267
 mail program 75
 MALLOC_TYPE 29, 411, 625
 master daemon 8
 MASTER keyword
 detailed description 267
 master node 99
 MASTER_COREDUMP_DIR keyword
 detailed description 267
 MASTER_DGRAM_PORT keyword
 detailed description 268
 master_node_exclusive keyword
 detailed description 320
 master_node_requirement keyword
 detailed description 320
 MASTER_STREAM_PORT keyword
 detailed description 268

- MasterMachPriority variable
 - detailed description 297
 - use on MACHPRIO keyword 266
- MAX_CKPT_INTERVAL 174
- MAX_CKPT_INTERVAL keyword
 - detailed description 268
- MAX_JOB_REJECT keyword
 - detailed description 268
- max_jobs_scheduled keyword
 - detailed description 320
- max_node keyword
 - detailed description 321
- max_processors keyword
 - detailed description 321, 355
- max_protocol_instances 187
- max_protocol_instances keyword
 - detailed description 321
- max_reservation_duration keyword
 - detailed description 321
- max_reservations keyword
 - detailed description 322
- MAX_RESERVATIONS keyword
 - detailed description 268
- MAX_STARTERS
 - limits set by 52
- MAX_STARTERS keyword
 - detailed description 269
- max_top_dogs keyword
 - detailed description 322
- MAX_TOP_DOGS keyword
 - detailed description 269
- max_total_tasks keyword
 - detailed description 322
- maxidle 668
- maxidle keyword
 - detailed description 323
- maxjobs 668
- maxjobs keyword
 - detailed description 323
- maxqueued 668
- maxqueued keyword
 - detailed description 323
- MCluster object
 - understanding 528
- mcm_affinity_options keyword
 - detailed description 355
- Memory
 - requirement in job command file 364
- Memory variable
 - detailed description 297
 - use on MACHPRIO keyword 266
- message retrieval tool, LookAt xv
- messages 238
- MIN_CKPT_INTERVAL 174
- MIN_CKPT_INTERVAL keyword
 - detailed description 269
- min_processors keyword
 - detailed description 356
- MINUTE keyword 294
- modifying consumable resources and
 - using the GUI 233
- monitor_program 618
- monitoring programs 618
- monitoring, file system 50
- move spool
 - GUI 161
- MPI 356

- MPICH
 - job command file 192
 - running jobs 188
- MPICH jobs, configuring 99
- MPICH-GM
 - job command file 194
 - running jobs 188
- MPICH-GM jobs, configuring 100
- multicluster
 - troubleshootingt 663
- multicluster support
 - overview 139
- multicluster_security keyword
 - detailed description 324
- multilink_address keyword
 - detailed description 324
- multilink_list keyword
 - detailed description 324
- multiple statements
 - in administration file 119
 - in configuration file 119
- MVAPICH
 - running jobs 188
- MVAPICH jobs, configuring 100

N

- name_server keyword
 - detailed description 325
- naming
 - checkpoint files 131
 - checkpointing files and directories 134
 - checkpointing files for interactive parallel jobs 135
 - checkpointing files serial and batch 135
- naming for checkpointing 134
- negotiator daemon 12
 - job states 18
- NEGOTIATOR keyword
 - detailed description 269
- NEGOTIATOR_COREDUMP_DIR keyword
 - detailed description 270
- NEGOTIATOR_CYCLE_DELAY keyword
 - detailed description 270
- NEGOTIATOR_CYCLE_TIME_LIMIT keyword
 - detailed description 270
- NEGOTIATOR_INTERVAL 666
- NEGOTIATOR_INTERVAL keyword
 - detailed description 270
 - using 67
- NEGOTIATOR_LOADAVG_INCREMENT keyword
 - detailed description 271
- NEGOTIATOR_PARALLEL_DEFER keyword
 - detailed description 271
- NEGOTIATOR_PARALLEL_HOLD keyword
 - detailed description 271
- NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL keyword
 - detailed description 271
- NEGOTIATOR_REJECT_DEFER keyword
 - detailed description 272
- NEGOTIATOR_REMOVE_COMPLETED keyword
 - detailed description 272
- NEGOTIATOR_RESCAN_QUEUE keyword
 - detailed description 272
- NEGOTIATOR_STREAM_PORT keyword
 - detailed description 272

