

IBM LoadLeveler for AIX 5L



# Using and Administering

*Version 3 Release 1*



IBM LoadLeveler for AIX 5L



# Using and Administering

*Version 3 Release 1*

Before using this information and the product it supports, read the information in "Notices" on page 455.

#### **First Edition, December 2001**

This edition applies to Version 3 Release 1.0 of IBM® LoadLeveler® for AIX®, program number 5765-E69, and to all subsequent releases and modifications until otherwise indicated in new editions or Technical Newsletters.

This edition replaces Version 2 Release 2.0 of IBM LoadLeveler for AIX, program number 5765-D61.

IBM welcomes your comments. If you wish to send comments to IBM, address your comments to:

IBM Corporation, Department 55JA, Mail Station P384  
2455 South Road  
Poughkeepsie, NY 12601-5400  
USA

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

FAX (Other Countries): Your International Access Code+1+845+432-9405

Internet email: [mhvrcfs@vnet.ibm.com](mailto:mhvrcfs@vnet.ibm.com)

World Wide Web: <http://www.ibm.com/eserver/pseries/>

Please 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 International Business Machines Corporation 1993, 2001. All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Who Should Use This Book.</b>	xi
<b>How this Book is Organized</b>	xiii
Typographic Conventions	xiii
<b>Related Information.</b>	xv
Information Formats	xv
Accessing This Book off the World Wide Web	xv
Accessing LoadLeveler Documentation Online	xv
LoadLeveler Man Pages	xv
<b>What's New in 3.1</b>	xvii
Book reorganization	xvii
Submitting jobs that use striping	xvii
Integration with AIX Workload Manager	xvii
Gang scheduling	xvii
Checkpoint/Restart	xvii
Support for 64-bit applications	xviii
File system monitoring	xviii
<b>Migration Considerations</b>	xix
Moving From 2.1 to 2.2	xix
Keyword Added to Administration File	xix
Changes in LoadLeveler Command Output	xix
Moving from 2.2 to 3.1	xix
Interaction with AIX Workload Manager	xix
Checkpoint considerations	xix
Gang scheduling considerations	xix
LoadLeveler 2.2 and 3.1 coexistence	xix
<b>Part 1. LoadLeveler overview</b>	1
<b>Chapter 1. What is LoadLeveler?</b>	3
LoadLeveler basics	4
How LoadLeveler works	5
Network job management and job scheduling systems	5
How LoadLeveler schedules jobs	7
LoadLeveler daemons	8
The LoadLeveler job cycle	9
Consumable resources	14
<b>Part 2. LoadLeveler interfaces</b>	17
<b>Chapter 2. LoadLeveler command line interface</b>	19
Summary of LoadLeveler commands	20
<b>Chapter 3. Using the Graphical User Interface</b>	21
Starting the Graphical User Interface	21
Specifying options	21
The LoadLeveler main window	21
Getting help using the Graphical User Interface	23

Differences between LoadLeveler's Graphical User Interface and other Graphical User Interfaces . . . . .	23
Graphical User Interface typographic conventions . . . . .	23
Customizing the Graphical User Interface . . . . .	24
Syntax of an Xloadl file . . . . .	24
Modifying windows and buttons . . . . .	24
Creating your own pull-down menus . . . . .	25
Customizing fields on the Jobs window and the Machines window . . . . .	26
Modifying help panels . . . . .	26
Administrative uses for the Graphical User Interface . . . . .	27
Job related administrative actions . . . . .	27
Machine related administrative actions . . . . .	29
<b>Chapter 4. LoadLeveler API interface . . . . .</b>	<b>33</b>
Summary of LoadLeveler APIs . . . . .	33

---

## Part 3. User tasks . . . . . 37

<b>Chapter 5. Submitting and managing jobs . . . . .</b>	<b>39</b>
Building a job command file . . . . .	39
Job command file syntax . . . . .	39
Submitting a job command file . . . . .	42
Managing jobs . . . . .	42
Editing job command files . . . . .	43
Querying the status of a job . . . . .	43
Placing and releasing a hold on a job . . . . .	44
Cancelling a job . . . . .	44
Checkpointing a job . . . . .	44
Setting and changing the priority of a job . . . . .	44
Working with machines . . . . .	45
Run-time environment variables. . . . .	46
Managing jobs that consume resources. . . . .	47
Specifying the consumption of resources by a job step . . . . .	47
Displaying currently available resources. . . . .	47
<b>Chapter 6. Special considerations for parallel jobs . . . . .</b>	<b>49</b>
Supported parallel environments . . . . .	49
Keyword considerations for parallel jobs . . . . .	49
Scheduler considerations . . . . .	49
Task assignment considerations . . . . .	50
Submitting jobs that use striping . . . . .	52
Understanding striping . . . . .	53
Using striping . . . . .	54
Running interactive POE jobs . . . . .	54
Job command file examples . . . . .	54
Obtaining status of parallel jobs. . . . .	54
Obtaining allocated host names. . . . .	55

---

## Part 4. Administrator tasks . . . . . 57

<b>Chapter 7. Administering and configuring LoadLeveler . . . . .</b>	<b>59</b>
Overview . . . . .	59
Planning considerations . . . . .	59
Where to begin? . . . . .	60
Quick set up. . . . .	61

Administering LoadLeveler . . . . .	62
Administration file structure and syntax . . . . .	62
Configuring LoadLeveler . . . . .	63
The configuration files . . . . .	64
Configuration file structure and syntax . . . . .	64
Considerations for integrating LoadLeveler with AIX Workload Manager . . . . .	68
Keyword summary . . . . .	70
<b>Chapter 8. Administration tasks for parallel jobs . . . . .</b>	<b>71</b>
Scheduling considerations for parallel jobs . . . . .	71
Allowing users to submit interactive POE jobs . . . . .	71
Allowing users to submit PVM jobs . . . . .	72
Restrictions and limitations for PVM jobs . . . . .	73
Setting up a class for parallel jobs . . . . .	73
Setting up a parallel master node . . . . .	74
<b>Chapter 9. Gathering job accounting data . . . . .</b>	<b>75</b>
Collecting job resource data on serial and parallel jobs . . . . .	75
Collecting job resource data based on machines . . . . .	75
Collecting job resource data based on events . . . . .	76
Collecting job resource information based on user accounts . . . . .	76
Collecting the accounting information and storing it into files . . . . .	77
Accounting reports . . . . .	77
Job accounting setup procedure . . . . .	78
<b>Chapter 10. Routing jobs to NQS machines . . . . .</b>	<b>79</b>
Setting up the NQS environment . . . . .	79
Designating machines to which jobs will be routed . . . . .	80
NQS scripts . . . . .	80
NQS machine job routing procedure . . . . .	81

---

## Part 5. Detailed descriptions . . . . . 83

<b>Chapter 11. Job command file keywords . . . . .</b>	<b>85</b>
account_no . . . . .	85
arguments . . . . .	85
blocking . . . . .	85
checkpoint . . . . .	86
ckpt_dir . . . . .	87
ckpt_file . . . . .	87
ckpt_time_limit . . . . .	88
class . . . . .	88
comment . . . . .	88
core_limit . . . . .	88
cpu_limit . . . . .	89
data_limit . . . . .	89
dependency . . . . .	89
environment . . . . .	91
error . . . . .	91
executable . . . . .	92
file_limit . . . . .	92
group . . . . .	92
hold . . . . .	92
image_size . . . . .	93
initialdir . . . . .	93
input . . . . .	94

job_cpu_limit . . . . .	94
job_name . . . . .	94
job_type . . . . .	95
max_processors . . . . .	95
min_processors . . . . .	95
network . . . . .	96
node . . . . .	97
node_usage . . . . .	98
notification . . . . .	99
notify_user . . . . .	99
output . . . . .	99
parallel_path . . . . .	100
preferences . . . . .	100
queue . . . . .	100
requirements . . . . .	101
resources . . . . .	103
restart . . . . .	104
restart_from_ckpt . . . . .	104
restart_on_same_nodes . . . . .	105
rss_limit . . . . .	105
shell . . . . .	105
stack_limit . . . . .	105
startdate . . . . .	106
step_name . . . . .	106
task_geometry . . . . .	106
tasks_per_node . . . . .	107
total_tasks . . . . .	107
user_priority . . . . .	108
wall_clock_limit . . . . .	108
Job command file variables . . . . .	109
Example 1 . . . . .	110
Example 2 . . . . .	110
<b>Chapter 12. Administration and Configuration file keywords . . . . .</b>	<b>111</b>
Administration file keywords . . . . .	111
Configuration file keywords and LoadLeveler variables . . . . .	115
Keywords . . . . .	116
User-defined keywords . . . . .	125
LoadLeveler variables . . . . .	126
<b>Chapter 13. LoadLeveler daemons and job states . . . . .</b>	<b>129</b>
Daemons . . . . .	129
The master daemon . . . . .	129
The schedd daemon . . . . .	129
The startd daemon . . . . .	130
The negotiator daemon . . . . .	133
The kbdd daemon . . . . .	133
The gsmonitor daemon . . . . .	133
Job states . . . . .	134
<b>Chapter 14. Commands . . . . .</b>	<b>137</b>
llactmrg - Collect machine history files . . . . .	138
llcancel - Cancel a submitted job . . . . .	140
llckpt - Checkpoint a running job step . . . . .	142
llclass - Query class information . . . . .	144
llctl - Control LoadLeveler daemons . . . . .	148



lldcegrpmain - LoadLeveler DCE group maintenance utility . . . . .	153
llextrSDR - Extract adapter information from the SDR . . . . .	155
llfavorjob - Reorder system queue by job . . . . .	159
llfavoruser - Reorder system queue by user . . . . .	160
llhold - Hold or release a submitted job . . . . .	161
llinit - Initialize machines in the LoadLeveler cluster . . . . .	163
llmatrix - Query Gang matrix . . . . .	165
llmodify - Change attributes of a submitted job step . . . . .	168
llpreempt - Preempt a submitted job step . . . . .	170
llprio - Change the user priority of submitted job steps . . . . .	171
llq - Query job status . . . . .	173
llstatus - Query machine status . . . . .	191
llsubmit - Submit a job. . . . .	200
llsummary - Return job resource information for accounting . . . . .	202
 <b>Chapter 15. Application Programming Interfaces (APIs)</b> . . . . .	 215
Accounting API . . . . .	215
Account validation user exit. . . . .	215
Report generation subroutine . . . . .	216
Checkpointing API . . . . .	218
ckpt subroutine . . . . .	218
ll_init_ckpt . . . . .	218
ll_ckpt. . . . .	219
ll_set_ckpt_callbacks . . . . .	221
ll_unset_ckpt_callbacks . . . . .	222
Data Access API. . . . .	223
Using the data access API . . . . .	223
ll_query subroutine . . . . .	224
ll_set_request subroutine. . . . .	224
ll_reset_request subroutine . . . . .	227
ll_get_objs subroutine . . . . .	228
Understanding the LoadLeveler job object model . . . . .	230
ll_get_data subroutine . . . . .	233
ll_next_obj subroutine . . . . .	251
ll_free_objs subroutine . . . . .	251
ll_deallocate subroutine . . . . .	252
Examples of using the Data Access API . . . . .	252
Error Handling API . . . . .	259
ll_error subroutine . . . . .	259
Parallel Job API . . . . .	260
Interaction between LoadLeveler and the parallel API . . . . .	260
ll_get_hostlist subroutine . . . . .	261
ll_start_host subroutine . . . . .	263
Examples . . . . .	264
Query API . . . . .	265
ll_get_jobs subroutine . . . . .	265
ll_free_jobs subroutine . . . . .	266
ll_get_nodes subroutine . . . . .	267
ll_free_nodes subroutine . . . . .	268
Submit API . . . . .	268
llsubmit subroutine . . . . .	268
llfree_job_info subroutine. . . . .	269
Monitoring programs . . . . .	270
Workload Management API. . . . .	270
ll_control subroutine . . . . .	271
ll_modify subroutine . . . . .	275

ll_preempt subroutine . . . . .	277
ll_start_job subroutine . . . . .	278
ll_terminate_job subroutine . . . . .	280
Usage notes . . . . .	281
User exits . . . . .	282
Handling DCE security credentials . . . . .	283
Handling an AFS token . . . . .	284
Filtering a job script. . . . .	285
Using your own mail program . . . . .	286
Writing prolog and epilog programs . . . . .	286
<b>Chapter 16. Procedures.</b> . . . .	293
Using the Graphical User Interface . . . . .	293
Step 1: Building jobs . . . . .	293
Step 2: Edit the job command file . . . . .	301
Step 3: Submit a job command file . . . . .	303
Step 4: Display, refresh, and obtain job status . . . . .	303
Step 5: Sort the Jobs window . . . . .	304
Step 6: Change priorities of jobs in a queue. . . . .	305
Step 7: Hold a job . . . . .	305
Step 8: Release a hold on a job . . . . .	305
Step 9: Cancel a job . . . . .	305
Step 10: Modify consumable CPUs and consumable memory . . . . .	306
Step 11: Take checkpoint. . . . .	306
Step 12: Display and refresh machine status . . . . .	306
Step 13: Sort the Machines window. . . . .	307
Step 14: Find the location of the central manager. . . . .	308
Step 15: Find the location of the public scheduling machines . . . . .	308
Step 16: Find the type of scheduler in use . . . . .	308
Step 17: Specify which jobs appear in the Jobs window . . . . .	308
Step 18: Specify which machines appear in Machines window . . . . .	309
Step 19: Save LoadLeveler messages in a file. . . . .	310
Customizing the administration file . . . . .	310
Step 1: Specify machine stanzas . . . . .	310
Step 2: Specify user stanzas . . . . .	316
Step 3: Specify class stanzas . . . . .	319
Step 4: Specify group stanzas . . . . .	329
Step 5: Specify adapter stanzas . . . . .	332
Customizing the global and local configuration file . . . . .	333
Step 1: Define LoadLeveler administrators . . . . .	333
Step 2: Define LoadLeveler cluster characteristics . . . . .	334
Step 3: Define LoadLeveler machine characteristics . . . . .	339
Step 4: Define consumable resources . . . . .	341
Step 5: Specify how many jobs a machine can run . . . . .	342
Step 6: Prioritize the queue maintained by the negotiator . . . . .	343
Step 7: Prioritize the order of executing machines maintained by the negotiator . . . . .	345
Step 8: Manage a job's status using control expressions . . . . .	347
Step 9: Define job accounting . . . . .	349
Step 10: Specify alternate central managers . . . . .	350
Step 11: Specify where files and directories are located . . . . .	351
Step 12: Record and control log files . . . . .	352
Step 13: Define network characteristics . . . . .	355
Step 14: Enable checkpointing. . . . .	356
Planning considerations for checkpointing jobs. . . . .	359
How to checkpoint a job . . . . .	362

Remove old checkpoint files . . . . .	364
Step 15: Specify process tracking . . . . .	364
Step 16: Configuring LoadLeveler to use DCE security services . . . . .	365
Step 17: Specify additional configuration file keywords . . . . .	370
Setting up job accounting files . . . . .	374
Task 1: Update the configuration file . . . . .	374
Task 2: Merge multiple files collected from each machine into one file . . . . .	374
Task 3: Report job information on all the jobs in the history file . . . . .	374
Task 4: Using account numbers and setting up account validation. . . . .	375
Task 5: Specifying machines and their weights. . . . .	375
Routing jobs to NQS machines . . . . .	375
Task 1: Modify the administration file . . . . .	376
Task 2: Modify the configuration file . . . . .	376
Task 3: Submit the jobs . . . . .	376
Task 4: Obtain status of NQS jobs . . . . .	379
Task 5: Cancel NQS jobs . . . . .	379
<b>Chapter 17. Using Gang scheduling . . . . .</b>	<b>381</b>
Overview . . . . .	381
Gang scheduling concepts . . . . .	381
Hierarchical communication . . . . .	383
Task switching. . . . .	384
Supported hardware . . . . .	384
Application support . . . . .	384
Preemption. . . . .	384
Keywords specific to Gang scheduling . . . . .	385
Configuration file keywords for Gang scheduling . . . . .	385
Sample configuration file . . . . .	388
Administration file keywords for Gang . . . . .	389
Sample administration file . . . . .	390
Gang scheduling interactions and restrictions . . . . .	392
Network Time Protocol (NTP) . . . . .	392
Consumable resource enforcement . . . . .	392
Reconfiguration . . . . .	392
Circular preemption. . . . .	392
Restrictions for Gang scheduling and preemption. . . . .	392
Implied START_CLASS values . . . . .	393
Last one wins rule . . . . .	393
Job command file and Gang scheduling . . . . .	394
LoadLeveler commands for Gang . . . . .	394
APIs used with Gang scheduling . . . . .	394
<b>Chapter 18. Support for 64-bit applications . . . . .</b>	<b>395</b>
64-bit support for Job Command, Configuration, and Administration keywords . . . . .	395
64-bit support for Job Command file keywords. . . . .	395
64-bit support for Administration keywords . . . . .	396
64-bit support for Configuration keywords and expressions . . . . .	396
64-bit support for Command line interfaces and the GUI . . . . .	397
64-bit support for Command line interfaces . . . . .	397
64-bit support for the GUI . . . . .	398
64-bit support for the LoadLeveler APIs . . . . .	399
64-bit support for Accounting functions. . . . .	399

---

<b>Part 6. Appendixes . . . . .</b>	<b>401</b>
-------------------------------------	------------

<b>Appendix contents . . . . .</b>	<b>403</b>
------------------------------------	------------

<b>Appendix A. Examples</b>	405
User tasks: building job command files	405
Using commands	405
Additional examples of building job command files	407
User tasks: building parallel job command files.	411
POE	411
PVM 3.3 (non-SP)	413
PVM 3.3.11+ (SP2MPI architecture).	413
<b>Appendix B. Customer case studies.</b>	417
Customer 1: technical computing at the Cornell Theory Center.	417
System configuration	417
LoadLeveler configuration	417
Customer 2: circuit simulation	425
System configuration	425
LoadLeveler configuration	425
Customer 3: high-energy physics.	427
System configuration	427
LoadLeveler batch configuration	427
LoadLeveler interactive configuration	428
Processor configuration	428
Customer 4: computer chip design	428
System configuration	429
Interactive configuration	429
Batch configuration	432
Configuration for a machine that schedules (but doesn't run) jobs.	433
<b>Appendix C. Troubleshooting</b>	435
Troubleshooting LoadLeveler	435
Frequently Asked Questions	435
Helpful hints	443
Getting help from IBM.	447
<b>Bibliography</b>	449
Information formats	449
Finding documentation on the World Wide Web	449
Accessing PSSP documentation online	449
Manual pages for public code	450
RS/6000 SP planning publications	450
RS/6000 SP hardware publications	450
RS/6000 SP Switch Router publications	451
Related hardware publications.	451
RS/6000 SP software publications	451
AIX publications	453
DCE publications	453
Redbooks	453
Non-IBM publications	453
<b>Notices</b>	455
Trademarks and service marks	457
<b>Glossary</b>	459
<b>Index</b>	461

---

## Who Should Use This Book

This manual is intended for users and those who are responsible for administering LoadLeveler.

LoadLeveler user tasks include:

- Submitting parallel, serial, and interactive jobs
- Managing parallel, serial, and interactive jobs

LoadLeveler administrative tasks include:

- Setting up configuration and administration files
- Maintaining LoadLeveler
- Setting up the distributed environment for allocating batch jobs.

Users and Administrators should be experienced with the UNIX<sup>®</sup> commands. Administrators should be familiar with system management techniques such as SMIT, as it is used in the AIX<sup>\*</sup> environment. Knowledge of networking and NFS<sup>\*\*</sup> or AFS<sup>\*\*</sup> protocols is helpful, as well as knowledge of DCE.



---

## How this Book is Organized

This book contains the following sections:

- “Part 1. LoadLeveler overview” on page 1 gives an overview of LoadLeveler functions and describes how LoadLeveler works. Part 1 also gives a brief description and references for LoadLeveler daemons and job states.
- “Part 2. LoadLeveler interfaces” on page 17 introduces the three LoadLeveler interfaces and gives an overview of the Graphical User Interface. Part 2 also summarizes the commands and APIs used in those two interfaces.
- “Part 3. User tasks” on page 37 describes how to create and submit serial and parallel job command files.
- “Part 4. Administrator tasks” on page 57 describes how to perform administration tasks, such as configuring LoadLeveler, gathering accounting data, and routing jobs to NQS.
- “Part 5. Detailed descriptions” on page 83 provides reference information for keywords, daemons, commands, APIs, and procedures.
- The appendices contains sections with examples, case studies, and information on troubleshooting LoadLeveler.

A glossary and index are also included.

Users of LoadLeveler should, at a minimum, become familiar with “Part 1. LoadLeveler overview” on page 1, “Part 3. User tasks” on page 37, and “Part 2. LoadLeveler interfaces” on page 17. Administrators should, at a minimum, become familiar with “Part 1. LoadLeveler overview” on page 1 and “Part 4. Administrator tasks” on page 57, and may find it helpful to read “Troubleshooting LoadLeveler” on page 435.

---

## Typographic Conventions

This book uses the following typographic conventions:

Typographic	Usage
<b>Bold</b>	<ul style="list-style-type: none"><li>• <b>Bold</b> words or characters represent system elements that you must use literally, such as commands, flags, and path names.</li><li>• <b>Bold</b> words also indicate the first use of a term included in the glossary.</li></ul>
<i>Italic</i>	<ul style="list-style-type: none"><li>• <i>Italic</i> words or characters represent variable values that you must supply.</li><li>• <i>Italics</i> are also used for book titles and for general emphasis in text.</li></ul>
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>.





---

## Related Information

In addition to this publication, the following books are also part of the LoadLeveler library:

- *Diagnosis and Messages Guide* , GA22-7277
- *Installation Memo* , GI11-2819

---

## Information Formats

Documentation supporting RS/6000® SP™ software licensed programs is no longer available from IBM in hardcopy format. However, you can view, search, and print documentation in the following ways:

- On the World Wide Web
- Online (from the product media or the SP Resource Center)

---

## Accessing This Book off the World Wide Web

You can view or download this book (in PDF format) from the World Wide Web using the following URL:

[http://www.rs6000.ibm.com/resource/aix\\_resource/sp\\_books/loadleveler](http://www.rs6000.ibm.com/resource/aix_resource/sp_books/loadleveler)

---

## Accessing LoadLeveler Documentation Online

IBM ships on the product media manual pages, HTML files, and PDF files. In order to use these files you must install the appropriate file sets. For more information, see *LoadLeveler Installation Memo* , which is shipped on the product media.

To view the LoadLeveler books in HTML format, you need access to an HTML document browser such as Netscape. Once you install the HTML files, an index to the LoadLeveler books is found in **/usr/lpp/LoadL/html/index.html**.