- network keyword
 - detailed description 356
- network_id keyword
 - detailed description 325
- network_type keyword
 - detailed description 325
- nice keyword
 - detailed description 325
- node availability 79
- node keyword 180
 - detailed description 359
- node_usage keyword
 - detailed description 360
- Not Run job state
 - abbreviations 18
 - detailed description 18
- notification keyword
 - detailed description 360
- notify_user keyword
 - detailed description 361
- NotQueued job state
 - abbreviations 18
 - detailed description 18

O

- OBITUARY_LOG_LENGTH keyword
 - detailed description 273
- objects
 - Blue Gene
 - understanding 524
 - Class
 - understanding 524
 - Cluster
 - understanding 525
 - data access
 - understanding 523
 - Fairshare
 - understanding 525
 - Job
 - understanding 525
 - Machine
 - understanding 527
 - MCluster
 - understanding 528
 - Reservations
 - understanding 528
 - Wlmstat
 - understanding 529
- obtaining status, parallel jobs 196
- openssl
 - multicluster, securing 143
- OpenSSL
 - administration keyword
 - multicluster_security 324
 - ssl_cipher_list 330
- operators 244
- OpSys
 - requirement in job command file 364
- OpSys variable
 - detailed description 297
- outbound_hosts keyword
 - detailed description 326
- output 669
 - debugging 47
 - from commands 673

- output keyword
 - detailed description 361
- overview
 - fair share scheduling 25

P

- PagesFreed variable
 - detailed description 297
 - use on MACHPRIO keyword 266
- PagesScanned variable
 - detailed description 297
 - use on MACHPRIO keyword 266
- parallel job command files 334
- parallel jobs
 - administration 96
 - checklist 655
 - Class keyword 98
 - class stanza 98
 - interactive, naming files for checkpointing 135
 - job command file examples 191
 - master node 99
 - obtaining status 196
 - scheduling considerations 96
 - supported keywords 96
- parallel jobs, batch
 - naming files for checkpointing 135
- pending job state 657
- Pending job state
 - abbreviations 18
 - detailed description 18
- planning
 - checkpointing 130
 - POE 97
- POE
 - environment variables 188
 - job command file 191
 - planning considerations 97
 - software levels 179
- POLLING_FREQUENCY keyword
 - detailed description 273
- POLLS_PER_UPDATE keyword
 - detailed description 273
- Pool
 - requirement in job command file 364
- pool_list keyword
 - detailed description 326
- port number definition 685
- port numbers, configuration file keywords 685
- port usage information 685
- port_number keyword
 - detailed description 326
- post-installation considerations
 - LoadLeveler directory location 30
 - starting LoadLeveler 29
 - submit-only LoadLevelerr directory location 31
- preempt
 - job step
 - using llpreempt command 444
- Preempt Pending job state
 - abbreviations 18
 - detailed description 18
- PREEMPT_CLASS keyword
 - detailed description 274
- Preempted job state
 - abbreviations 18
 - detailed description 18

- preemption
 - avoiding 117
 - releasing job resources 119
 - selecting a method 118
 - two types 116
- preemption and coscheduled job steps 172
- preemption method
 - selecting 118
- PREEMPTION_SUPPORT keyword
 - detailed description 275
- preferences keyword
 - detailed description 361
- PRESTARTED_STARTERS keyword
 - detailed description 273
- priority
 - of jobs
 - user priority 447
- priority (of jobs)
 - setting or changing 212
 - system priority 212
 - setting or changing 43, 67
 - user priority 212
- priority keyword
 - detailed description 326
- procedure
 - job accounting setup 61
 - job spool recovery 154
- process
 - starter 12
- PROCESS_TRACKING 64
- PROCESS_TRACKING keyword
 - detailed description 276
- PROCESS_TRACKING_EXTENSION 64
- PROCESS_TRACKING_EXTENSION keyword
 - detailed description 276
- productivity aids 666
- prolog programs 70
- public scheduling machine 329
- public scheduling machines 6, 212
- PUBLISH_OBITUARIES keyword
 - detailed description 276
- pull-down menus
 - creating 381
- purge Schedd
 - GUI 160