You can also view the LoadLeveler books from the SP Resource Center, which is available under the Parallel Systems Support Programs (PSSP) or as a separately installed program. You invoke the Resource Center from PSSP by entering **resource\_center**. To invoke the Resource Center from the product CD, see the **readme.txt** file.

To view the LoadLeveler books in PDF format, you will need Adobe Acrobat Reader 3.0.1 or higher. The Acrobat Reader is freely available for downloading from the Adobe web site at <http://www.adobe.com>.

## LoadLeveler Man Pages

Manual (man) pages are available for all LoadLeveler commands. You can view the man page for a command by entering **man** and the command name. For example: **man llq**.

The following man pages associated with LoadLeveler APIs (Application Programming Interfaces) are also available to you. You can view these man pages by entering **man** and the name of the man page. For example: **man LoadL\_submitapi**.

Man Page Name	What it Describes
LoadL_acctapi	Accounting API
LoadL_ckptapi	Checkpointing API
LoadL_dataapi	Data Access API
LoadL_jobctlapi	Workload Management API
LoadL_parallelapi	Parallel job API
LoadL_queryapi	Query API
LoadL_submitapi	Submit API
LoadL_errorapi	Error handling API

---

## What's New in 3.1

The following is a list of new features and functions added for this release.

---

### Book reorganization

This edition features a reorganized version of the LoadLeveler Using and Administering Guide. Previous versions had detailed information mixed in with basic descriptions. With the reorganization, the first four parts of the book describe basic information and provide links to detailed descriptions. Locating detailed descriptions in a separate section simplifies finding reference information when needed and removes the burden of too much information when you are trying to understand the basics. For more information on the information contained in each sections, see “How this Book is Organized” on page xiii.

---

### Submitting jobs that use striping

Parallel jobs can request to use striped communication between nodes. Striping allows the job to use all the communication paths to the node. Two methods of striping can be used—IP striping and user space striping. For more information on submitting jobs that use striping, see “Submitting jobs that use striping” on page 52.

---

### Integration with AIX Workload Manager

LoadLeveler 3.1 enhances consumable resources support by using AIX Workload Manager (WLM) to enforce the use of consumable CPUs and real memory resources. In addition, users can obtain WLM CPU and real memory statistics for running jobs and dynamically update resources for an idle job. For more information see:

- “Consumable resources and AIX Workload Manager” on page 15
- “Considerations for integrating LoadLeveler with AIX Workload Manager” on page 68
- “Step 4: Define consumable resources” on page 341

---

### Gang scheduling

Gang scheduling combines coordinated context switching with both space-sharing and time-sharing strategies to support execution of parallel applications. Gang scheduling provides good overall system utilization and responsiveness to interactive workloads. This scheduling option is different from backfill scheduling which supports time-sharing and space-sharing scheduling but does not provide coordinated context switching. It is also different from EASY scheduling (a common instance of an external scheduler) which only supports space-sharing. For more information see “Chapter 17. Using Gang scheduling” on page 381.

---

### Checkpoint/Restart

**Note:** Before you consider using the Checkpoint/Restart function refer to the LoadL.README file in /usr/lpp/LoadL/READMEs for information on availability and support of this function.

Checkpointing is a method of saving the state of a 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. Both serial and parallel jobs can be checkpointed. LoadLeveler

provides mechanisms for a program to checkpoint itself as well as providing means for checkpoints to take place outside of the programs control.

For more information see “Step 14: Enable checkpointing” on page 356.

---

## Support for 64-bit applications

LoadLeveler version 3.1 has been enhanced to provide 64-bit support for interactive and batch jobs. With these enhancements:

- Users and Administrators can assign 64-bit integer values to selected keywords in the Job Command, Configuration, and Administration files. System resource limits, with the exception of CPU limits, are treated by LoadLeveler daemons and commands as 64-bit limits.
- The LoadLeveler commands and the GUI (xloadl) are modified to accept and display 64-bit information where appropriate.
- Both sets of 32-bit and 64-bit LoadLeveler APIs and libraries are available for application development. LoadLeveler and MPI checkpointing libraries support both 64-bit and 32-bit applications.
- Accounting statistics of completed jobs that have 64-bit integer values are preserved in the LoadLeveler history files as 64-bit data. The LoadLeveler interfaces that access the history files are modified to correctly process these 64-bit statistics.

For more information see:

- “Chapter 18. Support for 64-bit applications” on page 395
- “LoadLeveler 2.2 and 3.1 coexistence” on page xix

---

## File system monitoring

File system monitoring uses keywords to check the file system and increase system fault tolerance. This is done by setting the frequency that LoadLeveler checks the file system free space and by setting the upper and lower thresholds that initialize system responses to the free space available.

---

## Migration Considerations

This section describes some differences between LoadLeveler 2.1, 2.2, and 3.1. The *LoadLeveler Installation Memo* has more specific information about and procedures for migration.

---

### Moving From 2.1 to 2.2

#### Keyword Added to Administration File

The **css\_type** keyword has been added to adapter stanzas in the administration file. This keyword designates the type of switch adapter to be used; for more information, see “Step 5: Specify adapter stanzas” on page 332.

#### Changes in LoadLeveler Command Output

The **llstatus -l** output now lists: windows, memory, and connectivity for adapters; the switch fabric connectivity vector; information about free memory and paging, and consumable resource availability and use. For more information on the command, see “llstatus - Query machine status” on page 191.

Device memory for parallel jobs has been added to the Allocated Hosts and Task Instances lists in the **llq -l** output. For more information, see “llq - Query job status” on page 173.

---

### Moving from 2.2 to 3.1

#### Interaction with AIX Workload Manager

If LoadLeveler is enabled to interact with AIX Workload Manager (WLM), any predefined WLM configurations and WLM classes will not be honored.

#### Checkpoint considerations

Different versions of the checkpoint/restart function will not coexist on the same cluster of nodes. An application built with earlier checkpoint libraries (such as LoadLeveler 2.2) can only run on nodes running that version of LoadLeveler. For example, an applications built to use LoadLeveler 3.1 checkpointing must be run on nodes running LoadLeveler 3.1. A program which was checkpointed on an earlier version of LoadLeveler cannot be restarted on 3.1, and vice versa.

#### Gang scheduling considerations

Although Gang scheduling does not place constraints on migrating between LoadLeveler versions, the Gang scheduling option will not function until you have upgraded all nodes in the system to the latest version of LoadLeveler. Also, configuration and administration files that specify new Gang scheduling keywords will not be compatible with previous versions of LoadLeveler.

#### LoadLeveler 2.2 and 3.1 coexistence

LoadLeveler 3.1 can coexist with LoadLeveler 2.2. It is possible to operate a “mixed” LoadLeveler cluster consisting of some machines running the new 3.1 software while the remaining machines continue to run LoadLeveler 2.2. The operation of this mixed cluster is subject to a number of restrictions. Most of these restrictions exist because LoadLeveler 3.1 has new features that are not supported

by LoadLeveler 2.2. Other restrictions are due to the incompatibilities of the Parallel Environment software levels that support LoadLeveler 3.1 and LoadLeveler 2.2.

The requirements, restrictions, and operating characteristics of a mixed LoadLeveler cluster include:

- The Central Manager must run on a LoadLeveler 3.1 machine.
- Machines running version 2.2 software must have LoadLeveler service PTF 2.2.0.10 (U478841) or later installed.
- The job command, configuration, and administration files on all machines in the cluster should not use any keywords that are new and unique to version 3.1.
- Starting with LoadLeveler 3.1, 64-bit integer values can be assigned to a number of LoadLeveler variables and keywords that previously accepted only 32-bit integers. To avoid compatibility problems in a mixed cluster, values outside the range of a 32-bit integer should not be associated with these variables and keywords.
- Checkpointing can not be enabled in a mixed cluster. LoadLeveler 3.1 and 2.2 use different methodologies to provide support for checkpointing applications and they are not compatible.
- The LoadLeveler interface to AIX Workload Manager (WLM) should not be enabled in a mixed cluster.
- In a mixed cluster, the configuration keyword `SCHEDULER_TYPE` should not be set to `GANG`. `GANG` scheduling is supported only when all machines in a cluster run LoadLeveler 3.1 software.
- In a mixed cluster, the **llacctmrg** and **llsummary** commands should be run only on a 3.1 machine. The **llacctmrg** command is normally used by a LoadLeveler administrator to merge the job accounting information in the history files of all the schedd machines into a single global history file. This file is then used by the **llsummary** command to generate a cluster-wide accounting report. History files of 3.1 machines contain 64-bit integers. Since the **llsummary** command of LoadLeveler 2.2 does not support 64-bit integer data type, the **llacctmrg** and **llsummary** commands should not be executed on machines running LoadLeveler 2.2.
- No special action is required by LoadLeveler administrators to provide support for serial jobs. Jobs submitted from a 2.2 machine can be executed on a 3.1 machine and jobs submitted from a 3.1 machine can run on a 2.2 machine.
- If support for parallel POE jobs is required, administrators must be aware that LoadLeveler uses Parallel Environment for parallel job submission, and that the PE software requires the same level of PE to be used throughout the parallel job. Different levels of PE cannot be mixed. PE 3.2 supports only LoadLeveler 3.1, and PE 3.1 supports only LoadLeveler 2.2. Therefore, a POE parallel job can not run some of its tasks on LoadLeveler 2.2 machines and the remaining tasks on LoadLeveler 3.1 machines.

The **requirements** keyword of the job command file can be used to ensure that all the tasks of a POE job run the same levels of PE and LoadLeveler software in a mixed cluster. Here are three examples showing different ways this can be done:

1. If the statement `"# @ requirements = (LL_Version >= "3.1") && (OpSys == "AIX51")"` is included in the job command file, LoadLeveler's Central Manager will select only 3.1.0.0 or higher machines with the appropriate OpSys level for this job step. 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.

2. The tasks of a POE job will see a consistent environment if a statement such as "# @ requirements = (Machine == { "hostname1" "hostname2" }) && (OpSys == "AIX51")" is specified and "hostname1" and "hostname2" run the same levels of PE and LoadLeveler software.
3. If the mixed cluster has been partitioned into 2.2 and 3.1 LoadLeveler pools then requirement specifications such as "# @ requirements = (Pool == 31) && (OpSys == "AIX51")" or "# @ requirements = (Pool == 22) && (OpSys == "AIX43")" can be used to select machines running the same levels of software. Here, it is assumed that all the 2.2 machines in this mixed cluster are assigned to pool 22 and all 3.1 machines are assigned to pool 31. 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 3.1 machines, then it is important that the job be submitted from a 3.1 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.





---

## Part 1. LoadLeveler overview

### Overview summary

This section provides background material to help you understand LoadLeveler operation.

<b>Chapter 1. What is LoadLeveler?</b> . . . . .	3
LoadLeveler basics . . . . .	4
How LoadLeveler works . . . . .	5
Network job management and job scheduling systems . . . . .	5
Job definition . . . . .	5
Machine definition . . . . .	6
How LoadLeveler schedules jobs . . . . .	7
LoadLeveler daemons . . . . .	8
The LoadLeveler job cycle . . . . .	9
LoadLeveler job states . . . . .	13
Consumable resources . . . . .	14
Consumable resources and AIX Workload Manager . . . . .	15



## 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*. An environment can include heterogeneous clusters, dedicated nodes, and the RISC System/6000® Scalable POWERparallel® System (SP).

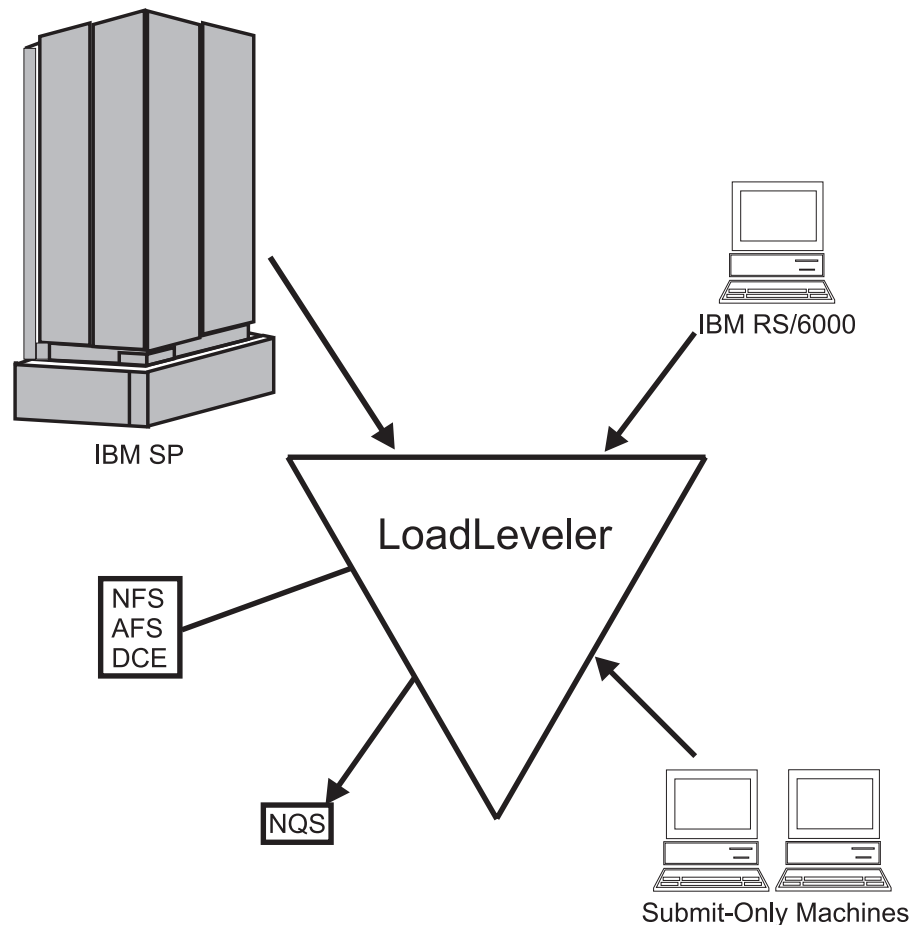


Figure 1. Example of a LoadLeveler configuration

In addition, LoadLeveler can schedule jobs written for NQS to run on machines outside of the 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. This type of machine is further discussed in "Roles of machines" on page 7.

### LoadLeveler basics

LoadLeveler has three 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:

1. Command line interface
  - The command line interface gives you access to basic job and administrative functions.
  - For more information see: “Chapter 2. LoadLeveler command line interface” on page 19
2. A Graphical User Interface (GUI)
  - LoadLeveler’s GUI 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.
  - For more information see: “Chapter 3. Using the Graphical User Interface” on page 21
3. Application Programming Interface (API)
  - The Application Programming Interface allows application programs written by users and administrators to interact with the LoadLeveler environment.
  - For more information see: “Chapter 4. LoadLeveler API interface” on page 33

All three types of interfaces permit different levels of access to users and administrators. User access is typically restricted to submitting and managing individual jobs, while administrative access allows setting up system configurations, job scheduling, and accounting.

- For background information on user and administrator tasks see “Network job management and job scheduling systems” on page 5
- For detailed information on user tasks see:
  - “Chapter 5. Submitting and managing jobs” on page 39
  - “Chapter 6. Special considerations for parallel jobs” on page 49
- For detailed information on administrator tasks see:
  - “Chapter 7. Administering and configuring LoadLeveler” on page 59
  - “Chapter 8. Administration tasks for parallel jobs” on page 71
  - “Chapter 9. Gathering job accounting data” on page 75
  - “Chapter 10. Routing jobs to NQS machines” on page 79

Using either the command line or the Graphical User Interface, 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.

For more information on job command files see:

- “Job definition” on page 5
- “Chapter 5. Submitting and managing jobs” on page 39

For more information on keywords used in job command files see:

- “Building a job command file” on page 39
- “Keyword considerations for parallel jobs” on page 49
- “Chapter 11. Job command file keywords” on page 85
- “Chapter 12. Administration and Configuration file keywords” on page 111

For more information on commands used in job command files see:

- “Submitting a job command file” on page 42
- “Managing jobs” on page 42
- “Chapter 14. Commands” on page 137

For more information on daemons see:

- “LoadLeveler daemons” on page 8
- “Chapter 13. LoadLeveler daemons and job states” on page 129

For more information on job classes see:

- “How LoadLeveler schedules jobs” on page 7
- “Administration file structure and syntax” on page 62
- “Setting up a class for parallel jobs” on page 73

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. For more information on scheduler options see:

- “How LoadLeveler schedules jobs” on page 7
- “Scheduler considerations” on page 49
- “Scheduling considerations for parallel jobs” on page 71
- “Choosing a scheduler” on page 335

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. For more information on consumable resources see:

- “Consumable resources” on page 14
- “Managing jobs that consume resources” on page 47
- “Chapter 9. Gathering job accounting data” on page 75

---

## How LoadLeveler works

This section introduces some basic job scheduling concepts.

### Network job management and job scheduling systems

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

## How LoadLeveler works

more job steps, where each job step depends upon the completion status of a previous job step. For example, Figure 2 illustrates a stream of job steps:

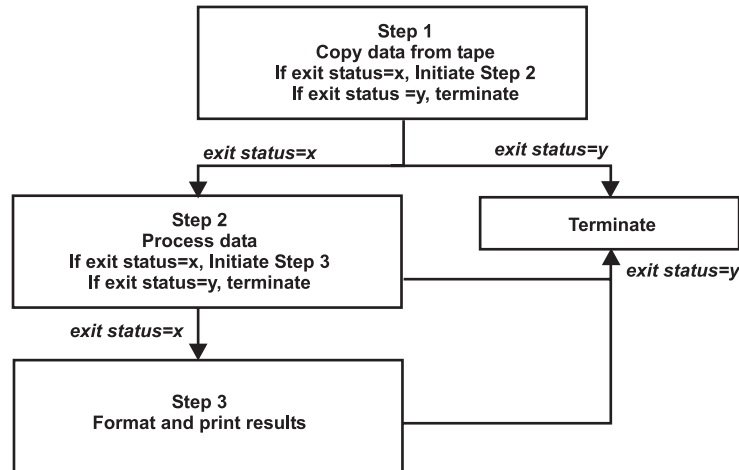


Figure 2. LoadLeveler job steps

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.

LoadLeveler will execute two different types of parallel jobs:

```
job_type = PVM
job_type = parallel
```

With a `job_type` of PVM, LoadLeveler supports a PVM API to allocate nodes and launch tasks. With a `job_type` of parallel, LoadLeveler interacts with Parallel Operating Environment (POE) to allocate nodes, assign tasks to nodes, and launch tasks.

### Machine definition

In order 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. The following types of machines can comprise a LoadLeveler cluster:

- RISC System/6000 (and compatible hardware running AIX)
- SP System

## How LoadLeveler works

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. These roles are described below:

- *Scheduling Machine:* When a job is submitted, it gets placed in a queue managed by a scheduling machine. This machine contacts another machine that serves as the central manager for the entire LoadLeveler cluster. (This role is described below). 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, which are described below.
- *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.

**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 machine(s). 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

## How LoadLeveler works

cluster—until it is ready to be dispatched to run on another machine, as shown in Figure 3.

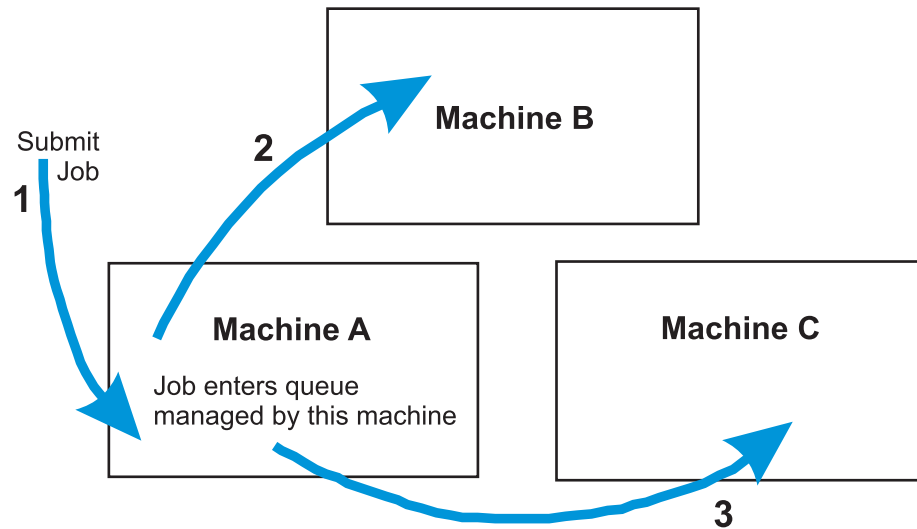


Figure 3. Job queues

Once LoadLeveler examines a job to determine its required resources, the job is dispatched to a machine to be processed. Arrows 2 and 3 indicate that the 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. For more information on job classes, see “Step 3: Specify class stanzas” on page 319.

You can specify which types of jobs will run on a machine by specifying the type(s) of job classes the machine will support. For more information, see “Step 1: Specify machine stanzas” on page 310.

LoadLeveler also examines a job’s *priority* in order 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. For more information on job priority, see “Setting and changing the priority of a job” on page 44.

## LoadLeveler daemons

LoadLeveler has its own set of daemons that control the processes moving jobs through the LoadLeveler cluster and a master daemon that manages the process control daemons. Table 1 on page 9 summarizes these daemons.



Table 1. LoadLeveler daemons

Daemon	Description	More information on page:
LoadL_master	Referred to as the master daemon. Runs on all machines in the LoadLeveler cluster and manages other daemons.	129
LoadL_schedd	Referred to as the schedd daemon. Receives jobs from the <b>llsubmit</b> command and manages them on machines selected by the negotiator daemon (as defined by the administrator).	129
LoadL_startd	Referred to as the startd daemon. Monitors job and machine resources on local machines and forwards information to the negotiator daemon. <b>Note:</b> The startd daemon spawns the starter process (LoadL_starter) which manages running jobs on the executing machine. For more information see “The starter process” on page 132.	130
LoadL_negotiator	Referred to as the negotiator daemon. Monitors the status of each job and machine in the cluster. Responds to queries from <b>llstatus</b> and <b>llq</b> commands. Runs on the central manager machine.	133
LoadL_kbdd	Referred to as the keyboard daemon. Monitors keyboard and mouse activity.	133
LoadL_GSmonitor	Referred to as the gsmonitor daemon. Monitors for down machines based on the heartbeat responses of the <b>MACHINE_UPDATE_INTERVAL</b> time period.	133

## The LoadLeveler job cycle

Figure 4 on page 10 illustrates the information flow through the LoadLeveler cluster:

## How LoadLeveler works

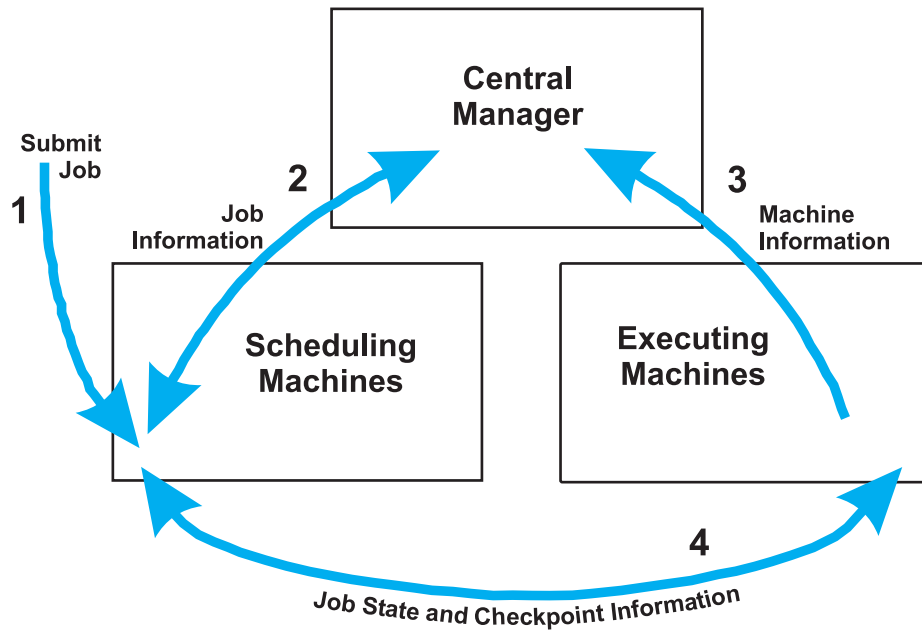


Figure 4. High-level job flow

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.