Q

- QDate variable
 - detailed description 297
 - use on SYSPRIO keyword 287
- query a job
 - using llq command 449
 - using the GUI 231
- query API 595
- querying class information
 - using llclass command 403
- querying multiple clusters 65
- questions and answers 651
- queue keyword
 - detailed description 362
- queue, see job queue 7
- quick start procedure
 - before you begin 27
 - LoadLeveler for Linux 28
 - using the default configuration files 27

R

- RDMA
 - configuring 57
 - specifying for jobs 173
- reconfig
 - GUI 158
- reconfiguration
 - changing scheduler types 115
- reconfiguring
 - fair share scheduling 150
- recycle
 - GUI 158
- Reject Pending job state
 - abbreviations 18
 - detailed description 18
- REJECT_ON_RESTRICTED_LOGIN keyword
 - detailed description 276
- Rejected job state
 - abbreviations 18
 - detailed description 18
- release from hold 156
- RELEASEDIR 45
- RELEASEDIR keyword
 - detailed description 277
- remote central manager 139
- remote cluster 139
- remote direct-memory access (RDMA)
 - configuring 57
 - specifying for jobs 173
- remote gateway Schedd 139
- remove job from reservation 234
- Remove Pending job state
 - abbreviations 18
 - detailed description 18
- Removed job state
 - abbreviations 18
 - detailed description 18
- requirements keyword
 - detailed description 362
- RESERVATION_CAN_BE_EXCEEDED keyword
 - detailed description 277
- RESERVATION_HISTORY keyword
 - detailed description 277
- RESERVATION_MIN_ADVANCE_TIME keyword
 - detailed description 277
- reservation_permitted keyword
 - detailed description 327
- RESERVATION_PRIORITY keyword
 - detailed description 278
- RESERVATION_SETUP_TIME keyword
 - detailed description 278
- reservation, add job to 234
- reservation, remove job from 234
- reservations
 - canceling 205
 - modifying attributes 204
 - owner tasks 200, 204, 205
 - querying 204
 - removing bound jobs 203
 - steps for configuring 121
 - submitting jobs 202
 - troubleshooting 652
- Reservations object
 - understanding 528
- resetting historic data
 - fair share scheduling 153

- resources
 - held by preemptable jobs 119
- resources keyword
 - detailed description 327, 365
- RESOURCES keyword
 - detailed description 278
- resources, consumable
 - job scheduling 21
 - Workload Manager 22
- restart
 - restarting a checkpointed job 655
- restart keyword
 - detailed description 366
- restart_from_ckpt keyword
 - detailed description 367
- restart_on_same_nodes keyword
 - detailed description 367
- restoring historic data
 - fair share scheduling 153
- restrictions
 - checkpointing 130
- resume
 - GUI 159
- Resume Pending job state
 - abbreviations 18
 - detailed description 18
- RESUME_ON_SWITCH_TABLE_ERROR_CLEAR keyword
 - detailed description 279
- rlim_infinity 305, 335
- RSCT peer domain
 - extracting data from
 - using llextrRPD command 415
- rset keyword
 - detailed description 367
- RSET_SUPPORT keyword
 - detailed description 279
- rss_limit keyword
 - detailed description 328
- run-time environment variables
 - in a job command file 374
- Running job state
 - abbreviations 18
 - detailed description 18
- running jobs at a specific time of day 667
- running-like job states 438