Figure 4 is broken down into the following more detailed diagrams illustrating how LoadLeveler processes a job.

1. Submit a LoadLeveler job:

## How LoadLeveler works

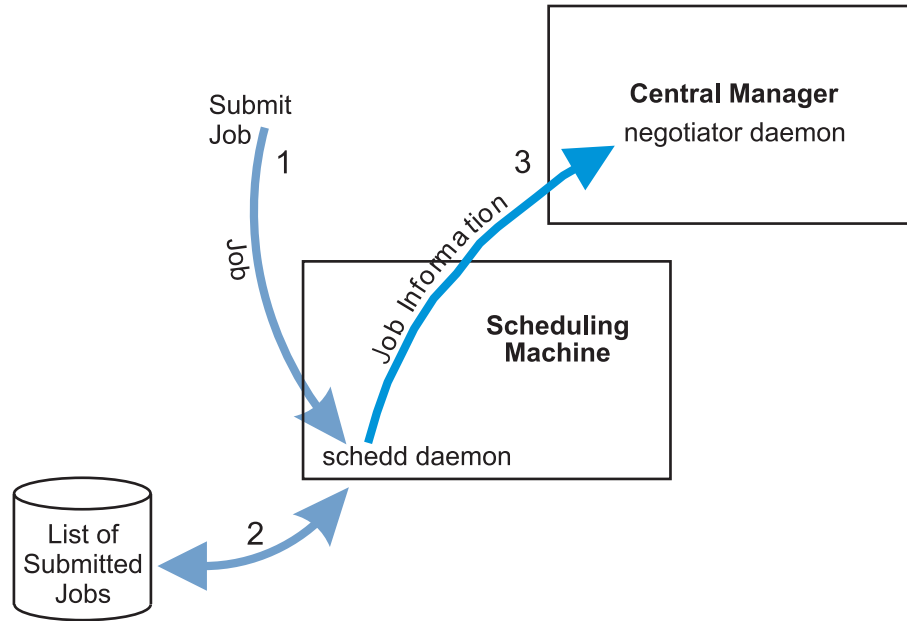


Figure 5. Job is submitted to LoadLeveler

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.

2. Permit to run:

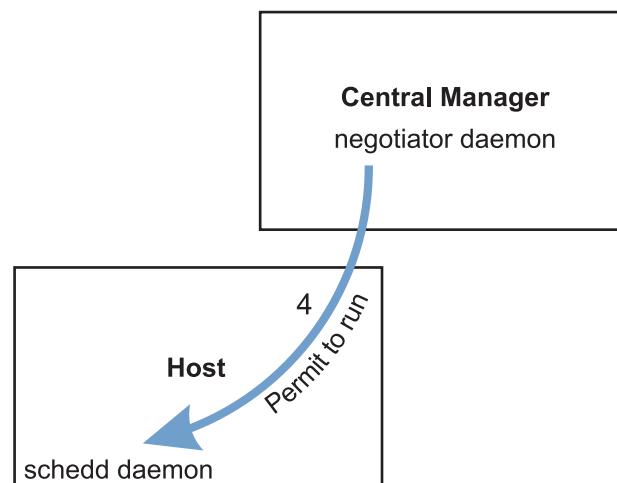


Figure 6. LoadLeveler authorizes the job

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

## How LoadLeveler works

*permit to run*. Once this is done, the job is considered Pending or Starting. (See “LoadLeveler job states” on page 13 for more information.)

3. Prepare to run:

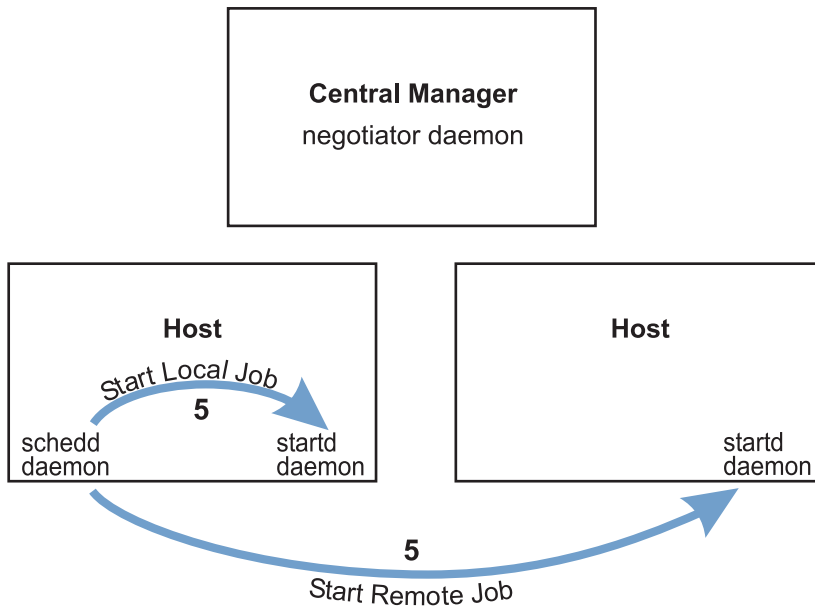


Figure 7. LoadLeveler prepares to run the job

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 from which the job was submitted) or a remote machine (another machine in the cluster).

4. Initiate job:

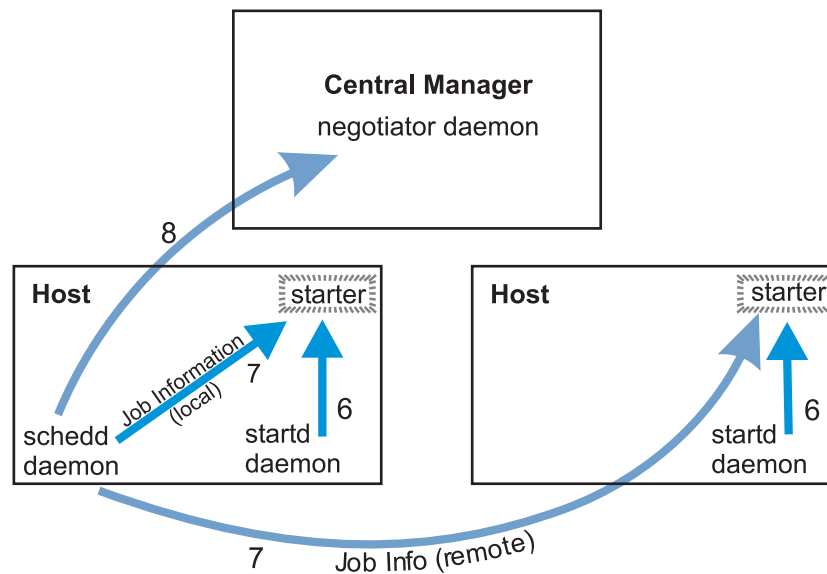


Figure 8. LoadLeveler starts the job

The arrows in Figure 8 illustrate the following:

## How LoadLeveler works

- The two arrows numbered 6 indicate that the **startd** daemon on the executing machine, spawns a **starter** process and awaits more work.
- The two arrows numbered 7 indicate 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. (See “LoadLeveler job states” for more information.)

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

### 5. Complete job:

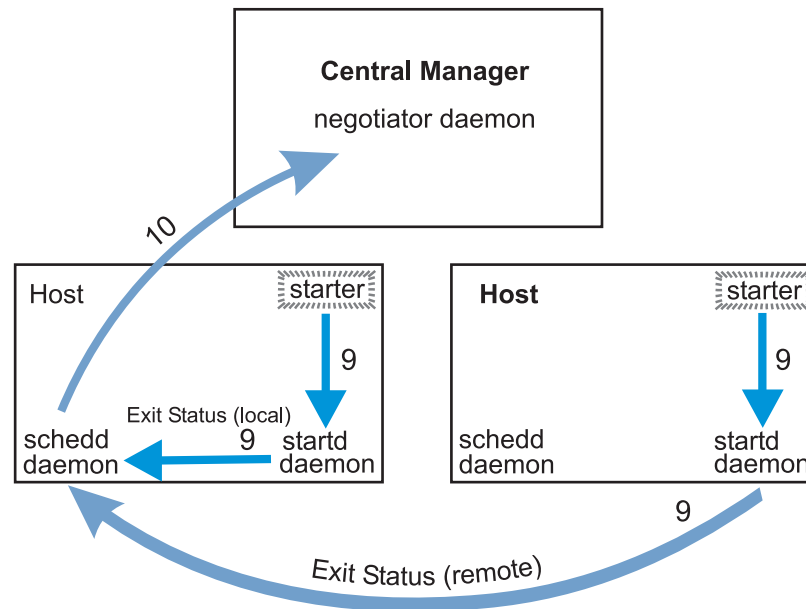


Figure 9. LoadLeveler completes the job

The arrows in Figure 9 illustrate the following:

- The arrows numbered 9 indicate that when the job completes, the starter process notifies the startd daemon, and the startd daemon notifies the schedd daemon.
- Arrow 10 indicates that the schedd daemon examines the information it has received and forwards it to the negotiator daemon.

## LoadLeveler job states

As LoadLeveler processes a job, the job moves through various states. Possible job states are listed in Table 2 and detailed in the appendix under “Job states” on page 134. For more information about the daemons controlling these job states see “Daemons” on page 129.

Table 2. Job states

Job state	Abbreviation	Details on page:
Canceled	CA	134
Checkpointing	CK	134
Completed	C	134
Complete Pending	CP	134

## How LoadLeveler works

Table 2. Job states (continued)

Job state	Abbreviation	Details on page:
Deferred	D	134
Idle	I	134
Not Queued	NQ	134
Not Run	NR	135
Pending	P	135
Preempted	E	135
Preempt Pending	EP	135
Rejected	X	135
Reject Pending	XP	135
Removed	RM	135
Remove Pending	RP	135
Resume Pending	MP	135
Running	R	135
Starting	ST	135
System Hold	S	135
User & System Hold	HS	136
Terminated	TX	135
User Hold	H	136
Vacated	V	136
Vacate Pending	VP	136
<b>Note:</b> Job states that include "Pending," such as Complete Pending and Vacate Pending are intermediate, temporary states.		

## 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 WLM
- Whether all jobs submitted need to specify resources

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

### 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 285.
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 reducing them by the amount requested when a job step is scheduled, and increasing them when a consuming job step completes.
3. If you are using Gang scheduling and a job is swapped out (suspended) because it is not in the currently executing time-slice or if a job is preempted, the job continues to use all consumable resources except for ConsumableCpus and ConsumableMemory (real memory). When a job is suspended or preempted, LoadLeveler makes the ConsumableCpus and ConsumableMemory used by that job available to other jobs.

## 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. This is done for each node that a job step is assigned to execute on. LoadLeveler creates and assigns each job step to its own WLM class (based on a job step’s resource requirements). LoadLeveler then defines “resource shares” for those WLM classes. These resource shares represent the job’s resource usage in relation to the amount of resources available on the machine. Since AIX resources are only allocated to a WLM class with active processes, WLM resource percentages are calculated based on the total number of shares requested by all active WLM classes. In other words, WLM creates a desired resource entitlement for processes within each WLM class by assigning a dynamic percentage equal to the resource shares of that class divided by the total shares of all active WLM classes. It is important to note that AIX Workload Manager will only enforce these percentages when the resources are under contention.

**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

## How LoadLeveler works

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 “Considerations for integrating LoadLeveler with AIX Workload Manager” on page 68.



---

## Part 2. LoadLeveler interfaces

### Interface summary

This section describes the three primary methods LoadLeveler offers to create, submit, and manage jobs:

- Command line interface
- Graphical User Interface
- Application Programming Interface

<b>Chapter 2. LoadLeveler command line interface</b> . . . . .	19
Summary of LoadLeveler commands . . . . .	20
<b>Chapter 3. Using the Graphical User Interface</b> . . . . .	21
Starting the Graphical User Interface . . . . .	21
Specifying options . . . . .	21
The LoadLeveler main window . . . . .	21
Getting help using the Graphical User Interface . . . . .	23
Differences between LoadLeveler's Graphical User Interface and other	
Graphical User Interfaces . . . . .	23
Graphical User Interface typographic conventions . . . . .	23
Task step conventions . . . . .	23
Selection table and decision table conventions . . . . .	23
Menu selection conventions . . . . .	24
Building jobs with the Graphical User Interface . . . . .	24
Customizing the Graphical User Interface . . . . .	24
Syntax of an Xloadl file . . . . .	24
Modifying windows and buttons . . . . .	24
Creating your own pull-down menus . . . . .	25
Example – creating a new pull-down . . . . .	26
Customizing fields on the Jobs window and the Machines window . . . . .	26
Modifying help panels . . . . .	26
Administrative uses for the Graphical User Interface . . . . .	27
Job related administrative actions . . . . .	27
Machine related administrative actions . . . . .	29
<b>Chapter 4. LoadLeveler API interface</b> . . . . .	33
Summary of LoadLeveler APIs . . . . .	33



---

## Chapter 2. LoadLeveler command line interface

LoadLeveler provides two types of commands: those that are available to all users of LoadLeveler, and those that are reserved for LoadLeveler administrators. If DCE is not used, then administrators are identified by the `LOADL_ADMIN` keyword in the configuration file. If DCE is enabled with `DCE_ENABLEMENT=TRUE`, the members of the DCE group specified by the keyword `DCE_ADMIN_GROUP` are LoadLeveler administrators.

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 3 on page 20 summarizes the LoadLeveler commands. Detailed descriptions of these commands including syntax, flags, and usage examples can be found in “Chapter 14. Commands” on page 137.

## Summary of LoadLeveler commands

Table 3. LoadLeveler command summary

Command	Description	Who Can Issue?	For More Information
<b>llacctmrg</b>	Collects all individual machine history files together into a single file.	Administrators	See page 138
<b>llcancel</b>	Cancels a submitted job.	Users and Administrators	See page 140
<b>llclass</b>	Returns information about LoadLeveler classes.	Users and Administrators	See page 144
<b>llctl</b>	Controls daemons on one or more machines in the LoadLeveler cluster.	Administrators	See page 148
<b>llckpt</b>	Used to checkpoint a single job step.	Users and Administrators	See page 142
<b>lldcegrpmain</b>	Sets up DCE groups and principal names.	DCE Administrators	See page 153
<b>llexstSDR</b>	Extracts adapter information from the system data repository (SDR).	Users and Administrators	See page 155
<b>llfavorjob</b>	Raises one or more jobs to the highest priority, or restores original priority.	Administrators	See page 159
<b>llfavoruser</b>	Raises job(s) submitted by one or more users to the highest priority, or restores original priority.	Administrators	See page 160
<b>llhold</b>	Holds or releases a hold on a job.	Users and Administrators	See page 161
<b>llinit</b>	Initializes a new machine as a member of the LoadLeveler cluster.	Administrators	See page 163
<b>llmatrix</b>	Returns GANG matrix information in the LoadLeveler cluster when GANG scheduling is used.	Users and Administrators	See page 165
<b>llmodify</b>	Changes attributes or characteristics of a submitted job step.	Users and Administrators	See page 168
<b>llpreempt</b>	Preempts a specified job step.	Administrators	See page 170
<b>llprio</b>	Changes the user priority of a submitted job step.	Users and Administrators	See page 171
<b>llq</b>	Queries the status of LoadLeveler jobs.	Users and Administrators	See page 173
<b>llstatus</b>	Queries the status of LoadLeveler machines.	Users and Administrators	See page 191
<b>llsubmit</b>	Submits a job.	Users and Administrators	See page 200
<b>llsummary</b>	Returns resource information on completed jobs.	Users and Administrators	See page 202

---

## Chapter 3. Using the Graphical User Interface

This chapter provides introductory information on the LoadLeveler Graphical User Interface (GUI). The LoadLeveler GUI is used to build jobs and submit them for processing. The procedures for completing the required tasks are detailed in the appendix under “Using the Graphical User Interface” on page 293.

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.

Beginning on page 24 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 Graphical User Interface

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)

### Specifying 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 Graphical User Interface” on page 24 for more information.

### The LoadLeveler main window

LoadLeveler’s main window has three sub-windows, titled Jobs, Machines, and Messages, as shown in Figure 10 on page 22. Each of these sub-windows has its own menu bar.

## GUI basics

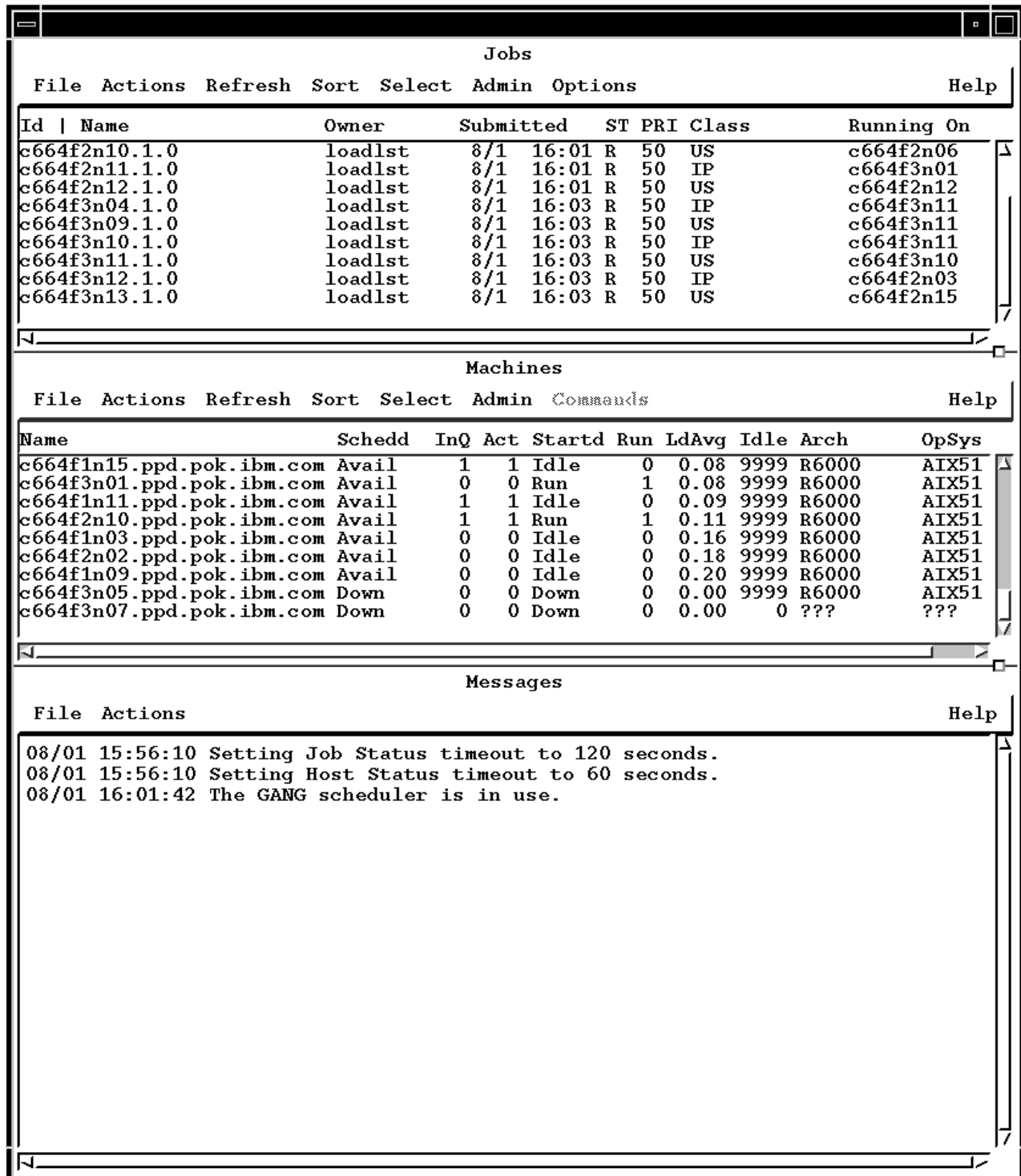


Figure 10. Main window of the LoadLeveler GUI

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 Graphical User Interface

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 executable. Otherwise, some GUI functions may not work correctly.

## Differences between LoadLeveler's Graphical User Interface 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.

## Graphical User Interface typographic conventions

This book uses the following typographic conventions 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. The following is an example of a selection table:

To	Do This
Submit a job	Refer to "Step 3: Submit a job command file" on page 303.
Cancel a job	Refer to "Step 9: Cancel a job" on page 305.

Decision tables present a question or series of questions before indicating the action. The following is an example of a decision table:

Did the job you submitted complete processing?
--

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**

### Building jobs with the Graphical User Interface

You can use the Graphical User Interface to accomplish a variety of tasks. The procedures for each task is detailed in "Using the Graphical User Interface" on page 293.

---

## Customizing the Graphical User Interface

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 section 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. This is **/usr/lpp/LoadL/full/lib** and **/usr/lpp/LoadL/so/lib**, respectively. These files contain the default values for the graphical user interface. This section 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.



## Customizing the GUI

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. The following table lists the windows and their respective identifiers:

Table 4. Window identifiers in the Xloadl file

Window	Identifier
Account Report Data	reporter
Build a Job	builder
Checkpoint	ckpt
Jobs	job_status
Limits	limits
Machines	machine_status
Messages	message_area
Network	network
Nodes	nodes
Preferences	preferences
PVM	pvm
Requirements	requirements
Script	script
Submit a Job	submit

The following table 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.