S

- SAVELOGS keyword 49
 - detailed description 279
- saving historic data
 - fair share scheduling 153
- scaling considerations 666
- Schedd
 - local gateway 139
 - remote gateway 139
 - troubleshooting 670
- Schedd daemon 9, 657
 - recovery 660
- SCHEDD keyword
 - detailed description 279
- SCHEDD_COREDUMP_DIR keyword
 - detailed description 280
- schedd_fenced keyword
 - detailed description 328
- schedd_host 666
 - schedd_host keyword
 - detailed description 329
 - SCHEDD_INTERVAL keyword
 - detailed description 280
 - SCHEDD_RUNS_HERE keyword
 - detailed description 280
 - SCHEDD_STATUS_PORT keyword
 - detailed description 281
 - SCHEDD_STREAM_PORT keyword
 - detailed description 281
 - SCHEDD_SUBMIT_AFFINITY 666
 - SCHEDD_SUBMIT_AFFINITY keyword
 - detailed description 280
 - SCHEDULE_BY_RESOURCES keyword
 - detailed description 281
 - SCHEDULER_TYPE keyword
 - detailed description 281
- schedulers
 - API 42, 619
 - BACKFILL 42
 - choosing 42
 - Default 42
 - external 42, 619
 - supported keywords 178
- scheduling
 - avoiding circular preemption 117
 - BACKFILL
 - implied START_CLASS values 118
 - releasing resources of preemptable jobs 119
 - selecting a preemption method 118
 - parallel jobs 96
 - reconfiguration 115
- scheduling affinity
 - configuring LoadLeveler to use 136
 - submitting jobs 205
- scheduling cycle
 - controlling
 - example 67
- scheduling machine 6
 - public 329
- scheduling, job
 - consumable resources 21
- script not executing
 - troubleshooting 661
- SEC_ADMIN_GROUP keyword
 - detailed description 282
- SEC_ENABLEMENT keyword
 - detailed description 282
- SEC_IMPOSED_MECHS keyword
 - detailed description 283
- SEC_SERVICES_GROUP keyword
 - detailed description 283
- secure_schedd_port keyword
 - detailed description 329
- security
 - configuring cluster security services 53
- security credentials
 - DCE 68
- security service
 - configuring 52
- serial job command files 333
- serial jobs
 - naming files for checkpointing 135
- service_class 356
- shell 221
- shell keyword
 - detailed description 368

- shortcut keys
 - keyboard 687
- smt keyword
 - detailed description 329, 368
- spacct_exclude_enable keyword
 - detailed description 330
- speed keyword
 - detailed description 330
- Speed variable
 - detailed description 298
 - use on MACHPRIO keyword 266
- SPOOL
 - log 45
- SPOOL keyword
 - detailed description 283
- ssl_cipher_list keyword
 - detailed description 330
- stack_limit keyword
 - detailed description 330, 369
- stanzas
 - adapter 80
 - characteristics 303
 - class 92
 - default 303
 - label 303
 - machine 78
 - type 303
 - user 77
- start all
 - GUI 158
- start Drained
 - GUI 158
- START expression
 - detailed description 283
- start failure
 - MALLOCTYPE 29, 411, 625
- start LoadLeveler
 - GUI 158
- START_CLASS keyword
 - detailed description 284
 - implied values 118
- START_DAEMONS keyword
 - detailed description 284
- startd daemon 10, 657, 666
- STARTID keyword
 - detailed description 285
- STARTD_COREDUMP_DIR keyword
 - detailed description 285
- STARTD_DGRAM_PORT keyword
 - detailed description 285
- STARTD_RUNS_HERE keyword
 - detailed description 285
- STARTD_STREAM_PORT keyword
 - detailed description 285
- startdate keyword
 - detailed description 369
- STARTER keyword
 - detailed description 286
- starter process 12
- STARTER_COREDUMP_DIR keyword
 - detailed description 286
- Starting job state
 - abbreviations 18
 - detailed description 18
- starting LoadLeveler
 - post-installation considerations 29
- State variable
 - detailed description 298
- StateTimer keyword 294
- status 494
 - query for machines
 - using llstatus command 477
- status, obtaining
 - parallel jobs 196
- step_name keyword
 - detailed description 369
- stop all
 - GUI 158
- stop LoadLeveler
 - GUI 158
- striping
 - definition of 182
 - examples of requesting striping in network statement 187
 - over multiple networks 184
 - submitting jobs 182
 - understanding over a single network 186
- structure
 - administration file 301
- SUBMIT_FILTER 70
- SUBMIT_FILTER keyword
 - detailed description 286
- submit_only keyword
 - detailed description 330
- submit-only LoadLeveler
 - directory location after installation 31
- submit-only machine
 - canceling jobs 214
 - definition 3
 - keywords 330
 - master daemon interaction 8
 - querying jobs from 211
 - querying multiple clusters 65
 - Schedd daemon interaction 9
 - submitting jobs from 178
 - troubleshooting 656
 - types 6
- submitting coscheduled job steps 171
- submitting jobs
 - across multiple clusters 65
 - using a job command file 177
 - using an API 614
 - using llsubmit 215
 - using llsubmit command 494
 - using the GUI 230
- subroutines
 - ckpt 511
 - GetHistory 507
 - ll_bind 601
 - ll_change_reservation 204, 604
 - ll_ckpt 512
 - ll_cluster 620
 - ll_cluster_auth 622
 - ll_config_changed 520
 - ll_control 624
 - ll_deallocate 530
 - ll_error 591
 - ll_fair_share 593
 - ll_free_jobs 596
 - ll_free_nodes 597
 - ll_free_objs 531
 - ll_get_data 532
 - ll_get_jobs 598
 - ll_get_nodes 599

- subroutines (*continued*)
 - ll_get_objs 574
 - ll_init_ckpt 515
 - ll_init_reservation_param 608
 - ll_make_reservation 201, 609
 - ll_modify 628
 - ll_move_job 632
 - ll_move_spool 634
 - ll_next_obj 577
 - ll_preempt 637
 - ll_preempt_jobs 639
 - ll_query 578
 - ll_read_config 521
 - ll_remove_reservation 205, 612
 - ll_reset_request 579
 - ll_run_scheduler 642
 - ll_set_ckpt_callbacks 517
 - ll_set_request 580
 - ll_start_job 643
 - ll_start_job_ext 645
 - ll_terminate_job 649
 - ll_unset_ckpt_callbacks 518
 - llactval user exit 509
 - llfree_job_info 615
 - llsubmit 616
- substanzas
 - defining in class stanzas 88
- summary
 - of LoadLeveler commands 385
- support on TWS LoadLeveler for InfiniBand adapters 81
- support services 670
- support, 64-bit keywords 244, 305, 335
- SUSPEND expression
 - detailed description 287
- syntax
 - administration file 301
- sys/wait.h 73
- syshold 156
- SYSPRIO keyword 43, 212
 - detailed description 287
- SYSPRIO_THRESHOLD_TO_IGNORE_STEP keyword
 - detailed description 289
- System Hold job state
 - abbreviations 18
 - detailed description 18
- system priority
 - definition 212
 - setting or changing 43, 67, 212
- system queue
 - reorder by job
 - using llfavorjob command 419
 - reorder by user
 - using llfavoruser command 421
- system-initiated checkpointing 129, 341

T

- take checkpoint 234
- task assignment 180
- task_geometry 180
- task_geometry keyword
 - detailed description 370
- tasks
 - administration file, modifying
 - steps 77
 - Blue Gene
 - overview 143