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.
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.

## Customizing the GUI

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 11. Creating a new pull-down menu*

The **Xloadl** definitions shown in the Figure 11 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:

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.
<b>Note:</b> * The work area and height depend upon your screen limitations.		

## Administrative uses for the Graphical User Interface

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 section. To find out how to grant someone administrative authority, see “Step 1: Define LoadLeveler administrators” on page 333.

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, as shown in Figure 10 on page 22, 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. 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 for whom you want to favor their jobs.

**Press** OK

## Administrative uses of the GUI

### 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.

**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 Gang 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**

### Undo Preempt

Available only when using the Gang scheduler. Undo preempt 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 -u** command.

**Select** One or more jobs from the Jobs window

**Select** **Admin** pull-down menu from the Jobs window

**Select** **Undo Preempt**

### Prevent Preempt

Available only when using the Gang scheduler. Prevent Preempt allows you to place the selected running job into a non-preemptable state and to preempt all other jobs running on the same nodes. This corresponds to the **llmodify -x 99** command.

**Note:** Use Modify Execution Factor to remove this protection from preemption.

**Select** One job from the Jobs window

**Select** **Admin** pull-down menu from the Jobs window

**Select** **Prevent Preempt**

### Modify Execution Factor

Available only when using the Gang scheduler. Modify Execution Factor allows you to modify the execution factor of the selected jobs. This corresponds to the **llmodify -x [123]** command.

**Select** One or more jobs from the Jobs window

**Select** **Admin** pull-down menu from the Jobs window

**Select** **Modify Execution Factor ...**

▲The Modify Jobs window appears

**Type in**

An execution factor with an integer value between 1 and 3

**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 section.

### Start All

Starts LoadLeveler on all machines listed in machine stanzas beginning with the central manager. Use this option when specifying alternate central managers.

**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.

## Administrative uses of the GUI

**Select Admin** from the Machines window.

**Select Start LoadLeveler**

### 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. Use this option when specifying alternate central managers.

**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 fileset 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 machines 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

## Administrative uses of the GUI

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 be resumed. If you select **startd by class**, a window appears which allows you to select classes to be resumed.

**Purge** Allows you to instruct the schedd daemon to purge all transactions pending on the selected machines. The selected machines must be machines that are not returning to the LoadLeveler cluster. This option is intended for recovery and clean up after a machine has permanently crashed or was inadvertently removed from the LoadLeveler cluster before all activity on it was quiesced.

**Select** One or more machines that are no longer available but still have queued transactions.

**Select Admin** pull-down menu from the Machines window

**Select**

**Purge**

▲ The Purge Machine window opens

**Type in**

The name of the machine running the schedd daemon which purges the transactions

**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.

## Administrative uses of the GUI

**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.

### 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.

### 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**



---

## Chapter 4. LoadLeveler API interface

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. Table 5 summarizes the LoadLeveler APIs.

Details on the following APIs can be found in Chapter 15. Application Programming Interfaces (APIs):

- “Accounting API” on page 215
- “Error Handling API” on page 259
- “Checkpointing API” on page 218
- “Submit API” on page 268
- “Data Access API” on page 223
- “Parallel Job API” on page 260
- “Workload Management API” on page 270
- “Query API” on page 265
- “User exits” on page 282

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** 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.

---

### Summary of LoadLeveler APIs

Table 5 summarizes LoadLeveler APIs.

Table 5. Application Programming Interfaces

API	Description	Who Can Issue?	For More Information
<b>Accounting</b>	Generates accounting reports	<b>GetHistory</b> — Users and LoadLeveler Administrators	See page 215
<b>Checkpointing</b>	Checkpoint LoadLeveler jobs	All APIs ( <b>ll_init_ckpt</b> , <b>ll_ckpt</b> , <b>ll_set_ckpt_callbacks</b> , <b>ll_unset_ckpt_callbacks</b> ) — Users and LoadLeveler Administrators	See page 218
<b>Data Access</b>	Accesses LoadLeveler objects and allows you to retrieve data from objects	All subroutines ( <b>ll_query</b> , <b>ll_set_request</b> , <b>ll_reset_request</b> , <b>ll_get_objs</b> , <b>ll_get_data</b> , <b>ll_next_obj</b> , <b>ll_free_objs</b> , <b>ll_deallocate</b> ) — Users and LoadLeveler Administrators	See page 223

## Summary of LoadLeveler APIs

Table 5. Application Programming Interfaces (continued)

API	Description	Who Can Issue?	For More Information
<b>Error Handling</b>	Converts an error object into an error message	<b>ll_error</b> — Users and LoadLeveler Administrators	See page 259
<b>Parallel Job</b>	Tools for parallel job submission	All subroutines ( <b>ll_get_hostlist</b> , <b>ll_start_host</b> ) — Users and LoadLeveler Administrators	See page 260
<b>Query</b>	Provides information about the jobs and machines in the LoadLeveler cluster	All subroutines ( <b>ll_get_jobs</b> , <b>ll_free_jobs</b> , <b>ll_get_nodes</b> , <b>ll_free_nodes</b> ) — Users and LoadLeveler Administrators	See page 265
<b>Submit</b>	Submits jobs to LoadLeveler	<ul style="list-style-type: none"> <li>• <b>llsubmit</b> — Users and LoadLeveler Administrators</li> <li>• <b>llfree_job_info</b> — Users and LoadLeveler Administrators</li> </ul>	See page 268
<b>Workload Management</b>	Performs LoadLeveler control operations and provides tools to be used with an external scheduler	<ul style="list-style-type: none"> <li>• <b>ll_control</b> <ul style="list-style-type: none"> <li>– LoadLeveler Administrators only: <ul style="list-style-type: none"> <li><b>LL_CONTROL_DRAIN</b></li> <li><b>LL_CONTROL_DRAIN_SCHEDD</b></li> <li><b>LL_CONTROL_DRAIN_STARTD</b></li> <li><b>LL_CONTROL_FAVOR_JOB</b></li> <li><b>LL_CONTROL_FAVOR_USER</b></li> <li><b>LL_CONTROL_FLUSH</b></li> <li><b>LL_CONTROL_HOLD_SYSTEM</b></li> <li><b>LL_CONTROL_PURGE_SCHEDD</b></li> <li><b>LL_CONTROL_RECONFIG</b></li> <li><b>LL_CONTROL_RECYCLE</b></li> <li><b>LL_CONTROL_RESUME</b></li> <li><b>LL_CONTROL_RESUME_SCHEDD</b></li> <li><b>LL_CONTROL_RESUME_STARTD</b></li> <li><b>LL_CONTROL_STOP</b></li> <li><b>LL_CONTROL_SUSPEND</b></li> <li><b>LL_CONTROL_UNFAVOR_JOB</b></li> <li><b>LL_CONTROL_UNFAVOR_USER</b></li> </ul> </li> <li>– Users and LoadLeveler Administrators: <ul style="list-style-type: none"> <li><b>LL_CONTROL_HOLD_USER</b></li> <li><b>LL_CONTROL_PRIO_ABS</b></li> <li><b>LL_CONTROL_PRIO_ADJ</b></li> <li><b>LL_CONTROL_START</b></li> </ul> </li> </ul> </li> <li>• <b>ll_modify</b> — Users and LoadLeveler Administrators (some functions limited to Gang scheduling)</li> <li>• <b>ll_preempt</b> — LoadLeveler Administrators (function limited to Gang and external schedulers)</li> <li>• <b>ll_start_job</b> — LoadLeveler Administrators, for use with external schedulers only</li> <li>• <b>ll_terminate_job</b> — LoadLeveler Administrators only</li> </ul>	See page 270

Table 5. Application Programming Interfaces (continued)

API	Description	Who Can Issue?	For More Information
User exits	Allows: <ul style="list-style-type: none"> <li>• Handling DCE security</li> <li>• Handling AFS tokens</li> <li>• Filtering job scripts</li> <li>• Overriding default mail program</li> <li>• Job prolog</li> <li>• Job epilog</li> </ul>	LoadLeveler administrators only <b>Note:</b> Other User exits are available with functions limited to specific APIs.	See page 282

## Summary of LoadLeveler APIs

---

## Part 3. User tasks

### User task summary

This section provides the LoadLeveler user with detailed descriptions for creating a job.

<b>Chapter 5. Submitting and managing jobs</b>	39
Building a job command file	39
Job command file syntax	39
Serial job command file	40
Parallel job command file	40
Using multiple steps in a job command file	40
Submitting a job command file	42
Managing jobs	42
Editing job command files	43
Querying the status of a job	43
Querying multiple LoadLeveler clusters	43
Placing and releasing a hold on a job	44
Cancelling a job	44
Checkpointing a job	44
Setting and changing the priority of a job	44
User priority	44
System priority	44
How does a job's priority affect dispatching order?	45
Working with machines	45
Run-time environment variables	46
Managing jobs that consume resources	47
Specifying the consumption of resources by a job step	47
Displaying currently available resources	47
 <b>Chapter 6. Special considerations for parallel jobs</b>	 49
Supported parallel environments	49
Keyword considerations for parallel jobs	49
Scheduler considerations	49
Task assignment considerations	50
node and total_tasks	50
node and tasks_per_node	51
blocking	51
unlimited blocking	51
task_geometry	52
Submitting jobs that use striping	52
Understanding striping	53
Using striping	54
Requesting striping using IP mode	54
Requesting striping using user space mode	54
Running interactive POE jobs	54
Job command file examples	54
Obtaining status of parallel jobs	54
Obtaining allocated host names	55



---

## Chapter 5. Submitting and managing jobs

This chapter tells you how to submit jobs to LoadLeveler. In general, the information in this chapter applies both to serial jobs and to parallel jobs. For information specific to parallel jobs, see “Chapter 6. Special considerations for parallel jobs” on page 49.

Many LoadLeveler actions, such as submitting a job, can be done in either of the following ways:

- Using LoadLeveler commands (this chapter)
  - For additional information on LoadLeveler commands, see “Chapter 2. LoadLeveler command line interface” on page 19 and “Chapter 14. Commands” on page 137.
- Using the LoadLeveler Graphical User Interface
  - For additional information on the LoadLeveler GUI, see “Chapter 3. Using the Graphical User Interface” on page 21 and “Using the Graphical User Interface” on page 293.
- Using LoadLeveler APIs
  - For additional information on LoadLeveler APIs, see “Chapter 4. LoadLeveler API interface” on page 33 and “Chapter 15. Application Programming Interfaces (APIs)” on page 215.

---

### 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 section. 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 “Chapter 11. Job command file keywords” on page 85.
- 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 109 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.

### 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 @.

## Building a job command file

- 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 Figure 43 on page 409 and Figure 44 on page 411 for examples using 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 12 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.

```
# 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 12. 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 “Chapter 6. Special considerations for parallel jobs” on page 49.

### 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.



## Building a job command file

For example, Figure 13 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 13. Job command file with multiple steps

In Figure 13, 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 13, 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 14, 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 14. Job command file with multiple steps and one executable

See “Additional examples of building job command files” on page 407 for more information.

### 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 43 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 the job a three part identifier and also sets environment variables for the job.

The identifier consists of the following:

- Machine name: the name of the machine that schedules the job. This is not necessarily the name of the machine that runs the job.
- Job ID: an identifier given to a group of job steps that were initiated from the same job command file. For example, if you created a job command file that submitted the same program five times (using five queue statements) possibly with different input and output, each program would have the same job ID.
- Step ID: an identifier 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 a unique step ID but the same job ID.

For an example of submitting a job, see “Step 3: Submit a job” on page 405.

**Submitting a Job Command File to be Routed to NQS Machines:** When submitting a job command file to be routed to an NQS machine for processing, the job command file must contain the shell script to be submitted to the NQS node. NQS accepts only shell scripts; binaries are not allowed. All options in the command file pertaining to scheduling are used by LoadLeveler to schedule the job. When the job is dispatched to the node running the specified NQS class, the LoadLeveler options pertaining to the runtime environment are converted to NQS options and the job is submitted to the specified NQS queue. For more information on submitting jobs to NQS, see Figure 18 on page 79. For more information on the **llsubmit** command, see “llsubmit - Submit a job” on page 200.

**Submitting a Job Command File 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 *LoadLeveler Installation Memo*, or consult the appropriate README file.

In addition to allowing you to submit jobs, the submit-only feature allows you to cancel and query jobs from a submit-only machine.

---

## Managing jobs

This section discusses:

- “Editing job command files” on page 43
- “Querying the status of a job” on page 43
- “Placing and releasing a hold on a job” on page 44
- “Cancelling a job” on page 44
- “Checkpointing a job” on page 44

- “Setting and changing the priority of a job” on page 44
- “Working with machines” on page 45

## 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.

## 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 job owner. 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 “Step 4: Display the status of a job” on page 405.

**Querying the Status of a Job Running on an NQS Machine:** If your job command file was routed to an NQS machine for processing, you can obtain its status by using either the GUI or the **llq** command. Keep in mind that a machine in the LoadLeveler cluster monitors the NQS machine where your job is running. The status you see on the GUI (or from **llq**) is generated by the machine in the LoadLeveler cluster. Since LoadLeveler only checks the NQS machine for status periodically, the status of the job on the NQS machine may change before LoadLeveler has an opportunity to update the GUI. If this happens, NQS will notify you, before LoadLeveler notifies you, regarding the status of the job.

**Querying the Status of a Job Using a Submit-Only Machine:** A submit-only machine, in addition to allowing you to submit and cancel jobs, 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 173.

## Querying multiple LoadLeveler clusters

This section applies only to those installations having more than one LoadLeveler cluster.

Using the **LOADL\_CONFIG** environment variable, you can query, submit, or cancel jobs in multiple LoadLeveler clusters. The **LOADL\_CONFIG** environment variable allows you to specify that the master configuration file be located in a directory other than the home directory of the **loadl** user ID. The file that **LOADL\_CONFIG** points to must be in the **/etc** directory.

You need to set up your own master configuration file to point to the location of the LoadLeveler user ID, group ID, and configuration files. By default, the location of the master file is **/etc/LoadL.cfg**.

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 “default” configuration files, 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:

```
$ llq
```

## Managing jobs

The above command uses the configuration from **/etc/LoadL.cfg** (this is determined by the **LOADL\_CONFIG** environment variable). To send a query to the scheduler defined in the configuration file of **/etc/othercluster.cfg**, enter:

```
$ env LOADL_CONFIG=/etc/othercluster.cfg llq
```

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

## 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 “Step 6: Hold a job” on page 406 and “Step 7: Release a hold on a job” on page 406.

As a user or an administrator, you can also use the **startdate** keyword described in “startdate” on page 106 to place a hold on a job. This keyword allows you to specify when you want to run a job.

## Cancelling 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 and jobs routed to NQS. Note that you can also cancel jobs from a submit-only machine.

## 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. For a detailed explanation of checkpointing, see “Step 14: Enable checkpointing” on page 356.

## 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. You can use the **llprio** command to change job priorities. See “llprio - Change the user priority of submitted job steps” on page 171 for more information. This section discusses the different types of priorities and how LoadLeveler uses these priorities when considering jobs for dispatch.

### User priority

Every job has a user priority associated with it. This priority, which can be specified by the user in the job command file, is a number between 0 and 100 inclusively. A job with a higher priority runs before a job with a lower priority (when both jobs are owned by the same user). The default user priority is 50. Note that this is not the UNIX *nice* priority.

### System priority

Every job has a system priority associated with it. This priority is specified in LoadLeveler's configuration file using the **SYSPRIO** expression.

**Understanding the SYSPRIO expression:** SYSPRIO is evaluated by LoadLeveler to determine the overall system priority of a job. A system priority value is assigned when the negotiator adds the new job to the queue of jobs eligible for dispatch.

The **SYSPRIO** expression can contain class, group, and user priorities, as shown in the following example:

$\text{SYSPRIO} : (\text{ClassSysprio} * 100) + (\text{UserSysprio} * 10) + (\text{GroupSysprio} * 1) - (\text{QDate})$

For more information on the system priority expression, including all the variables you can use in this expression, see “Step 6: Prioritize the queue maintained by the negotiator” on page 343.

### How does a job’s priority affect dispatching order?

LoadLeveler schedules jobs based on the *adjusted system priority*, which takes in account both system priority and user priority. Jobs with a higher adjusted system priority are scheduled ahead of jobs with a lower adjusted system priority. In determining which jobs to run first, LoadLeveler does the following:

1. Assigns all jobs a SYSPRIO at job submission time.
2. Orders jobs first by SYSPRIO.
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.

For example, Table 6 represents 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 6. 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

## Working with machines

Throughout this book, the terms *workstation*, *machine*, and *node* refer to the machines in your cluster. See “Machine definition” on page 6 for information on the roles these machines can play.

You can perform the following types of tasks related to machines:

## Managing jobs

- 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 "Step 8: Display the status of a machine" on page 406.

- 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.

---

## 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 286.

### **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\_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 “Step 3: Specify class stanzas” on page 319.

**LOADL\_STEP\_OUT**

The file used for standard output (stdout).

**LOADL\_STEP\_OWNER**

The job step owner.

**LOADL\_STEP\_TYPE**

The job type (SERIAL, PARALLEL, PVM3, or NQS)

---

## Managing jobs that consume resources

### Specifying the consumption of resources by 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 50.

### 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

## Managing jobs that consume resources

the LoadLeveler cluster. The user can specify a hostlist with the **llstatus** command to display only the consumable resources associated with specific hosts.



---

## Chapter 6. Special considerations for parallel jobs

This section describes special considerations for submitting and managing parallel jobs. For information on setting up and planning for parallel jobs, see “Chapter 8. Administration tasks for parallel jobs” on page 71.

---

### Supported parallel environments

LoadLeveler allows you to schedule parallel batch jobs that have been written using the following:

- IBM Parallel Environment Library (POE/MPI/LAPI) 3.2
- Parallel Virtual Machine (PVM) 3.3 (RS/6000 architecture)
- Parallel Virtual Machine (PVM) 3.3.11+ (SP2MPI architecture)

---

### Keyword considerations for parallel jobs

#### Scheduler considerations

Several LoadLeveler job command language keywords are associated with parallel jobs. Whether a keyword is appropriate is dependent upon the type of job and the type of LoadLeveler scheduler you are running.

Table 7 shows you the parallel keywords supported by LoadLeveler's Backfill and Gang schedulers, based on the type of job you are running.

*Table 7. Parallel keywords supported by the Backfill and Gang scheduler*

<b>job_type=parallel</b>	<b>job_type=pvm3</b>
network node node_usage tasks_per_node total_tasks task_geometry blocking All keywords supported for job_type=pvm3 (supported for compatibility reasons)	Adapter requirement max_processors min_processors network parallel_path

Table 8 shows you the parallel keywords supported by the default LoadLeveler scheduler, based on the type of job you are running.

*Table 8. Parallel keywords supported by the default scheduler*

<b>job_type=parallel</b>	<b>job_type=pvm3</b>
max_processors min_processors Adapter requirement	max_processors min_processors parallel_path Adapter requirement

These keywords are used in the examples in this chapter, and are described in more detail in “Chapter 11. Job command file keywords” on page 85.

If you disable the default LoadLeveler scheduler to run an external scheduler, see “Usage notes” on page 281 for an explanation of which keywords are supported.

## Keyword considerations for parallel jobs

### Task assignment considerations

You can use the following keywords 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 9. Valid combinations of task assignment keywords

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 node(s) on the list that can accept them. In this example, 14 tasks must be allocated among 3 nodes:

```
# @ node=3
# @ total_tasks=14
```

Table 10. 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. Here is how a blocking factor of 4 would work with 17 tasks:

Table 11. Blocking

Machine	Available Initiators	Assigned Tasks
N1	4	4
N2	6	5
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 12. 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.

## Keyword considerations for parallel jobs

### **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 `css0` or `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 can be transmitted and the application must wait for the device to transmit and empty the buffer before it can be refilled. Thus time is wasted by each waiting for the other. The technique of striping refers to using two 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 the 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 reverts to the buffer of the first device. Much, if not all of the buffer on the first device has been transmitted while the application wrote to the second buffer so the application waits for a minimal amount of time or possibly does not wait at all. LoadLeveler supports job requests to use striped communication with the virtual device **csss**. When one or more switch adapters are specified on a node, a **csss** adapter is automatically created. This **csss** adapter is requested on the job's network statement when striping is desired.

- **User Space Striping:** When the **csss** device is specified on a network statement with **US** mode, LoadLeveler commits resources (windows and memory) on all of the switch adapters on nodes assigned to the job. 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 **csss** 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 choice of the IP addresses is alternated 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 **csss** adapter on that node. If a multi-link address is not present on a node, the **csss** 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.

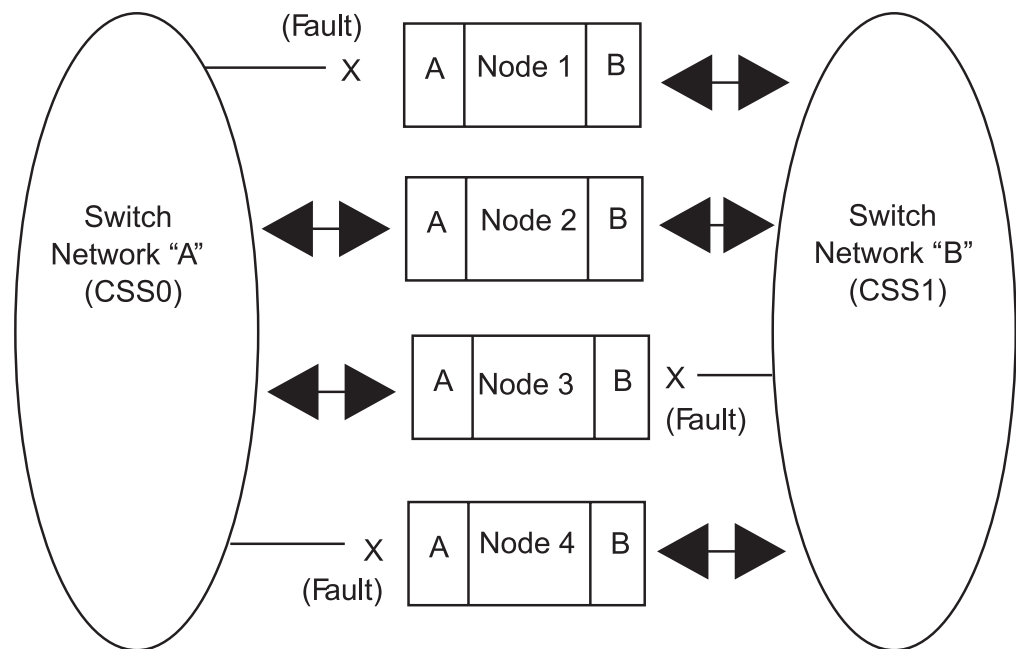
## Keyword considerations for parallel jobs

Jobs that request striping through the **csss** device (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 **css0**.

**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 **llexSDR** 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 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 **csss** 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.

## Understanding striping

For a job to successfully run using the striped adapter method, there must be a common communication path among the nodes and adapters on the system. This communication path between different nodes and adapters is called the communication fabric.



### Switch Network Adapters

Consider these sample scenarios using the network configuration as indicated in the preceding figure:

- If a three node job requests a striped adapter, it will be dispatched to Node 1, Node 2 and Node 4 where it can communicate on Network B. 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 requests **css0**, it will not be run because there are not enough connected adapters on **css0** to run the job. Notice both adapter A on Node1 and adapter A on Node 4 are both at fault.

## Keyword considerations for 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.

As you can see from these scenarios, LoadLeveler will find enough nodes on the same fabric to run the job. If enough nodes in the fabric cannot be found, no communication can take place and the job will not run.

## Using striping

To request that a job be run using striping, you must modify the **network** statement in your job command file. Shown here are examples of IP and user space network modes.

### Requesting striping using IP mode

To submit a job using IP striping, your network statement would look like this:

```
network.MPI = csss,,IP
```

### Requesting striping using user space mode

To submit a job using user space striping, your network statement would look like this:

```
network.MPI = csss,,US
```

For more information on the **network** statement, see “network” on page 96.

---

## 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.

For information on other POE environment variables, see *IBM Parallel Environment for AIX; Operation and Use, Volume 1*.

For information on running POE in a mixed cluster (some machines running LoadLeveler 3.1 and others running LoadLeveler 2.2) see “LoadLeveler 2.2 and 3.1 coexistence” on page xix.

For information on setting up POE for interactive use see “Allowing users to submit interactive POE jobs” on page 71.

---

## Job command file examples

Examples of building job command files for the parallel environment can be found under “User tasks: building parallel job command files” on page 411.

---

## 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 173 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 15.

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 15 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 = css0,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 15. Using **LOADL\_PROCESSOR\_LIST** in a shell script*

## Obtaining status of parallel jobs



---

## Part 4. Administrator tasks

### Administrator task summary

This section provides the LoadLeveler administrator with detailed descriptions for operating a LoadLeveler cluster.

<b>Chapter 7. Administering and configuring LoadLeveler</b>	59
Overview	59
Planning considerations	59
Where to begin?	60
Intermediate or beginner	60
Expert	60
Quick set up	61
Administering LoadLeveler	62
Administration file structure and syntax	62
Configuring LoadLeveler	63
The configuration files	64
Customizing the configuration file	64
Configuration file structure and syntax	64
Numerical and alphabetical constants	65
Mathematical operators	65
User-defined variables	65
LoadLeveler variables	66
Considerations for integrating LoadLeveler with AIX Workload Manager	68
Keyword summary	70
 <b>Chapter 8. Administration tasks for parallel jobs</b>	 71
Scheduling considerations for parallel jobs	71
Allowing users to submit interactive POE jobs	71
Allowing users to submit PVM jobs	72
Restrictions and limitations for PVM jobs	73
Setting up a class for parallel jobs	73
Setting up a parallel master node	74
 <b>Chapter 9. Gathering job accounting data</b>	 75
Collecting job resource data on serial and parallel jobs	75
Collecting job resource data based on machines	75
Collecting job resource data based on events	76
Collecting job resource information based on user accounts	76
Collecting the accounting information and storing it into files	77
Accounting reports	77
Job accounting setup procedure	78
 <b>Chapter 10. Routing jobs to NQS machines</b>	 79
Setting up the NQS environment	79
Designating machines to which jobs will be routed	80
NQS scripts	80
NQS machine job routing procedure	81



---

## Chapter 7. Administering and configuring LoadLeveler

This chapter tells you how to administer and configure LoadLeveler. In general, the information in this chapter applies to both serial and parallel jobs. For more specific information on parallel jobs, see “Chapter 8. Administration tasks for parallel jobs” on page 71.

---

### Overview

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

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

You can, however, have multiple local configuration files that specify information specific to individual machines. For more information on the global and local configuration files, refer to “Configuring LoadLeveler” on page 63.

Before working with these two files, you should read the following planning considerations to help you decide how to modify the files.

---

### Planning considerations

#### 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 system ID number for a user) and gid (the system ID number for a group) on every machine in the cluster. The term cluster refers to all machines mentioned in the configuration file.

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*. This ensures that the **getuid** system call returns the same user ID on both systems. (This allows 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.

## Planning considerations

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.

### Performance

You should keep the **log**, **spool**, and **execute** directories in a local file system in order to maximize performance. Also, to measure the performance of your network, consider using one of the available products, such as Toolbox/6000.

### Management

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.

### 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.

## Where to begin?

Setting up LoadLeveler involves defining machines, users, and how they interact, in such a way that LoadLeveler is able to run jobs quickly and efficiently. If you have a good deal of experience in system administration and job scheduling, you should begin by reading "Expert". If you are relatively new to job scheduling tasks, begin by reading "Intermediate or beginner".

No matter what your level of experience, it will prove worthwhile to read all the information in this chapter at some point to help you optimize LoadLeveler's performance.

### Intermediate or beginner

If you are experienced in UNIX system administration but are unfamiliar with job scheduling systems or your experience is limited, you may want to start with the section "Administration file structure and syntax" on page 62 and read to the end of this chapter. This section provides a relatively slow, step-by-step approach to administering LoadLeveler. If you would rather start up LoadLeveler quickly using mostly default characteristics, follow the procedures in "Quick set up" on page 61.

### Expert

If you are very familiar with UNIX system administration and job scheduling, and have some idea how you want to distribute your workload, go to "Quick set up" on page 61. Each step in this short procedure refers you to a detailed discussion of the task at hand. The sample configuration and administration files included in the samples subdirectory also provide assistance.

If you plan to run interactive jobs using the Parallel Operating Environment (POE) running under LoadLeveler, see "Allowing users to submit interactive POE jobs" on page 71.

## Quick set up

If you are very familiar with UNIX system administration and job scheduling, follow the steps listed in this section 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, you will have to consult the rest of this manual. This section also does not address how to configure DCE. For more information about configuring DCE for LoadLeveler, see “Step 16: Configuring LoadLeveler to use DCE security services” on page 365. For this set up, it is recommended that you use **loadl** as the LoadLeveler user ID. Afterward, you can fine tune your configuration for greater efficiency when you become more familiar with the details of LoadLeveler.

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 in **/etc/LoadL.cfg** (if this file exists). See “Configuring LoadLeveler” on page 63 for more information.
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 “Step 1: Define LoadLeveler administrators” on page 333 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 *machinename* with the actual name of the machine.)

```
machinename: 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 you entered and updates the administration and configuration files. See “Step 1: Specify machine stanzas” on page 310 for more information.

4. Issue the following command for each machine to be included in the LoadLeveler cluster. (Replace *hostname* with the actual name of the machine.)

```
llctl -h hostname start
```

Issue this command for the central manager machine first. See “llctl - Control LoadLeveler daemons” on page 148 for more information.

You can also issue the following command to start LoadLeveler on all machines beginning with the central manager. Before you issue this command, make sure all the machines are listed in the administration file. This command only affects machines that are defined in the administration file.

```
llctl -g start
```

**llctl** uses **rsh** or **remsh** to start LoadLeveler on the target machine. Therefore, the administrator using **llctl** must have rsh authority on the target machine. LoadLeveler will fail to start if any value has been set for the **MALLOCTYPE** environment variable.

### Administering LoadLeveler

This section gives a brief overview of the LoadLeveler administration file. “Customizing the administration file” on page 310 describes the procedure for modifying this file.

#### 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 “Step 1: Specify machine stanzas” on page 310 for more information.

##### User stanza

Defines LoadLeveler users and their characteristics. See “Step 2: Specify user stanzas” on page 316 for more information.

##### Class stanza

Defines the characteristics of the job classes. See “Step 3: Specify class stanzas” on page 319 for more information.

##### Group stanza

Defines the characteristics of a collection of users that form a LoadLeveler group. See “Step 4: Specify group stanzas” on page 329 for more information.

##### Adapter stanza

Defines the network adapters available on the machines in the LoadLeveler cluster. See “Step 5: Specify adapter stanzas” on page 332 for more information.

Stanzas have the following general format:

```
label: type = type_of_stanza
keyword1 = value1
keyword2 = value2
...
```

*Figure 16. Format of administration file stanzas*

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 17. Sample administration file stanzas

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 may 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 may 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.

---

## Configuring LoadLeveler

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

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

- **loadl** is the LoadLeveler user ID and the LoadLeveler group ID. LoadLeveler daemons run under this user ID in order 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.

You can run your installation with these default values, or you can change any or all of them. To override the defaults, you must update the following keywords in the **/etc/LoadL.cfg** file:

## Configuring LoadLeveler

### **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.

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 43.

## The configuration files

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. This chapter describes these configuration files and the parameters you can set.

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 “Step 11: Specify where files and directories are located” on page 351 for more information. “Customizing the global and local configuration file” on page 333 describes how to tailor this file to suit your needs.

### **Customizing the configuration file**

“Customizing the global and local configuration file” on page 333 describes the procedure for modifying the configuration file.

## Configuration file structure and 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 keywords and LoadLeveler variables” on page 115 and in “Step 17: Specify additional configuration file keywords” on page 370
- User-defined variables, described in “User-defined variables” on page 65
- LoadLeveler variables, described in “LoadLeveler variables” on page 66

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.

To continue configuration file statements, use the back-slash character (\).

In the configuration file, comments must be on a separate line from keyword statements.

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:

```
!  
* /  
- +  
< <= > >=  
== !=  
&&  
||
```

### User-defined variables

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.

## Configuring LoadLeveler

The sample configuration file shipped with the product defines and uses some “user-defined” variables.

### LoadLeveler variables

The LoadLeveler product includes variables that you can use in the configuration file. 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.

LoadLeveler provides the following variables that you can use in your configuration file statements.

#### 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)*

#### 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**.

#### ConsumableMemory

The amount of **ConsumableMemory** 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**.

#### ConsumableVirtualMemory

The amount of **ConsumableVirtualMemory** 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**.

#### Cpus

The number of CPU's installed.

#### CurrentTime

The **UNIX date**; the current system time, in seconds, since January 1, 1970, as returned by the time() function.

#### CustomMetric

Sets a relative machine priority.

#### 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.

#### 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

## Configuring LoadLeveler

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).

### **host** or **hostname**

Dynamically indicates the official name of the host machine where the program is running. **host** returns the machine name without the domain name; **hostname** returns the machine and the domain.

### **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.

### **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.

### **Memory**

The physical memory installed on the machine in megabytes.

### **MasterMachPriority**

A value that is equal to 1 for nodes which are master nodes, and is equal to 0 otherwise.

### **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.

### **PagesScanned**

The number of pages scanned per second by the page replacement algorithm of the virtual memory manager.

### **QDate**

The difference in seconds between when LoadLeveler (specifically the negotiator daemon) comes up and when the job is submitted using **lsubmit**.

### **Speed**

The relative speed of a machine.

### **State**

The state of the **startd** daemon.

### **tilde**

The home directory for the LoadLeveler userid.

### **UserPrio**

The user defined priority of the job. The priority ranges from 0 to 100, with higher numbers corresponding to greater priority.

## Configuring LoadLeveler

### VirtualMemory

The size of available swap space (free paging space) on the machine in kilobytes.

**Time:** 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).

### tm\_min

Number of minutes after the hour (0-59).

### tm\_sec

Number of seconds after the minute (0-59).

### 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)
```

### Date:

### tm\_mday

The number of the day of the month (1-31).

### tm\_wday

Number of days since Sunday (0-6).

### tm\_yday

Number of days since January 1 (0-365).

### tm\_mon

Number of months since January (0-11).

### 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.

## Considerations 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

## Configuring LoadLeveler

LoadLeveler to balance workloads at the machine level. 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 you must:

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. Using the **resources** keyword of the machine stanza, 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.
4. 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 "Step 4: Define consumable resources" on page 341.

### Keyword summary

Listings of the keywords used in administration and configuration files can be found under:

- “Administration file keywords” on page 111
- “Configuration file keywords and LoadLeveler variables” on page 115

---

## Chapter 8. Administration tasks for parallel jobs

This chapter describes administration tasks that apply to parallel jobs. For more general information on administering and configuring LoadLeveler, see “Chapter 7. Administering and configuring LoadLeveler” on page 59. For information on submitting parallel jobs, see “Chapter 6. Special considerations for parallel jobs” on page 49.

---

### Scheduling considerations for parallel jobs

For parallel jobs, LoadLeveler supports Gang scheduling and 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.

Gang and Backfill scheduling also support:

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

In addition, Gang Scheduling also supports:

- Time sharing for parallel jobs
- Resource sharing
- 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 335.

---

### Allowing users to submit interactive POE jobs

Follow the steps in this section 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 “Quick set up” on page 61 for information on defining LoadLeveler administrators.
2. Run the **llextrSDR** command to extract node and adapter information from the SDR. See “llextrSDR - Extract adapter information from the SDR” on page 155 for information on using this command.
3. Incorporate the appropriate node and adapter information into your LoadLeveler administration file stanzas.

For example, the following output represents two adapter stanzas and their corresponding machine stanza:

```
k10n09.ppd.pok.ibm.com: type = adapter
adapter_name = en0
network_type = ethernet
interface_address = 9.114.51.73
interface_name = k10n09.ppd.pok.ibm.com
```

```
k10sn09.ppd.pok.ibm.com: type = adapter
adapter_name = css0
css_type = SP_Switch_MX_Adapter
network_type = switch
interface_address = 9.114.51.137
interface_name = k10sn09.ppd.pok.ibm.com
switch_node_number = 8
```

## Allowing users to submit interactive POE jobs

```
k10n09.ppd.pok.ibm.com: type=machine
adapter_stanzas = k10n09.ppd.pok.ibm.com k10sn09.ppd.pok.ibm.com
spacct_exclusive_enable = true
```

4. Define a machine to act as the LoadLeveler central manager. See “Quick set up” on page 61 for more information.
5. Define your scheduler to be the LoadLeveler Backfill or Gang scheduler by specifying **SCHEDULER\_TYPE = BACKFILL** or **SCHEDULER\_TYPE = GANG** in the LoadLeveler configuration file. See “Choosing a scheduler” on page 335 for more information.
6. Consider setting up a class stanza for your interactive POE jobs. See “Setting up a class for parallel jobs” on page 73 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 “Step 2: Specify user stanzas” on page 316 for more information.
7. Configure optional functions, including:
  - Setting up pools: you can organize nodes into pools by using the **pool\_list** keyword in the machine stanza. See “Step 1: Specify machine stanzas” on page 310 for more information.
  - Specifying 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.
  - 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 “Step 1: Specify machine stanzas” on page 310 for more information on these keywords.
8. Start LoadLeveler using the **llctl** command. See “Quick set up” on page 61 for more information.
9. LoadLeveler version 3.1 and version 2.2 can coexist in a mixed cluster. However, coexistence creates some restrictions on POE jobs. See “LoadLeveler 2.2 and 3.1 coexistence” on page xix for more information.

---

## Allowing users to submit PVM jobs

If users will be submitting PVM jobs, your installation must first obtain and install PVM. PVM is a public domain package distributed through electronic mail by Oak Ridge National Labs. To obtain information on PVM, issue the following:

```
echo "send index from pvm3" | mail netlib@ornl.gov
```

For RS6K architecture PVM, LoadLeveler expects to find PVM installed in **loadl/pvm3**. You can override this using the **pvm\_root** entry in the machine stanza. The value of **pvm\_root** is used to set the environment variable **\$(PVM\_ROOT)** required by PVM. For example:

```
gallifrey: type = machine
central_manager = true
schedd_host = true
alias = drwho
pvm_root = /home/userid/loadl/pvm3
```

For PVM 3.3.11+ (that is, SP2MPI architecture), LoadLeveler does not expect to find PVM installed in **loadl/pvm3**. PVM 3.3.11+ must be installed in a directory



## Allowing users to submit PVM jobs

accessible to, and executable by, all nodes in the LoadLeveler cluster. Administrators must communicate the location of this directory to their users.

Running PVM requires that each user be allowed to run only one instance of PVM per machine. In order to ensure that LoadLeveler does not attempt to start more than one PVM job per machine, you can set up a class for PVM jobs. To do this, you need to add a class stanza to your administration file and a class statement to your configuration file. The following is an example of a PVM class stanza that you can add to your administration file:

```
PVM3:  type = class
max_node = 15  # max of 15 processors per user per job
```

The following is an example of statements that you can add to your configuration file:

```
MAX_STARTERS = 2
Class = {"ClassA" "ClassA" "PVM3" }
```

This combination of the **MAX\_STARTERS** keyword and the **Class** keyword allows two jobs of Class A, or one job of Class A and one of class PVM3, to start. Limiting PVM jobs by using a class where **MAX\_STARTERS** is greater than 1 is only a policy. The user can still submit a PVM job to Class A. Note also that specifying **MAX\_STARTERS=1** would enforce a policy of one job per machine.

See “Common set up problems with parallel jobs” on page 437 for more information.

## Restrictions and limitations for PVM jobs

For PVM 3.3, dynamic allocation and de-allocation of parallel machines are not supported.

---

## 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 the above 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:

## Setting up a class for parallel jobs

- 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
- 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 “Step 3: Define LoadLeveler machine characteristics” on page 339.

---

## Setting up a parallel master node

LoadLeveler allows you to define a parallel master node—which 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 “Step 1: Specify machine stanzas” on page 310 and “Step 3: Specify class stanzas” on page 319.

---

## Chapter 9. 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.

---

### 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 machine(s) 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. In order to enable **llq -x**, you should specify the following keywords in the configuration file:

#### **ACCT = A\_ON A\_DETAIL**

Turns accounting data recording on. For more information on this keyword, see "Step 9: Define job accounting" on page 349.

#### **JOB\_ACCT\_Q\_POLICY = *number***

Where *number* is 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. The default is 300 seconds.

#### **JOB\_LIMIT\_POLICY = *number***

Where *number* is an amount of time in seconds. 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. The default for **JOB\_LIMIT\_POLICY** is **POLLING\_FREQUENCY** multiplied by **POLLS\_PER\_UPDATE**.

---

### 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 "Step 1: Specify machine stanzas" on page 310.

### 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 148. For more information on the collection of accounting records, see “llq - Query job status” on page 173.

---

### 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 “Chapter 11. Job command file keywords” on page 85.

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 which 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
- Account number specified on the Job

## Collecting job resource information based on user accounts

- 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 non-zero 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. For more information, refer to the "Step 9: Define job accounting" on page 349.
- 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:

**GLOBAL\_HISTORY = \$(SPOOL) (optional)**

---

## 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. See the **llsummary** command, "llsummary - Return job resource information for accounting" on page 202, for examples of the short and extended versions of the report.

## **Accounting reports**

For information on the accounting Application Programming Interfaces, refer to "Chapter 4. LoadLeveler API interface" on page 33.

---

## **Job accounting setup procedure**

You can find the procedure for setting up job accounting files in the appendix under "Setting up job accounting files" on page 374.

---

## Chapter 10. Routing jobs to NQS machines

Users can submit NQS scripts to LoadLeveler and have them routed to a machine outside of the LoadLeveler cluster that runs NQS. LoadLeveler supports COSMIC NQS version 2.0 and other versions of NQS that support the same commands and options and produce similar output for those commands.

The following diagram illustrates a typical environment that allows users to have their jobs routed to machines outside of LoadLeveler for processing:

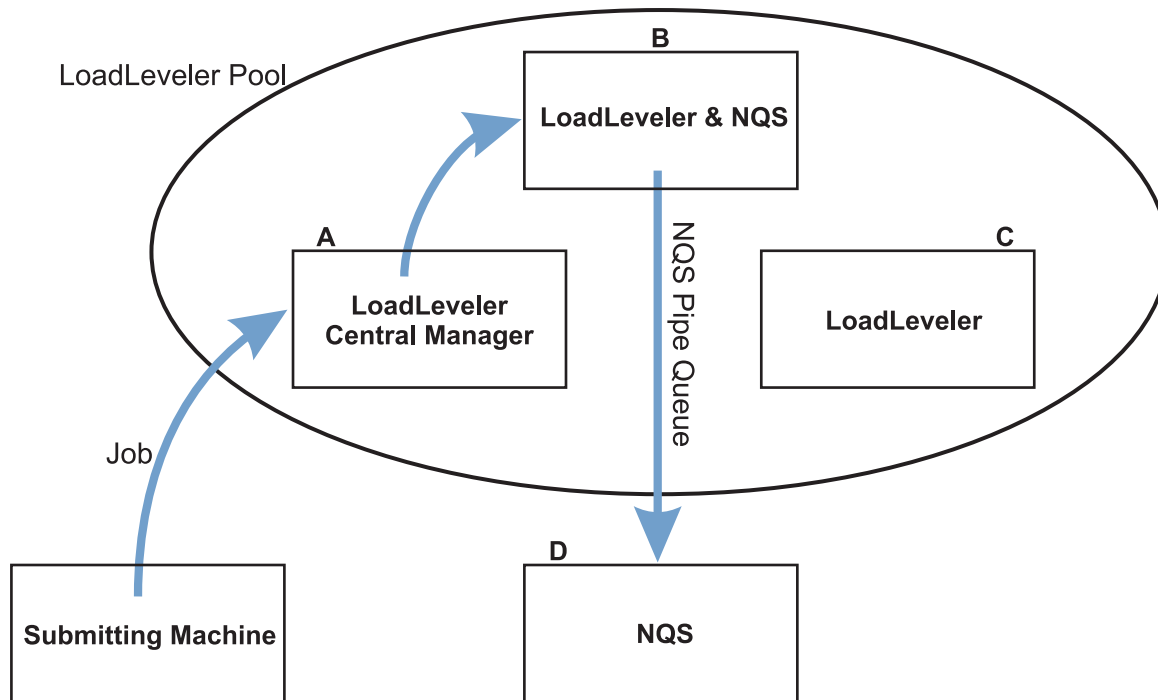


Figure 18. Environment illustrating jobs being routed to NQS machines.

As the diagram illustrates, machines A, B, and C, are members of the LoadLeveler cluster. Machine A has the central manager running on it and machine B has both LoadLeveler and NQS running on it. Machine C is a third member of the cluster. Machine D is outside of the cluster and is running NQS.

When a user submits a job to LoadLeveler, machine A, that runs the central manager, schedules the job to machine B. LoadLeveler running on machine B routes the job to machine D using NQS. Keep this diagram in mind as you continue to read this chapter.

---

### Setting up the NQS environment

Setting up the NQS environment involves the following:

- Install NQS on each node that an NQS class is defined. In the previous diagram, this is machine B.
- Create an NQS pipe queue on the LoadLeveler machine whose destination is the NQS batch queue on the machine designated to run the NQS jobs.

In the previous diagram, you would create the NQS pipe queue on machine B.

## Setting up the NQS environment

- Create an NQS batch queue on the machine designated to run the NQS jobs. In the previous diagram, this is machine D.

---

## Designating machines to which jobs will be routed

To designate a machine to which your jobs will be routed, follow these steps:

1. Set up a special class in the **LoadL\_admin** file by adding the following class definitions to the file:

**NQS\_class = true | false**

When this flag is set to **true**, any job submitted to this class will be routed to an NQS machine.

**NQS\_submit = name**

The name of the NQS pipe queue to which the job will be routed. When the job is dispatched by LoadLeveler, LoadLeveler will invoke the **qsub** command using the name of the this queue.

**NQS\_query = queue names**

A blank delimited list of queue names (including host names if necessary) to be used with the **qstat** command to monitor the job and **qdel** to cancel the job.

You can set up multiple classes to access different machines.

2. Modify the local configuration file on the machines that you want to accept this class of jobs.
3. Add the **NQS\_DIR** keyword to the **LoadL\_config** file:

**NQS\_DIR = NQS directory**

Defines the directory where NQS commands **qsub**, **qstat**, and **qdel** reside. The default is **/usr/bin**.

---

## NQS scripts

Scripts originally written for NQS that contain NQS options are acceptable to LoadLeveler. The options are mapped as closely as possible to the features provided by LoadLeveler, but the exact function is not always available. NQS options map to LoadLeveler as follows:

<b>a</b>	startdate
<b>e</b>	error
<b>ke</b>	ignored
<b>ko</b>	ignored
<b>lc</b>	core_limit
<b>ld</b>	data_limit
<b>lf</b>	file_limit
<b>lm</b>	rss_limit
<b>IM</b>	ignored
<b>ln</b>	ignored
<b>ls</b>	stack_limit
<b>lt</b>	cpu_limit
<b>IT</b>	ignored
<b>lv</b>	ignored
<b>lw</b>	ignored
<b>mb</b>	notification (always)
<b>me</b>	notification (complete)
<b>mu</b>	notify_user
<b>nr</b>	restart = no



<b>o</b>	output
<b>p</b>	user_priority
<b>q</b>	class
<b>r</b>	ignored
<b>re</b>	ignored
<b>ro</b>	ignored
<b>s</b>	shell
<b>x</b>	environment = copyall
<b>z</b>	suppresses messages but not mail

---

## NQS machine job routing procedure

You can find the procedures for setting up job routing and running jobs on NQS machines under “Routing jobs to NQS machines” on page 375.

## NQS machine job routing procedure

---

## Part 5. Detailed descriptions

### Descriptions summary

This section provides detailed LoadLeveler reference information.

Chapter 11. Job command file keywords	85
Chapter 12. Administration and Configuration file keywords	111
Administration file keywords	111
Configuration file keywords and LoadLeveler variables	115
Chapter 13. LoadLeveler daemons and job states	129
Daemons	129
Job states	134
Chapter 14. Commands	137
Chapter 15. Application Programming Interfaces (APIs)	215
Accounting API	215
Checkpointing API	218
Submit API	268
Data Access API	223
Parallel Job API	260
Workload Management API	270
Query API	265
User exits	282
Chapter 16. Procedures	293
Using the Graphical User Interface	293
Customizing the administration file	310
Customizing the global and local configuration file	333
Setting up job accounting files	374
Routing jobs to NQS machines	375



---

## Chapter 11. Job command file keywords

This section 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 multi-step job, its value is inherited by all proceeding job steps. Exceptions to this are noted in the keyword description.

---

### 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 “Chapter 9. Gathering job accounting data” on page 75.

The syntax is:

```
account_no = string
```

where *string* is a text string that can consist of a combination of numbers and letters. For 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.

The syntax is:

```
arguments = arg1 arg2 ...
```

For example, if your job requires the numbers 5, 8, 9 as input, your arguments keyword would be similar to:

```
arguments = 5 8 9
```

---

### blocking

Blocking specifies that tasks be assigned to machines in multiples of a certain integer. Unlimited blocking specifies that tasks be assigned to the 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.

The syntax is:

```
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. For example:

## Job command file keywords

```
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.

---

## 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.

The syntax is:

```
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. This is the default.

If you specify an invalid value for the **checkpoint** keyword, an error message is generated and the job is not submitted.

For 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 “Step 14: Enable checkpointing” on page 356 .

---

## ckpt\_dir

Specifies the directory which contains the checkpoint file.

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 “Step 14: Enable checkpointing” on page 356.

The syntax is:

```
ckpt_dir = pathname
```

For 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 358.

### Notes:

1. The values for **ckpt\_dir** are case sensitive.
2. The keyword **ckpt\_dir** is not allowed in the command file for interactive POE sessions.

---

## ckpt\_file

Used to specify the base name of the checkpoint file.

The syntax is:

```
ckpt_file = filename
```

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.

For 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 358.

## Job command file keywords

### Notes:

1. The value for the **ckpt\_file** keyword is case sensitive.
2. The keyword `ckpt_file` is not allowed in the command file for interactive POE sessions.

---

## 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 abort the checkpoint and allow the job to continue. If the checkpoint is not able to be aborted and the hard limit is exceeded, LoadLeveler will terminate the job.

The syntax is:

**ckpt\_time\_limit** = *hardlimit,softlimit*

Some examples of the checkpoint time limit include:

```
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 detailed information on limits and additional examples, see “Limit keywords” on page 323.

---

## class

Specifies the name of a job class defined locally in your cluster. If not specified, the default job class, **No\_Class**, is assigned. You can use the **llclass** command to find out information on job classes.

The syntax is:

**class** = *name*

For example, if you are allowed to submit jobs belonging to a class called “largejobs”, your class keyword would look like the following:

`class = largejobs`

---

## 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 is a per process limit.

The syntax is:

**core\_limit** = *hardlimit,softlimit*

Some examples are:

```
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

```

See “Limit keywords” on page 323 for more information on the values and units you can use with this keyword.

**Note:** This keyword accepts 64-bit integer values.

---

## 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 is a per process limit.

The syntax is:

**cpu\_limit** = *hardlimit,softlimit*

For example:

```

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

```

See “Limit keywords” on page 323 for more information on the values and units you can use with this keyword.

---

## 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 is a per process limit.

The syntax is:

**data\_limit** = *hardlimit,softlimit*

For example:

```

data_limit = ,125621
data_limit = 5621kb
data_limit = 2mb
data_limit = 2.5mw,2mb

```

See “Limit keywords” on page 323 for more information on the values and units you can use with this keyword.

**Note:** This keyword accepts 64-bit integer values.

---

## dependency

Specifies the dependencies between job steps. A job dependency, if used in a given job step, must be explicitly specified for that step.

The syntax is:

**dependency** = *expression*

where the syntax for the *expression* is:

## Job command file keywords

*step\_name operator*  
*value*

where *step\_name* (as described in “step\_name” on page 106) must be a previously defined job step and *operator* can be one of the following:

<b>==</b>	Equal to
<b>!=</b>	Not equal to
<b>&lt;=</b>	Less than or equal to
<b>&gt;=</b>	Greater than or equal to
<b>&lt;</b>	Less than
<b>&gt;</b>	Greater than
<b>&amp;&amp;</b>	And
<b>  </b>	Or

The *value* is usually a number which 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.

**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)
```

---

## environment

Specifies your initial environment variables when your job step starts. Separate environment specifications 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 is most useful in conjunction 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.

The syntax is:

```
environment = env1 ; env2 ; ...
```

For example:

```
environment = COPY_ALL; !env2;
```

---

## error

Specifies the name of the file to use as standard error (stderr) when your job step runs. If you do not specify this keyword, the file **/dev/null** is used.

The syntax is:

```
error = filename
```

For example:

```
error = $(jobid).$(stepid).err
```

### executable

For serial jobs, **executable** identifies the name of the program to run. The program can be a shell script or a binary. For parallel jobs, **executable** can be a shell script or the following:

- For Parallel Operating Environment (POE) jobs – specifies the full path name of the POE executable.
- For Parallel Virtual Machine (PVM) jobs – specifies the name of your parallel job.

If you do not include this keyword and the job command file is a shell script, LoadLeveler uses the script file as the executable.

The syntax is:

**executable** = *name*

Examples:

```
# @ executable = a.out
# @ executable = /usr/bin/poe (for POE jobs)
# @ executable = my_parallel_job (for PVM jobs)
```

Note that the **executable** statement automatically sets the **\$(base\_executable)** variable, which is the file name of the executable without the directory component. See Figure 43 on page 409 for an example of using the **\$(base\_executable)** variable.

---

### file\_limit

Specifies the hard limit, soft limit, or both limits for the size of a file. This is a per process limit.

The syntax is:

**file\_limit** = *hardlimit,softlimit*

For example:

```
file_limit = 100pb,50tb
```

See “Limit keywords” on page 323 for more information on the values and units you can use with this keyword.

**Note:** This keyword accepts 64-bit integer values.

---

### group

Specifies the LoadLeveler group. If not specified, this defaults to the default group, **No\_Group**. The syntax is:

**group** = *group\_name*

For 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

**system**

Specifies system hold

**usersys**

Specifies user and system hold

The syntax is:

**hold** = **user**|**system**|**usersys**

For example, to put a user hold on a job, the keyword statement would be:

**hold** = **user**To remove the hold on the job, you can use either the GUI or the **llhold -r** command.**image\_size**

Maximum virtual image size, in kilobytes, 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 do not specify the image size of your job command file, the image size is that of the executable. 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.

The syntax is:

**image\_size** = *number*

Where *number* must be a positive integer. For example, to set an image size of 11 KB, the keyword statement would be:

**image\_size** = 11**Notes:**

1. The units associated with the **image\_size** keyword are predefined as kilobytes and cannot be specified as part of the keyword value.
2. This keyword accepts 64-bit integer values.

**initialdir**

The path name of the directory to use as the initial working directory during execution of the job step. If none is specified, the initial directory is the current working directory at the time you submitted the job. File names mentioned in the command file which do not begin with a **/** 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.

The syntax is:

**initialdir** = *pathname*

For 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. If not specified, the file **/dev/null** is used.

The syntax is:

```
input = filename
```

For 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 75.

The syntax is:

```
job_cpu_limit = hardlimit,softlimit
```

For example:

```
job_cpu_limit = 12:56,12:50
```

See “Limit keywords” on page 323 for more information on the values and units you can use with this keyword.

---

### 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. You can name the job using any combination of letters, numbers, or both.

The syntax is:

```
job_name = job_name
```

For example:

```
job_name = my_first_job
```

The `job_name` only appears in the long reports of the **llq**, **llstatus**, and **llsummary** commands, and in mail related to the job.

## job\_type

Specifies the type of job step to process. Valid entries are:

**pvm3** For PVM jobs with a non-SP architecture.

**parallel**

For other parallel jobs, including PVM 3.3.11+ (SP architecture).

**serial** For serial jobs. This is the default.

Note that when you specify **job\_type=pvm3** or **job\_type=serial**, you cannot specify the following keywords: **node**, **tasks\_per\_node**, **total\_tasks**, **network.LAPI**, and **network.MPI**.

The syntax is:

```
job_type = string
```

For example:

```
job_type = pvm3
```

This keyword must be specified for each job step in a job command file.

## 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 new **node** keyword. In any new job command files you create for non-PVM jobs, you should use the **node** keyword to request nodes/processors. The **max\_processors** keyword should be used by existing job command files and new PVM job command files. Note that if you specify in a job command file both the **max\_processors** keyword and the **node** keyword, the job is not submitted.

The syntax is:

```
max_processors = number
```

For example:

```
max_processors = 6
```

## 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 new **node** keyword. In any new job command files you create for non-PVM jobs, you should use the **node** keyword to request nodes/processors. The **min\_processors** keyword should be used by existing job command files and new PVM job command files. Note that if you specify in a job command file both the **min\_processors** keyword and the **node** keyword, the job is not submitted.

## Job command file keywords

The syntax is:

```
min_processors = number
```

For 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 via 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 (or the **Adapter** preference) 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.)

The syntax is:

```
network.protocol = network_type [, [usage] [, mode [, comm_level] ] ]
```

Where:

*protocol*

Specifies the communication protocol(s) that are used with an adapter, and can be the following:

- MPI** Specifies the Message Passing Interface. You can specify in a job step both **network.MPI** and **network.LAPI**.
- LAPI** Specifies the Low-level Application Programming Interface. You can specify in a job step both **network.MPI** and **network.LAPI**.
- PVM** Specifies a Parallel Virtual Machine job. When you specify in a job step **network.PVM**, you cannot specify any other network statements in that job step. Also, the adapter *mode* must be **IP**.

*network\_type*

Specifies either an adapter name or a network type. This field is required. The possible values for adapter name are the names associated with the interface cards installed on a node (for example, en0, tk1 and css0 ). If you specify a **network\_type** you must have made sure in the admin file that the **network\_type** identifies a unique communication path. The possible values for network type are installation-defined; the LoadLeveler administrator must specify them 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 css0 adapters. When a switch adapter exists on a node, the network\_type can be specified as csss, which indicates that striped communication should be used. For more information on:

- Specifying network type, see "Step 5: Specify adapter stanzas" on page 332
- Striping, see "Submitting jobs that use striping" on page 52

*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**.



**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 SP switch 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. For more information on adapter windows, see *Parallel System Support Programs for AIX Administration Guide*.

**comm\_level**

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. For more information on device resources, consult the PSSP Admin Guide. 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.

**Example 1:** To use the MPI protocol with an SP switch adapter in User Space mode without sharing the adapter, enter the following:

```
network.MPI = css0,not_shared,US,HIGH
```

**Example 2:** To use the MPI protocol with a shared SP switch adapter in IP mode, enter the following:

```
network.MPI = css0,,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 = css0,,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.)

The syntax is:

```
node = [min] [,max]
```

## Job command file keywords

Where:

- min* Specifies the minimum number of nodes requested by the job step. The default is 1.
- max* Specifies the maximum number of nodes requested by the job step. The default is the *min* value of this keyword. 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). That is, the maximum must be less than or equal to any **max\_node** value specified in a user, group, or class stanza.

For 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
```

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
```

For information on specifying the number of tasks you want to run on a node, see “Task assignment considerations” on page 50, “tasks\_per\_node” on page 107, and “total\_tasks” on page 107.

---

## node\_usage

Specifies whether this job step shares nodes with other job steps.

The syntax is:

```
node_usage = shared | not_shared | slice_not_shared
```

Where:

### **shared**

Specifies that nodes can be shared with other tasks of other job steps. This is the default.

### **not\_shared**

Specifies that nodes are not shared: no other job steps are scheduled on this node.

### **slice\_not\_shared**

Specifies that a job step will not share nodes with other job steps when it is running in its own time-slice. When the time slice ends, other job steps can run. On nodes where the job step is running, Gang matrix columns can have rows with only this job step specified but not a row with both this job step and another job step specified.

**Note:** When SCHEDULER\_TYPE is LL\_DEFAULT or BACKFILL, slice\_not\_shared will have the same meaning as not\_shared. This allows use of the same job command file with different values of SCHEDULER\_TYPE.

## notification

Specifies when the user specified in the **notify\_user** keyword is sent mail. The syntax is:

**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. This is the default.

For example, 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 UNIX 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. The default is the submitting user and the submitting machine.

The syntax is:

**notify\_user** = *userID*

For example, if you are the job step owner but you want a co-worker whose name and user ID is **bob**, to receive mail regarding the job step, your notify keyword would be:

`notify_user = bob`

---

## output

Specifies the name of the file to use as standard output (stdout) when your job step runs. If not specified, the file **/dev/null** is used.

The syntax is:

**output** = *filename*

For example:

`output = out.${jobid}`

### parallel\_path

Specifies the path that should be used when starting a PVM 3.3 slave process. This is used for PVM 3.3 only and is translated into the **ep** keyword as defined in the PVM 3.3 **hosts** file.

For example:

```
parallel_path = /home/userid/cmds/pvm3/$PVM_ARCH:$PVM_ROOT/lib/$PVM_ARCH
```

The **parallel\_path** statement above has two components, separated by a colon. The first component points to the location of the user's programs. The second component points to the location of the **pvmgs** routine – required if the job uses PVM 3.3 group support – assuming PVM 3.3 is installed “normally”. Note that your installation must install PVM 3.3 to include group support in order for you to use group support within LoadLeveler. \$PVM\_ARCH will be replaced by the architecture of the machine, as defined by PVM 3.3. This will specify the path to be searched for executables when the user's job issues a **pvm\_spawn()** command.

\$PVM\_ARCH, and \$PVM\_ROOT are PVM environment variables. For more information, see the appropriate PVM 3.3 documentation.

---

### 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 which 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. See “requirements” on page 101 for more information.

**Note:** Preferences are ignored when using Gang scheduling.

The syntax is:

```
preferences = Boolean_expression
```

Some examples are:

```
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.

The syntax is:

```
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.

The syntax is:

**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.

The requirements supported are:

### Adapter

Specifies the pre-defined 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. The **Adapter** requirement is provided for compatibility with Version 1.3 job command files. Note that you cannot specify both the **Adapter** requirement and the **network** statement in a job command file.

The pre-defined network types are:

#### **hps\_ip**

Refers to an SP switch in IP mode.

#### **hps\_user**

Refers to an SP switch in user space mode. If the switch in user mode is requested by the job, no other jobs using the switch in user mode will be allowed on nodes running that job.

#### **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 above 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 UNIX platform for which your executable has been compiled. The default is the architecture of the submitting machine.

### 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

## Job command file keywords

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.

### LL\_Version

Specifies the LoadLeveler version, in dotted decimal format, on which you want your job step to run. For example, LoadLeveler Version 2 Release 1 (with no modification levels) is written as 2.1.0.0.

### 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.

**Note:** If you have a mixed LoadLeveler cluster where the OpSys values of the machines may be either AIX43 or AIX51, 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 AIX51, the **llsubmit** command automatically adds (OpSys == "AIX51") to the other job requirements unless an OpSys requirement has already been explicitly specified. This means that the specification:

```
requirements = (Machine == "jupiter")
```

automatically becomes:

```
requirements = (Machine == "jupiter") && (OpSys == "AIX51")
```

This requirement can not be satisfied unless the OpSys value of "jupiter" is also AIX51. In this case, a better strategy would be to use an expression such as:

```
requirements = (Machine == "jupiter") && ((OpSys == "AIX43") || (OpSys == "AIX51"))
```

### Memory

Specifies the amount of physical memory required in megabytes 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 UNIX platform for which your executable has been compiled. The default is the operating system of the submitting machine. The executable must be compiled on a machine that matches these requirements.

### Pool

Specifies the number of a pool where you want your job step to run.

**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 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.

For 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 2 Release 1 or any follow-on release, enter:

```
requirements = (LL_Version >= "2.1")
```

Note that the statement **requirements = (LL\_Version == "2.1")** matches only the value 2.1.0.0.

---

## resources

Specifies quantities of the consumable resources "consumed" by each task of a job step. The resources may be machine resources or floating resources. The syntax is:

```
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, with three exceptions:

**ConsumableCpus**, and **ConsumableMemory** must be specified with a value which is greater than zero, and **ConsumableVirtualMemory** must be specified with a value greater than 0, and greater than or equal to the **image\_size**. If the count is not valid, then LoadLeveler will issue an error message, and will not submit the job. 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)
```

## Job command file keywords

pb petabytes (2\*\*50 bytes)  
pw petawords (2\*\*52 bytes)  
eb exabytes (2\*\*60 bytes)  
ew exawords (2\*\*62 bytes)

**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. 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.

When resource usage and resource submission is enforced, the **resources** keyword must specify requirements for the resources defined in the **ENFORCE\_RESOURCE\_USAGE** keyword.

**Note:** The **resources** keyword accepts 64-bit integer values. However, these values are assigned to the consumable resources defined in the **resources** keyword and not to the keyword itself.

---

## restart

Specifies whether LoadLeveler considers a job “restartable.” The syntax is:

**restart** = yes | no

If **restart=yes**, which is the default, 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”.

---

## restart\_from\_ckpt

Indicates whether a job step is to be restarted from a checkpoint file.

The syntax is:

**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 358 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. This is the default value for this keyword.



**Note:** 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 is when the job step is terminated and then returned to the LoadLeveler job queue).

The syntax is:

**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. This is the default

---

## **rss\_limit**

Specifies the hard limit, soft limit, or both limits for the resident set size.

The syntax is:

**rss\_limit** = *hardlimit,softlimit*

For example:

**rss\_limit**=12mb,10mb

The above example specifies the limits in megabytes, but If no units are specified, then bytes are assumed. See “Limit keywords” on page 323 for more information on the values and units you can use with this keyword.

**Note:** This keyword accepts 64-bit integer values.

---

## **shell**

Specifies the name of the shell to use for the job step. If not specified, the shell used in the owner’s password file entry is used. If none is specified, the /bin/sh is used.

The syntax is:

**shell** = *name*

For example, if you wanted to use the Korn shell, the shell keyword would be:

**shell** = /bin/ksh

---

## **stack\_limit**

Specifies the hard limit, soft limit, or both limits for the size of the stack that is created.

The syntax is:

**stack\_limit** = *hardlimit,softlimit*

## Job command file keywords

For example:

```
stack_limit = 120000,100000
```

Because no units have been specified in the above example, LoadLeveler assumes that the figure represents a number of bytes. See “Limit keywords” on page 323 for more information on the values and units you can use with this keyword.

**Note:** This keyword accepts 64-bit integer values.

---

## startdate

Specifies when you want to run the job step. If not specified, the current date and time are used.

The syntax is:

```
startdate = date time
```

*date* is expressed as *MM/DD/YYYY*, and *time* is expressed as *HH:mm(:ss)*.

For example, if you want the job to run on August 28th, 2000 at 1:30 PM, issue:

```
startdate = 08/28/2000 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. 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.

The syntax is:

```
step_name = step_name
```

For 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. The syntax is:

```
task_geometry={(task id,task id,...)(task id,task id, ...) ... }
```

In this example, a job with 6 tasks will run on 4 different nodes:

```
task_geometry={{(0,1) (3) (5,4) (2)}}
```

Each number in the example above 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: (a) the step is serial, (b) **job\_type** is anything other than "parallel", or (c) 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 50.

---

### tasks\_per\_node

Specifies the number of tasks of a parallel job you want to run per node. Use this keyword in conjunction 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.

The syntax is:

```
tasks_per_node = number
```

Where *number* is the number of tasks you want to run per node. The default is one task per node.

For 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
```

The above 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 in conjunction 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.

## Job command file keywords

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.

The syntax is:

**total\_tasks** = *number*

Where *number* is the total number of tasks you want to run.

For 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
```

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.

---

## 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.

The syntax is:

**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. The default priority is 50. Note that this 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.

---

## 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 either the Backfill or Gang 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 335.

The syntax is:

```
wall_clock_limit = hardlimit,softlimit
```

An example is:

```
wall_clock_limit = 5:00,4:30
```

See “Limit keywords” on page 323 for more information on the values and units you can use with this keyword.