- tasks (*continued*)
 - Blue Gene jobs, submitting
 - steps 208
 - Blue Gene support, configuring
 - roadmap 145
 - steps 145
 - configuration file, modifying
 - steps 40
 - Configuring and managing the LoadLeveler environment
 - roadmap 37
 - fair share scheduling
 - examples 151, 152
 - jobs, building
 - roadmap 165, 211
 - jobs, preempting
 - roadmap 116
 - jobs, submitting
 - roadmap 165, 211
 - jobs, submitting in multicluster
 - steps 206
 - LoadLeveler interfaces, using
 - roadmap 241
 - multicluster
 - overview 139
 - multicluster, configuring
 - roadmap 140, 205
 - steps 141
 - multicluster, securing
 - steps 143
 - parallel jobs, launching
 - steps for reducing overhead 96
 - Providing additional job-processing controls
 - roadmap 66
 - reservations
 - removing bound jobs 203
 - submitting jobs 202
 - reservations, configuring
 - roadmap 121
 - steps 121
 - reservations, creating
 - administrators only 200
 - owners only 200
 - reservations, managing
 - owners only 204, 205
 - querying 204
 - resources, reserving
 - roadmap 121
 - scheduler, configure for preemption
 - steps 120
- tasks_per_node keyword 180
 - detailed description 370
- Terminated job state
 - abbreviations 18
 - detailed description 18
- termination of coscheduled job steps 173
- tilde variable
 - detailed description 298
- tm_isdst variable 300
- tm_mday variable 299
- tm_min variable 300
- tm_mon variable 299
- tm_sec variable 300
- tm_wday variable 299
- tm_yday variable 299
- tm_year variable 299
- tm4_year variable 299
- total_tasks keyword 180

- total_tasks keyword (*continued*)
 - detailed description 331, 371
- TotalMemory
 - requirement in job command file 364
- trademarks 691
- troubleshooting 651
 - .login script not executing 661
 - .profile script not executing 661
 - central manager isn't operating 658
 - checkpointed job won't restart 655
 - configuration or administration file 661
 - core file on Linux 660
 - history file and Schedd 670
 - inconsistencies in llfs output 661
 - job stays in pending or starting state 657
 - job won't run 652
 - job won't run on cluster with both AIX and Linux machines 656
 - Linux CPU affinity support 656
 - llstatus does not agree with llq 658
 - mksysb created when running jobs 662
 - multicluster environment 663
 - parallel job won't run 654
 - recovering resources 660
 - reservations 652
 - reserved node is down 662
 - running jobs when a machine goes down 657
 - set up problems with parallel jobs 655
 - setuid = 0 661
 - starting LoadLeveler 651
 - submit-only job won't run 656
- TRUNC_GSMONITOR_LOG_ON_OPEN keyword
 - detailed description 289
 - usage 47
- TRUNC_KBDD_LOG_ON_OPEN keyword
 - detailed description 289
 - usage 47
- TRUNC_MASTER_LOG_ON_OPEN keyword
 - detailed description 290
 - usage 47
- TRUNC_NEGOTIATOR_LOG_ON_OPEN keyword
 - detailed description 290
 - usage 47
- TRUNC_SCHEDD_LOG_ON_OPEN keyword
 - detailed description 290
 - usage 47
- TRUNC_STARTD_LOG_ON_OPEN keyword
 - detailed description 290
 - usage 47
- TRUNC_STARTER_LOG_ON_OPEN keyword
 - detailed description 290
 - usage 47
- type keyword
 - detailed description 331

U

- understanding striping over a single network 186
- understanding striping over multiple networks 184
- unfavor jobs 156
- unfavor users 155
- UNIX group 311
- unlimited blocking 180, 340
- UPDATE_ON_POLL_INTERVAL_ONLY keyword
 - detailed description 291
- User and System Hold job state
 - abbreviations 18