---

## 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:

### **\$(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.

### **\$(domain)**

The domain of the host from which the job was submitted.

### **\$(schedd\_host)**

The hostname of the scheduling machine.

### **\$(schedd\_hostname)**

The hostname and domain name of the scheduling machine.

### **\$(jobid)**

The sequential number assigned to this job by the schedd daemon. The **\$(jobid)** variable and the **\$(cluster)** variable are equivalent.

### **\$(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.

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 “Chapter 11. Job command file keywords” on page 85.

- **\$(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 43 on page 409 for an example of using the **\$(base\_executable)** variable.

## Job command file variables

### Example 1

The following job command file creates an output file called **stance.78.out**, where stance is the host and 78 is the jobid.

```
# @ executable = my_job
# @ arguments  = 5
# @ output     = $(host).$(jobid).out
# @ queue
```

### Example 2

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
```

**Note:** For detailed information and examples on this topic, see “Additional examples of building job command files” on page 407.

## Chapter 12. Administration and Configuration file keywords

### Administration file keywords

The following table contains a brief description of the keywords you can use in the administration file. For more information on a specific keyword, see the section and page number referenced in the “For Details” column.

Table 13. Administration file keywords

Admin. File Keyword	Stanzas	Brief Description	For Details See:
account	User, Group	A list of account numbers available to a user submitting jobs.	“Step 2: Specify user stanzas” on page 316  “Step 4: Specify group stanzas” on page 329
adapter_name	Adapter	Specifies the name the operating system uses to refer to an interface card installed on a node (such as en0).	“Step 5: Specify adapter stanzas” on page 332
adapter_stanzas	Machine	A list of adapter stanza names that define the adapters on a machine which can be requested.	“Step 1: Specify machine stanzas” on page 310
admin	Class, Group	A list of administrators for a group or class.	“Step 3: Specify class stanzas” on page 319  “Step 4: Specify group stanzas” on page 329
alias	Machine	Lists one or more alias names to associate with the machine name.	“Step 1: Specify machine stanzas” on page 310
central_manager	Machine	When <b>true</b> , this designates the machine as the LoadLeveler central manager.	“Step 1: Specify machine stanzas” on page 310
ckpt_dir	Class	Specifies the directory to be used for checkpoint files for jobs that did not specify this directory in the job command file.	“Step 3: Specify class stanzas” on page 319
ckpt_time_limit	Class	Specifies the hard limit, soft limit, or both limits for the elapsed time that checkpointing a job can take.	“Step 3: Specify class stanzas” on page 319  “Limit keywords” on page 323
class_comment	Class	Text characterizing the class	“Step 3: Specify class stanzas” on page 319
core_limit	Class	Specifies the hard limit, soft limit, or both limits for the size of a core file a job can create.	“Step 3: Specify class stanzas” on page 319  “Limit keywords” on page 323
cpu_limit	Class	Specifies the hard limit, soft limit, or both limits for the CPU time a job can use.	“Step 3: Specify class stanzas” on page 319  “Limit keywords” on page 323
cpu_speed_scale	Machine	Determines whether CPU time is normalized according to machine speed.	“Step 1: Specify machine stanzas” on page 310

## Administration file keywords

Table 13. Administration file keywords (continued)

Admin. File Keyword	Stanzas	Brief Description	For Details See:
data_limit	Class	Specifies the hard limit, soft limit, or both limits for the size of a data segment a job can use.	"Step 3: Specify class stanzas" on page 319 "Limit keywords" on page 323
default_class	User	A class name that is the default value assigned to jobs submitted by users for which no class statement appears.	"Step 2: Specify user stanzas" on page 316
default_group	User	A group name to which the user belongs.	"Step 2: Specify user stanzas" on page 316
default_interactive_class	User	A class to which interactive jobs are assigned for jobs submitted by users who do not specify a class using <code>LOADL_INTERACTIVE_CLASS</code> .	"Step 2: Specify user stanzas" on page 316
default_resources	Class	Specifies the default amount of resources consumed by a task of a job step, provided that no <b>resources</b> keyword is coded for the step in the job command file.	"Step 3: Specify class stanzas" on page 319
exclude_groups	Class	A list of groups names identifying those who cannot submit jobs of a particular class.	"Step 3: Specify class stanzas" on page 319
exclude_users	Class, Group	A list of user names identifying those who cannot submit jobs of a particular class or who are not members of the group.	"Step 3: Specify class stanzas" on page 319 "Step 4: Specify group stanzas" on page 329
execution_factor	Class	Specifies the relative amount of processing time a specified job class receives. Time is measured relative to other jobs that are time-sharing on the same nodes.	"Step 3: Specify class stanzas" on page 319
file_limit	Class	Specifies the hard limit, soft limit, or both limits for the size of a file that a job can create.	"Step 3: Specify class stanzas" on page 319 "Limit keywords" on page 323
include_groups	Class	A list of groups names identifying those who can submit jobs of a particular class.	"Step 3: Specify class stanzas" on page 319
include_users	Class, Group	A list of user names identifying those who can submit jobs of a particular class or who do belong to the group.	"Step 3: Specify class stanzas" on page 319 "Step 4: Specify group stanzas" on page 329
interface_address	Adapter	Specifies the IP address by which the adapter is known to other nodes in the network.	"Step 5: Specify adapter stanzas" on page 332
interface_name	Adapter	Specifies the name by which the adapter is known to other nodes in the network.	"Step 5: Specify adapter stanzas" on page 332



Table 13. Administration file keywords (continued)

Admin. File Keyword	Stanzas	Brief Description	For Details See:
job_cpu_limit	Class	Specifies the hard limit, soft limit, or both limits for the amount of CPU time an individual job step can use per processor.	“Step 3: Specify class stanzas” on page 319 “Limit keywords” on page 323
machine_mode	Machine	Specifies the type of jobs this machine can run (batch, interactive, or both).	“Step 1: Specify machine stanzas” on page 310
master_node_exclusive	Machine	When <b>true</b> , this machine is used only as a master node for parallel jobs.	“Step 1: Specify machine stanzas” on page 310
master_node_requirement	Class	When <b>true</b> , jobs in this class have the requirement that they run on a master node having the <b>master_node_exclusive</b> setting.	“Step 3: Specify class stanzas” on page 319
max_adapter_windows	Machine	Specifies how many of a machine’s available adapter windows LoadLeveler can use.	“Step 1: Specify machine stanzas” on page 310
maxidle	User, Group	Maximum number of idle job steps this user or group can have simultaneously.	“Step 2: Specify user stanzas” on page 316 “Step 4: Specify group stanzas” on page 329
maxjobs	User, Class, Group	Maximum number of job steps this user, class, or group can have running simultaneously.	“Step 2: Specify user stanzas” on page 316 “Step 3: Specify class stanzas” on page 319 “Step 4: Specify group stanzas” on page 329
max_jobs_scheduled	Machine	The maximum number of job steps that this machine can run.	“Step 1: Specify machine stanzas” on page 310
max_node	User, Class, Group	The maximum number of nodes a user can request for a parallel job.	“Step 2: Specify user stanzas” on page 316 “Step 3: Specify class stanzas” on page 319 “Step 4: Specify group stanzas” on page 329
max_processors	User, Class, Group	The maximum number of machines a user can request for a parallel job.	“Step 2: Specify user stanzas” on page 316 “Step 3: Specify class stanzas” on page 319 “Step 4: Specify group stanzas” on page 329
maxqueued	User, Group	The maximum number of job steps a single group or user can have queued at the same time.	“Step 2: Specify user stanzas” on page 316 “Step 4: Specify group stanzas” on page 329

## Administration file keywords

Table 13. Administration file keywords (continued)

Admin. File Keyword	Stanzas	Brief Description	For Details See:
max_smp_tasks	Machine	Specifies the maximum number of tasks that may run concurrently on a specified machine.	"Step 1: Specify machine stanzas" on page 310
max_total_tasks	User, Class, Group	Defines the maximum number of tasks allowed to run by the scheduler for jobs of a specified class, user, or group.	"Step 2: Specify user stanzas" on page 316 "Step 3: Specify class stanzas" on page 319 "Step 4: Specify group stanzas" on page 329
multilink_address	Adapter	Specifies the multilink address used for IP striping on the associated adapter.	"Submitting jobs that use striping" on page 52 "Customizing the administration file" on page 310
multilink_list	Adapter	Specifies the IP addresses of the adapters that this multilink device stripes across.	"Submitting jobs that use striping" on page 52 "Customizing the administration file" on page 310
name_server	Machine	A list of nameservers used for a machine.	"Step 1: Specify machine stanzas" on page 310
network_type	Adapter	The type of network the adapter supports (for example, Ethernet). This is an administrator defined name.	"Step 5: Specify adapter stanzas" on page 332
nice	Class	Increments the <i>nice</i> value of a job.	"Step 3: Specify class stanzas" on page 319
NQS_class	Class	When <b>true</b> , any job submitted to this class is routed to an NQS machine.	"Step 3: Specify class stanzas" on page 319
NQS_query	Class	A list of queue names to use to monitor and cancel jobs.	"Step 3: Specify class stanzas" on page 319
NQS_submit	Class	A name that identifies the name of the NQS pipe queue to which the job will be routed.	"Step 3: Specify class stanzas" on page 319
pool_list	Machine	Specifies a list of pool numbers to which the machine belongs. Do not use negative numbers in a machine pool_list.	"Step 1: Specify machine stanzas" on page 310
priority	User, Class, Group	A number that identifies the priority of the appropriate user, class, or group.	"Step 2: Specify user stanzas" on page 316 "Step 3: Specify class stanzas" on page 319 "Step 4: Specify group stanzas" on page 329
pvm_root	Machine	A directory in which PVM 3.3 is installed.	"Step 1: Specify machine stanzas" on page 310

Table 13. Administration file keywords (continued)

Admin. File Keyword	Stanzas	Brief Description	For Details See:
resources	Machine	Specifies quantities of the consumable resources initially available on the machine.	"Step 1: Specify machine stanzas" on page 310
rss_limit	Class	Specifies the hard limit, soft limit, or both limits for the resident set size for a job.	"Step 3: Specify class stanzas" on page 319 "Limit keywords" on page 323
schedd_fenced	Machine	When <b>true</b> , the central manager ignores connections from this schedd machine.	"Step 1: Specify machine stanzas" on page 310
schedd_host	Machine	When <b>true</b> , this machine is used to help submit-only machines access LoadLeveler hosts that run LoadLeveler jobs.	"Step 1: Specify machine stanzas" on page 310
spacct_exclude_enable	Machine	Specifies whether the SP accounting function is informed whenever this machine is being used exclusively by a particular job.	"Step 1: Specify machine stanzas" on page 310
speed	Machine	The weight associated with the machine for scheduling purposes.	"Step 1: Specify machine stanzas" on page 310
stack_limit	Class	Specifies the hard limit, soft limit, or both limits for the size of a stack.	"Step 3: Specify class stanzas" on page 319 "Limit keywords" on page 323
submit_only	Machine	When <b>true</b> , designates this as a submit-only machine.	"Step 1: Specify machine stanzas" on page 310
switch_node_number	Adapter	The node on which the SP switch adapter is installed.	"Step 5: Specify adapter stanzas" on page 332
total_tasks	User, Class, Group	The maximum number of tasks a user can request for a parallel job.	"Step 2: Specify user stanzas" on page 316 "Step 3: Specify class stanzas" on page 319 "Step 4: Specify group stanzas" on page 329
type	All	The type of stanza.	"Administering LoadLeveler" on page 62
wall_clock_limit	Class	Specifies the hard limit, soft limit, or both limits for the amount of elapsed time for which a job can run.	"Step 3: Specify class stanzas" on page 319 "Limit keywords" on page 323

## Configuration file keywords and LoadLeveler variables

The following tables contain a brief description of the keywords you can use in the configuration file. The term *configuration file keywords* refers to keywords, user-defined variables, and LoadLeveler variables. A summary table is provided for each of the three types of configuration file keywords.

## Configuration file keywords

### Keywords

The following table serves only as a reference. For more information on a specific keyword, see the section and page number referenced in the “For Details” column.

Table 14. Configuration file keywords

Configuration file keyword	Brief description	For details
ACCT	Turns the accounting function on or off.	“Step 9: Define job accounting” on page 349
ACCT_VALIDATION	The module called to perform account validation.	“Step 9: Define job accounting” on page 349
ACTION_ON_MAX_REJECT	Specifies whether a job is canceled or put in User Hold or System Hold status when the job exceeds the <b>MAX_JOB_REJECT</b> value.	“Step 17: Specify additional configuration file keywords” on page 370
ACTION_ON_SWITCH_TABLE_ERROR	Points to an administrator supplied program that will be run when <b>DRAIN_ON_SWITCH_TABLE_ERROR</b> is set to true and a switch table unload error occurs.	“Step 17: Specify additional configuration file keywords” on page 370
ADMIN_FILE	Points to the administration file containing user, class, and machine list stanzas.	“Step 11: Specify where files and directories are located” on page 351
AFS_GETNEWTOKEN	A filter which can be used to renew an AFS token.	“Step 17: Specify additional configuration file keywords” on page 370
ARCH	The standard architecture of the system.	“Step 3: Define LoadLeveler machine characteristics” on page 339
BIN	The directory where LoadLeveler binaries are kept.	“Step 11: Specify where files and directories are located” on page 351
CENTRAL_MANAGER_HEARTBEAT_INTERVAL	The amount of time in seconds that defines how frequently primary and alternate central manager communicate with each other.	“Step 10: Specify alternate central managers” on page 350
CENTRAL_MANAGER_TIMEOUT	The number of heartbeat intervals that an alternate central manager will wait before declaring that the primary central manager is not operating.	“Step 10: Specify alternate central managers” on page 350
CKPT_CLEANUP_INTERVAL	The interval, in seconds, at which <b>CKPT_CLEANUP_PROGRAM</b> will be run.	“Remove old checkpoint files” on page 364
CKPT_CLEANUP_PROGRAM	An administrator provided program designed to delete old checkpoint files.	“Remove old checkpoint files” on page 364

Table 14. Configuration file keywords (continued)

Configuration file keyword	Brief description	For details
CLASS	The class of jobs that can run on the machine.	"Step 3: Define LoadLeveler machine characteristics" on page 339
CLIENT_TIMEOUT	The maximum time, in seconds, that a daemon waits for a response over TCP/IP from a process .	"Step 13: Define network characteristics" on page 355
COLLECTOR_DGRAM_PORT	The port number used when connecting to a daemon.	"Step 13: Define network characteristics" on page 355
CONTINUE	Continue expression. Determines if a job should continue.	"Step 8: Manage a job's status using control expressions" on page 347
CUSTOM_METRIC	A machine's relative priority to run jobs. Negative values are not allowed.	"Step 2: Define LoadLeveler cluster characteristics" on page 334
CUSTOM_METRIC_COMMAND	An executable whose exit code value is assigned to <b>CUSTOM_METRIC</b> .	"Step 2: Define LoadLeveler cluster characteristics" on page 334
DCE_ADMIN_GROUP	Specifies the DCE group containing the DCE ids of those users who will have administrator authority for the current cluster.	"Step 16: Configuring LoadLeveler to use DCE security services" on page 365
DCE_AUTHENTICATION_PAIR	A pair of installation supplied programs that are used to authenticate DCE security credentials.	"Step 17: Specify additional configuration file keywords" on page 370
DCE_ENABLEMENT	Activates the exploitation of DCE security.	"Step 16: Configuring LoadLeveler to use DCE security services" on page 365
DCE_SERVICES_GROUP	Specifies the DCE group containing all of the principal names of the LoadLeveler daemons that are authorized to run in the current cluster.	"Step 16: Configuring LoadLeveler to use DCE security services" on page 365
DRAIN_ON_SWITCH_TABLE_ERROR	Specifies that the <b>startd</b> should be drained when the switch table fails to unload.	"Step 17: Specify additional configuration file keywords" on page 370
ENFORCE_RESOURCE_SUBMISSION	Indicates whether jobs submitted should be checked for the <b>resources</b> keyword.	"Step 4: Define consumable resources" on page 341

## Configuration file keywords

Table 14. Configuration file keywords (continued)

Configuration file keyword	Brief description	For details
ENFORCE_RESOURCE_USAGE	Specifies that the resource usage for jobs needs to be enforced using the AIX Workload Manager.	"Step 4: Define consumable resources" on page 341
EXECUTE	The local directory to store the executable checkpoints of jobs submitted by other machines.	"Step 11: Specify where files and directories are located" on page 351
FEATURE	A string specifying unique characteristics of a machine.	"Step 3: Define LoadLeveler machine characteristics" on page 339
FLOATING_RESOURCES	Specifies which consumable resources are available collectively on all of the machines in the LoadLeveler cluster.	"Step 4: Define consumable resources" on page 341
FS_INTERVAL	Defines the number of seconds used as the interval for checking free file system space.	"Setting up file system monitoring" on page 337
FS_NOTIFY	<p>Specifies the amount of free file system space (in blocks) at which mail is sent to the administrator.</p> <ul style="list-style-type: none"><li>• If the amount of free space is less than the lower threshold value, then the notification indicates that there is a problem.</li><li>• When the amount of free space becomes greater than the upper threshold value, the notification indicates that the problem has been resolved.</li></ul>	"Setting up file system monitoring" on page 337
FS_SUSPEND	<p>Specifies the amount of free file system space (in blocks) at which LoadLeveler is either drained or resumed.</p> <ul style="list-style-type: none"><li>• If the amount of free space is less than the lower threshold value, then LoadLeveler is drained on the node.</li><li>• When the amount of free space becomes greater than the upper threshold value, the problem is considered resolved and LoadLeveler is resumed on the node.</li></ul>	"Setting up file system monitoring" on page 337
FS_TERMINATE	<p>Specifies the amount of free file system space (in blocks) at which LoadLeveler is terminated.</p> <ul style="list-style-type: none"><li>• If the amount of free space is less than the lower threshold value, then LoadLeveler is terminated on the node.</li><li>• An upper threshold value is required for this keyword. However, since LoadLeveler has been terminated at the lower threshold, no action occurs.</li></ul>	"Setting up file system monitoring" on page 337

Table 14. Configuration file keywords (continued)

Configuration file keyword	Brief description	For details
GANG_MATRIX_BROADCAST_CYCLE	Specifies the time in seconds that the scheduler will wait before sending the complete matrix information to the startd daemons.	"Chapter 17. Using Gang scheduling" on page 381
GANG_MATRIX_NODE_SUBSET_SIZE	Specifies the ideal number of nodes for subsets in the Gang matrix.	"Chapter 17. Using Gang scheduling" on page 381
GANG_MATRIX_REORG_CYCLE	Specifies the maximum number of negotiation cycles that the scheduler will wait to reorganize the Gang matrix into subsets.	"Chapter 17. Using Gang scheduling" on page 381
GANG_MATRIX_TIME_SLICE	Specifies the time (in seconds) that each row in the gang matrix will use to run jobs.	"Chapter 17. Using Gang scheduling" on page 381
GLOBAL_HISTORY	The directory containing the global history files.	"Step 9: Define job accounting" on page 349
GSMONITOR	Location of the gsmonitor executable (LoadL_GSmonitor).	"The gsmonitor daemon" on page 133
GSMONITOR_RUNS_HERE	When true, specifies that you want to start the gsmonitor daemon (you must have PSSP Groups Service).	"The gsmonitor daemon" on page 133
HIERARCHICAL_FANOUT	The number of children a hierarchical message is distributed to at each level of the hierarchy.	"Chapter 17. Using Gang scheduling" on page 381
HISTORY	The pathname of the history file for local LoadLeveler jobs.	"Step 11: Specify where files and directories are located" on page 351
HISTORY_PERMISSION	Specifies the permissions associated with the history file of a LoadL_schedd daemon.	"Chapter 9. Gathering job accounting data" on page 75
JOB_ACCT_Q_POLICY	The amount of time in seconds that determines how often the startd daemon updates the schedd daemon with accounting data of running jobs.	"Chapter 9. Gathering job accounting data" on page 75
JOB_EPILOG	Pathname of the epilog program.	"Writing prolog and epilog programs" on page 286
JOB_LIMIT_POLICY	The amount of time in seconds that LoadLeveler checks to see if <b>job_cpu_limit</b> has been exceeded.	"Chapter 9. Gathering job accounting data" on page 75
JOB_PROLOG	Pathname of the prolog program.	"Writing prolog and epilog programs" on page 286
JOB_USER_EPILOG	Pathname of the user epilog program.	"Writing prolog and epilog programs" on page 286
JOB_USER_PROLOG	Pathname of the user prolog program.	"Writing prolog and epilog programs" on page 286

## Configuration file keywords

Table 14. Configuration file keywords (continued)

Configuration file keyword	Brief description	For details
KBDD	Location of kbdd executable (LoadL_Kbdd).	"LoadLeveler daemons" on page 8
KILL	Kill expression. Determines if vacated jobs should be sent the SIGKILL signal.	"Step 8: Manage a job's status using control expressions" on page 347
LIB	The directory where LoadLeveler libraries are kept.	"Step 11: Specify where files and directories are located" on page 351
LOADL_ADMIN	List of LoadLeveler administrators.	"Step 1: Define LoadLeveler administrators" on page 333
LOCAL_CONFIG	Pathname of the optional local configuration file containing information specific to a node in the LoadLeveler network.	"Step 11: Specify where files and directories are located" on page 351
LOG	Local directory for storing log files.	"Step 11: Specify where files and directories are located" on page 351
MACHINE_AUTHENTICATE	Specifies whether machine validation is performed.	"Step 2: Define LoadLeveler cluster characteristics" on page 334
MACHINE_UPDATE_INTERVAL	The time, in seconds, during which machines must report to the central manager.	"Step 17: Specify additional configuration file keywords" on page 370
MACHPRIO	Machine priority expression	"Step 7: Prioritize the order of executing machines maintained by the negotiator" on page 345
MAIL	Name of a local mail program used to override default mail notification.	"Using your own mail program" on page 286
MASTER	Location of the master executable (LoadL_master).	"LoadLeveler daemons" on page 8
MASTER_DGRAM_PORT	The port number used when connecting to the daemon.	"Step 13: Define network characteristics" on page 355
MASTER_STREAM_PORT	The port number to used when connecting to the daemon.	"Step 13: Define network characteristics" on page 355
MAX_CKPT_INTERVAL	The maximum number of seconds between checkpoints for running jobs.	"Step 14: Enable checkpointing" on page 356



Table 14. Configuration file keywords (continued)

Configuration file keyword	Brief description	For details
MAX_JOB_REJECT	The number of times a job is rejected before it is canceled or put in User Hold or System Hold status.	"Step 17: Specify additional configuration file keywords" on page 370
MAX_STARTERS	The maximum number of jobs that can run simultaneously.	"Step 5: Specify how many jobs a machine can run" on page 342
MIN_CKPT_INTERVAL	The minimum number of seconds between checkpoints for running jobs.	"Step 14: Enable checkpointing" on page 356
NEGOTIATOR	Location of the negotiator executable (LoadL_negotiator).	"LoadLeveler daemons" on page 8
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.	"Step 17: Specify additional configuration file keywords" on page 370
NEGOTIATOR_CYCLE_DELAY	The 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, etc. Delaying the scheduling of jobs might improve the overall performance of the negotiator by preventing it from spending excessive time attempting to schedule jobs.	"Step 17: Specify additional configuration file keywords" on page 370
NEGOTIATOR_LOADAVG_INCREMENT	The factor added to the startd machine's load average to compensate for the increased load caused by starting another machine.	"Step 17: Specify additional configuration file keywords" on page 370
NEGOTIATOR_PARALLEL_DEFER	The length of time that a job is given to accumulate processors.	"Step 17: Specify additional configuration file keywords" on page 370
NEGOTIATOR_PARALLEL_HOLD	The length of time a job attempts to collect machines before releasing them.	"Step 17: Specify additional configuration file keywords" on page 370
NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL	The amount of time in seconds between calculation of the <b>SYSPRIO</b> values for waiting jobs.	"Step 17: Specify additional configuration file keywords" on page 370

## Configuration file keywords

Table 14. Configuration file keywords (continued)

Configuration file keyword	Brief description	For details
NEGOTIATOR_REJECT_DEFER	The amount of time in seconds the negotiator waits before it considers scheduling a job to a machine that recently rejected the job.	"Step 17: Specify additional configuration file keywords" on page 370
NEGOTIATOR_REMOVE_COMPLETED	The amount of time the negotiator keeps information on completed and removed jobs.	"Step 17: Specify additional configuration file keywords" on page 370
NEGOTIATOR_RESCAN_QUEUE	The amount of time the negotiator waits to rescan the job queue for machines that temporarily have non-runnable jobs.	"Step 17: Specify additional configuration file keywords" on page 370
NEGOTIATOR_STREAM_PORT	The port number used when connecting to the daemon.	"Step 13: Define network characteristics" on page 355
NQS_DIR	The directory where NQS commands reside.	"Step 11: Specify where files and directories are located" on page 351
OBITUARY_LOG_LENGTH	The number of lines from the need of the file that are appended to the Master_Log.	"Step 17: Specify additional configuration file keywords" on page 370
POLLING_FREQUENCY	The frequency in seconds the startd daemon uses to evaluate the load on the local machine and to decide whether to suspend, resume, or abort jobs.	"Step 17: Specify additional configuration file keywords" on page 370
POLLS_PER_UPDATE	The frequency, in POLLING_FREQUENCY intervals, with which the startd daemon updates the central manager.	"Step 17: Specify additional configuration file keywords" on page 370
PREEMPT_CLASS [class]	Specifies the preemption rule for a job class.	"Chapter 17. Using Gang scheduling" on page 381
PROCESS_TRACKING	When <b>true</b> ensures that when a job is terminated, no processes created by the job will continue running.	"Step 15: Specify process tracking" on page 364
PROCESS_TRACKING_EXTENSION	The directory containing the kernel extension binary <b>LoadL_pt_ke</b> .	"Step 15: Specify process tracking" on page 364
PUBLISH_OBITUARIES	When <b>true</b> , specifies that the master daemon sends mail to the administrator(s) when any daemon it manages dies abnormally.	"Step 17: Specify additional configuration file keywords" on page 370

Table 14. Configuration file keywords (continued)

Configuration file keyword	Brief description	For details
RELEASEDIR	The directory where all the LoadLeveler software resides.	“Step 11: Specify where files and directories are located” on page 351
RESOURCES	Specifies quantities of the consumable resources “consumed” by each task of a job step.	“Step 4: Define consumable resources” on page 341
RESTARTS_PER_HOUR	The number of times the master daemon attempts to restart a daemon that dies abnormally.	“Step 17: Specify additional configuration file keywords” on page 370
SCHEDD	Location of the schedd executable (LoadL_schedd).	“LoadLeveler daemons” on page 8
SCHEDD_INTERVAL	Specifies the interval, in seconds, at which the schedd daemon checks the local job queue.	“Step 17: Specify additional configuration file keywords” on page 370
SCHEDD_RUNS_HERE	Specifies whether this daemon will run on the host.	“Step 3: Define LoadLeveler machine characteristics” on page 339
SCHEDD_SUBMIT_AFFINITY	Specifies whether the <b>llsubmit</b> command submits a job to the machine where the command was invoked provided the schedd daemon is running on the machine.	“Step 3: Define LoadLeveler machine characteristics” on page 339
SCHEDD_STREAM_PORT	The port number used when connecting to the daemon.	“Step 13: Define network characteristics” on page 355
SCHEDULE_BY_RESOURCES	Specifies which consumable resources are considered by the LoadLeveler schedulers.	“Step 4: Define consumable resources” on page 341
SCHEDULER_API	When <b>YES</b> , disables the native LoadLeveler scheduling algorithm. <b>Note:</b> This keyword is obsolete and should be replaced by SCHEDULER_TYPE = API.	“Step 2: Define LoadLeveler cluster characteristics” on page 334
SCHEDULER_TYPE	Specifies the LoadLeveler scheduling algorithm: <ul style="list-style-type: none"> <li>• LL_DEFAULT</li> <li>• BACKFILL</li> <li>• API</li> <li>• GANG</li> </ul>	“Step 2: Define LoadLeveler cluster characteristics” on page 334  Additional information on Gang scheduling can be found in “Chapter 17. Using Gang scheduling” on page 381.

## Configuration file keywords

Table 14. Configuration file keywords (continued)

Configuration file keyword	Brief description	For details
SPOOL	The local directory where LoadLeveler keeps the local job queue and checkpoint files.	"Step 11: Specify where files and directories are located" on page 351
START	Start expression. Determines if a machine can run a job.	"Step 8: Manage a job's status using control expressions" on page 347
STARTD	Location of the startd executable (LoadL_startd).	"LoadLeveler daemons" on page 8
STARTER	Location of the starter executable (LoadL_starter).	"LoadLeveler daemons" on page 8
START_CLASS [ <i>class</i> ]	Specifies the job starting rule for a job class.	"Chapter 17. Using Gang scheduling" on page 381
STARTD_RUNS_HERE	Specifies whether this daemon will run on the host.	"Step 3: Define LoadLeveler machine characteristics" on page 339
START_DAEMONS	Specifies whether to start the daemons on the machine.	"Step 3: Define LoadLeveler machine characteristics" on page 339
STARTD_DGRAM_PORT	The port number used when connecting to the daemon.	"Step 13: Define network characteristics" on page 355
STARTD_STREAM_PORT	The port number used when connecting to the daemon.	"Step 13: Define network characteristics" on page 355
SUBMIT_FILTER	The program you want to run to filter a job script when the job is submitted.	"Filtering a job script" on page 285
SUSPEND	Suspend expression. Determines if a job should be suspended.	"Step 8: Manage a job's status using control expressions" on page 347
SYSPRIO	System priority expression.	"Step 6: Prioritize the queue maintained by the negotiator" on page 343
TRUNC_GSMONITOR_LOG_ON_OPEN	When <b>true</b> , specifies that the log file is restarted with every invocation of the daemon.	"Step 12: Record and control log files" on page 352
TRUNC_KBDD_LOG_ON_OPEN	When <b>true</b> , specifies the log file is restarted with every invocation of the daemon.	"Step 12: Record and control log files" on page 352
TRUNC_MASTER_LOG_ON_OPEN	When <b>true</b> , specifies the log file is restarted with every invocation of the daemon.	"Step 12: Record and control log files" on page 352

Table 14. Configuration file keywords (continued)

Configuration file keyword	Brief description	For details
TRUNC_NEGOTIATOR_LOG_ON_OPEN	When <b>true</b> , specifies the log file is restarted with every invocation of the daemon.	“Step 12: Record and control log files” on page 352
TRUNC_SCHEDD_LOG_ON_OPEN	When <b>true</b> , specifies the log file is restarted with every invocation of the daemon.	“Step 12: Record and control log files” on page 352
TRUNC_STARTD_LOG_ON_OPEN	When <b>true</b> , specifies the log file is restarted with every invocation of the daemon.	“Step 12: Record and control log files” on page 352
TRUNC_STARTER_LOG_ON_OPEN	When <b>true</b> , specifies the log file is restarted with every invocation of the daemon.	“Step 12: Record and control log files” on page 352
VACATE	The vacate expression. Determines whether suspended jobs should be vacated.	“Step 8: Manage a job’s status using control expressions” on page 347
VM_IMAGE_ALGORITHM	Virtual memory algorithm. Used for checking the image_size requirement.	“Step 17: Specify additional configuration file keywords” on page 370
WALLCLOCK_ENFORCE	When <b>true</b> , specifies that the <b>wall_clock_limit</b> on the job will be enforced. The <b>WALLCLOCK_ENFORCE</b> keyword is only valid when the External Scheduler is enabled.	See page 373
X_RUNS_HERE	When <b>true</b> , specifies that you want to start the keyboard daemon.	“Step 3: Define LoadLeveler machine characteristics” on page 339

## User-defined keywords

The following table serves only as a reference. These keywords are described in more detail in “User-defined variables” on page 65.

Keyword	Brief Description
BackgroundLoad	Defines the variable <b>BackgroundLoad</b> and assigns to it a floating point constant. This might be used as a noise factor indicating no activity.
CPU_Busy	Defines the variable <b>CPU_Busy</b> 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 <b>CPU_Idle</b> 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.
HOURL	Defines the variable <b>HOURL</b> and assigns to it a constant integer value.
JobLoad	Defines the variable <b>JobLoad</b> which defines the load on the machine caused by running the job.

## Configuration file keywords

Keyword	Brief Description
KeyboardBusy	Defines the variable <b>KeyboardBusy</b> 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.
LowLoad	Defines the variable <b>LowLoad</b> and assigns to it the value of <b>BackgroundLoad</b> . 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 <b>MINUTE</b> and assigns to it a constant integer value.
StateTimer	Defines the variable <b>StateTimer</b> and reassigns to it at each evaluation the number of seconds since the current state was entered.

## LoadLeveler variables

The following table serves only as a reference. For more information on a specific keyword, see the section and page number referenced in the “For Details” column.

Variable	Brief Description	For Details
Arch	Standard architecture of the system.	“LoadLeveler variables” on page 66
ClassSysprio	Job priority for the class.	“Step 6: Prioritize the queue maintained by the negotiator” on page 343
Cpus	Number of CPU’s installed.	“LoadLeveler variables” on page 66
ConsumableCpus	Number of ConsumableCpus currently available on the machine, if defined in <b>SCHEDULE_BY_RESOURCES</b> . If not, then it is the same as Cpus.	“LoadLeveler variables” on page 66
ConsumableMemory	Amount of ConsumableMemory currently available on the machine, if defined in <b>SCHEDULE_BY_RESOURCES</b> . If not, then it is the same as Memory.	“LoadLeveler variables” on page 66
ConsumableVirtualMemory	Amount of ConsumableVirtualMemory currently available on the machine, if defined in <b>SCHEDULE_BY_RESOURCES</b> . If not, then it is the same as VirtualMemory.	“LoadLeveler variables” on page 66
CurrentTime	The UNIX date that includes the current system time, in seconds, since January 1, 1970.	“LoadLeveler variables” on page 66
CustomMetric	The relative machine priority.	“LoadLeveler variables” on page 66
Disk	Free disk in megabytes on the file system where checkpoints are stored.	“LoadLeveler variables” on page 66
domain or domainname	Dynamically indicates the domain name of the current host machine where the program is running.	“LoadLeveler variables” on page 66
EnteredCurrentState	Value of CurrentTime when the current state was entered.	“LoadLeveler variables” on page 66
FreeRealMemory	The amount of free real memory in megabytes on the machine.	“LoadLeveler variables” on page 66

## Configuration file keywords

Variable	Brief Description	For Details
GroupQueuedJobs	The number of jobs either running or queued for the LoadLeveler group.	"Step 6: Prioritize the queue maintained by the negotiator" on page 343
GroupRunningJobs	The number of jobs currently running for the LoadLeveler group.	"Step 6: Prioritize the queue maintained by the negotiator" on page 343
GroupSysprio	The job priority for the group.	"Step 6: Prioritize the queue maintained by the negotiator" on page 343
GroupTotalJobs	The total number of jobs associated with the LoadLeveler group.	"Step 6: Prioritize the queue maintained by the negotiator" on page 343
host or hostname	Dynamically indicates the name of the host machine where the program is running.	"LoadLeveler variables" on page 66
KeyboardIdle	Number of seconds since the keyboard or mouse was last used.	"LoadLeveler variables" on page 66
LoadAvg	Berkeley one-minute load average.	"LoadLeveler variables" on page 66
Machine	Name of the current machine.	"LoadLeveler variables" on page 66
MasterMachPrio	A value that is 1 for master nodes and is 0 otherwise.	"LoadLeveler variables" on page 66
Memory	Physical memory installed on the machine in megabytes.	"LoadLeveler variables" on page 66
OpSys	Indicates the operating system on the host where the program is running.	"LoadLeveler variables" on page 66
PagesFreed	The number of pages freed per second.	"LoadLeveler variables" on page 66
PagesScanned	The number of pages scanned per second.	"LoadLeveler variables" on page 66
QDate	Difference in seconds between when the negotiator starts up and when the job is submitted.	"LoadLeveler variables" on page 66
Speed	The relative machine speed.	"LoadLeveler variables" on page 66
State	State of the startd. Can be None, Busy, Running, Idle, Suspend, Flush, or Drain.	"LoadLeveler variables" on page 66
tilde	Dynamically defines the pathname of the LoadLeveler home directory.	"LoadLeveler variables" on page 66
tm_hour	Number of hours since midnight (0-23).	"LoadLeveler variables" on page 66
tm_isdst	Daylight Savings Time flag: positive when in effect, zero when not in effect, negative when information is unavailable.	"LoadLeveler variables" on page 66
tm_mday	Number of the day of the month (1-31).	"LoadLeveler variables" on page 66
tm_min	Number of minutes after the hour (0-59).	"LoadLeveler variables" on page 66
tm_mon	Number of months since January (0-11).	"LoadLeveler variables" on page 66
tm_sec	Number of seconds after the minute (0-59).	"LoadLeveler variables" on page 66
tm_wday	Number of days since Sunday (0-6).	"LoadLeveler variables" on page 66
tm_yday	Number of days since January 1 (0-365).	"LoadLeveler variables" on page 66
tm_year	Number of years since 1900 (0-9999).	"LoadLeveler variables" on page 66
tm4_year	The four-digit integer representation of the current year.	"LoadLeveler variables" on page 66

## Configuration file keywords

Variable	Brief Description	For Details
UserPrio	User defined priority of a job.	"Step 6: Prioritize the queue maintained by the negotiator" on page 343
UserQueuedJobs	The number of jobs either running or queued for the user.	"Step 6: Prioritize the queue maintained by the negotiator" on page 343
UserRunningJobs	The number of jobs currently running for the user.	"Step 6: Prioritize the queue maintained by the negotiator" on page 343
UserSysprio	The priority of the user who submitted the job.	"Step 6: Prioritize the queue maintained by the negotiator" on page 343
UserTotalJobs	The total number of jobs associated with the this user.	"Step 6: Prioritize the queue maintained by the negotiator" on page 343
VirtualMemory	The size of the available swap space (free paging space) on the machine in kilobytes.	"LoadLeveler variables" on page 66



---

## Chapter 13. LoadLeveler daemons and job states

---

### Daemons

#### The master daemon

The **master** daemon runs on every machine in the LoadLeveler cluster, except the submit-only machine. 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.

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 administrator(s) 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
- Kill all daemons and execute a new master
- Re-run the **.llrc** file, reread the configuration files, stop or start daemons as appropriate for the new configuration files
- Send drain request to **startd** and **schedd**
- Send flush request to **startd** and send result to caller
- Send suspend request to **startd** and send result to caller
- Send resume request to **startd** and **schedd**, and send result to caller

#### 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.

## Daemons

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

### Available

This machine is available to schedule jobs.

### Draining

The schedd daemon has been 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.

### Drained

The schedd machine accepts no more jobs; jobs in the Starting or Running state are allowed to continue running, and jobs in the Idle state are drained, meaning they will not get dispatched.

**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 ids when requested by the job submission process (for example, by the **lsubmit** command).
- Receives new jobs from the **lsubmit** 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.
  - 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 a POE job.
- Requests that a remote startd daemon end a job.
- Receives accounting information from startd.

## The startd daemon

The **startd** daemon monitors jobs and machine resources on the local machine 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.

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

**Busy** The maximum number of jobs are running on this machine.

**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. However, any jobs that are already running on the startd machine will be allowed to complete.

**Draining**

The startd daemon has been 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. These variables are described in “LoadLeveler variables” on page 66.
  - State (of the startd daemon)
  - EnteredCurrentState
  - Memory
  - Disk
  - KeyboardIdle
  - Cpus
  - LoadAvg
  - Machine
  - Adapter
  - AvailableClasses
- Calculates the SUSPEND, RESUME, CONTINUE, and VACATE expressions. These are described in “Step 8: Manage a job’s status using control expressions” on page 347.
- Receives job requests from the schedd daemon to:
  - Start a job

## Daemons

- Vacate a job
- Cancel

When the schedd daemon tells the startd to start a job, the 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 <b>starter</b> 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"><li>– Jobs have been suspended, flushed, or drained</li><li>– The job limit set for the <b>MAX_STARTERS</b> keyword has been reached</li><li>– There are not enough classes available for the designated job class</li></ul>

- Receives requests from the master (via **lctd**) to do one of the following:
  - Drain
  - Flush
  - Suspend
  - Resume.
- 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 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 startd.

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. The starter child creates a new process group of which it is the process group leader, and executes the user's program or a shell. The starter parent is responsible for detecting the termination of the starter child. LoadLeveler does not monitor the children of the parent.
- 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.

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
  - 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 7. Administering and configuring LoadLeveler” on page 59.

## The gsmonitor daemon

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 and notify the negotiator quickly when a machine is no longer reachable. Because it uses the

## Daemons

GSAPI, the gsmonitor daemon requires that the Group Services subsystem, which is provided by the IBM Parallel System Support Programs (PSSP), be installed and operational.

The gsmonitor daemon should be run on one or two nodes in each of the Group Services domains. By running LoadL\_GSmonitor on two nodes, this allows for a backup in case one of the nodes goes down. A Group Services domain consists of the set of nodes that makes up a system partition. 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, including those that are not in the LoadLeveler cluster.

To start the gsmonitor daemon, set **GSMONITOR\_RUNS\_HERE** to True in the local config file. The default for **GSMONITOR\_RUNS\_HERE** is False.

Notes:

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.

It will not cause a problem to run more than one LoadL\_GSmonitor daemon per SP System Partition, this will just cause the negotiator to be notified by each running daemon.

For more information about the Group Services subsystem, see *PSSP: Administration Guide*, SA22-7348. For more information about GSAPI, see *Group Services Programming Guide and Reference*, SA22-7355.

---

## Job states

Possible job states are:

### Canceled

The job was canceled either by a user or by an administrator.

### Checkpointing

Indicates that a checkpoint has been initiated.

### Completed

The job has completed.

### Complete Pending

The job is in the process of being completed.

### Deferred

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** The job is being considered to run on a machine, though no machine has been selected.

### NotQueued

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 444. (Only the negotiator places a job in the NotQueued state.)

### **Not Run**

The job will never be run because a dependency associated with the job was found to be false.

### **Pending**

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**

The job is preempted.

### **Preempt Pending**

The job is in the process of being preempted.

### **Rejected**

The job is rejected.

### **Reject Pending**

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**

The job was stopped by LoadLeveler.

### **Remove Pending**

The job is in the process of being removed, but not all associated machines have acknowledged the removal of the job.

### **Resume Pending**

The job is in the process of being resumed.

### **Running**

The job is running: the job was dispatched and has started on the designated machine.

### **Starting**

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**

The job has been put in system hold.

### **Terminated**

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

## Job states

may actually complete and therefore be removed from the scheduler'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 `NEGOTIATOR_REMOVE_COMPLETED` keyword in the configuration file). When this time has passed, the negotiator will remove the jobs from its active list.

### **User and System Hold**

The job has been put in both system hold and user hold.

### **User Hold**

The job has been put in user hold.

### **Vacated**

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 `VACATE` 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 `VACATE` expression, see "Step 8: Manage a job's status using control expressions" on page 347.

### **Vacate Pending**

The job is in the process of being vacated.

You may also see other states that include "Pending," such as Complete Pending and Vacate Pending. These are intermediate, temporary states usually associated with parallel jobs.



---

## Chapter 14. Commands

---

## llacctmrg - Collect machine history files

### Purpose

Collects individual machine history files together into a single file specified as a parameter.

### Syntax