- User and System Hold job state (*continued*)
 - detailed description 18
- user exits
 - llacctval 509
 - monitoring programs 618
- User Hold job state
 - abbreviations 18
 - detailed description 18
- user name 79
- user priority
 - definition 212
 - setting or changing 212
- user space jobs
 - configuring bulk data transfer 57
 - using bulk data transfer 173
- user stanza keywords
 - account 306
 - default_class 310
 - default_group 311
 - default_interactive_class 311
 - env_copy_name 313
 - fair_shares 316
 - max_node 321
 - max_processors 321
 - max_reservation_duration 321
 - max_reservations 322
 - max_total_tasks 322
 - maxidle 323
 - maxjobs 323
 - maxqueued 323
 - total_tasks 331
 - type 331
- user stanzas
 - examples 91
 - format 77
- user substanzas
 - examples 88, 89
- user substanzas in class stanzas
 - defining 88
- user_priority keyword
 - detailed description 372
- user-defined keywords 293
 - BackgroundLoad 293
 - CPU_Busy 293
 - CPU_Idle 293
 - HighLoad 293
 - HOUR 293
 - JobLoad 293
 - KeyboardBusy 293
 - LowLoad 294
 - mail 294
 - MINUTE 294
 - StateTimer 294
- user-initiated checkpointing 129, 341
- UserBgSharesExceeded user-defined variable
 - use on SYSPRIO keyword 149
- UserHasBgShares user-defined variable
 - use on SYSPRIO keyword 149
- UserHasShares user-defined variable
 - use on SYSPRIO keyword 149
- UserPrio variable
 - detailed description 298
 - use on SYSPRIO keyword 287
- UserQueuedJobs variable
 - detailed description 298
 - use on SYSPRIO keyword 287

- UserRemainingBgShares user-defined variable
 - use on SYSPRIO keyword 149
- UserRemainingShares user-defined variable
 - use on SYSPRIO keyword 149
- UserRunningJobs variable
 - detailed description 298
 - use on SYSPRIO keyword 287
- UserSharesExceeded user-defined variable
 - use on SYSPRIO keyword 149
- UserSysprio variable
 - detailed description 298
 - use on SYSPRIO keyword 287
- UserTotalJobs variable
 - detailed description 298
 - use on SYSPRIO keyword 287
- UserTotalShares variable
 - detailed description 298
 - use on SYSPRIO keyword 287
- UserUsedBgShares variable
 - detailed description 299
 - use on SYSPRIO keyword 287
- UserUsedShares variable
 - detailed description 299
 - use on SYSPRIO keyword 287
- using the default configuration files
 - quick start procedure 27

V

- VACATE expression
 - detailed description 291
- Vacate Pending job state
 - abbreviations 18
 - detailed description 18
- Vacated job state
 - abbreviations 18
 - detailed description 18
- var=value specification
 - on environment keyword 349
- variables
 - LoadLeveler 372
 - run-time environment 374
- version
 - GUI 161
- VirtualMemory variable
 - detailed description 299
 - use on MACHPRIO keyword 266
- VM_IMAGE_ALGORITHM keyword
 - detailed description 291

W

- wall_clock_limit keyword
 - detailed description 331, 372
- WALLCLOCK_ENFORCE keyword
 - detailed description 292
- WLM
 - consumable resources 22
 - steps for integrating with LoadLeveler 127
- Wlmstat object
 - understanding 529
- workload manager
 - consumable resources 22
- Workload Manager
 - steps for integrating with LoadLeveler 127

X

- X_RUNS_HERE keyword
 - detailed description 292
- xloadl 377
- Xloadl 380
- Xloadl_so 380

Readers' comments – We'd like to hear from you

Tivoli Workload Scheduler LoadLeveler
Using and Administering
Version 3 Release 4

Publication No. SA22-7881-06

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:

- Send your comments to the address on the reverse side of this form.

If you would like a response from IBM, please fill in the following information:

Name

Address

Company or Organization

Phone No.

E-mail address



Fold and Tape

Please do not staple

Fold and Tape



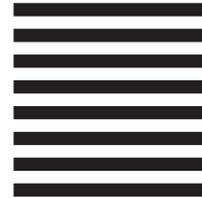
NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie NY 12601-5400



Fold and Tape

Please do not staple

Fold and Tape



Program Number: 5765-E69 and 5724-I23

Printed in USA

SA22-7881-06