```
llacctmrg [-?] [ -H] [-v] [-h hostlist] [-d directory]
```

### Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
- 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**. Information such as the amount of resources consumed by the job and other job-related data is stored in this file. In this file:

**YYYY** Indicates the year  
**MM** Indicates the month  
**DD** Indicates the day  
**HH** Indicates the hour  
**mm** Indicates the minute.

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.

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

The following example collects data from machines named mars and pluto:

```
llacctmrg -h mars pluto
```

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
```

## Results

The following shows a sample system response from the **llacctmrg -h mars -d merge** command.

```
llacctmrg: History transferred successfully from mars (10080 bytes)
```

## llcancel - Cancel a submitted job

---

### Purpose

Cancels one or more jobs from the LoadLeveler queue.

### Syntax

```
llcancel [-?] [-H] [-v] [-q] [-u userlist] [-h hostlist] [joblist]
```

### Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
- q Specifies quiet mode: print no messages other than error messages.
- 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 jobs of the form *host.jobid.stepid* where:
  - *host* is the name of the schedd machine to which the job was submitted (delimited by dot). The default is the local machine.
  - *jobid* is the job ID assigned to the job when it was submitted using the **llsubmit** command. The *jobid* is required.
  - *stepid* (delimited by dot) is the step ID assigned to the job when it was submitted using the **llsubmit** command. The default is to include all steps of the job.

The -u or -h flags override the *host.jobid.stepid* parameters.

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 (Cancelled) 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

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
```

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
```

## Results

The following shows a sample system response for the **llcancel gold.8** command.

```
llcancel: Cancel command has been sent to the central manager.
```

## llckpt - Checkpoint a running job step

---

### Purpose

Checkpoints a single job step.

**Note:** Before you consider using the Checkpoint/Restart function refer to the LoadL.README file in /usr/lpp/LoadL/READMEs for information on availability and support of this function.

### Syntax

```
llckpt { -? | -H | -v | [-k | -u] [-r] [-q] <jobstep> }
```

### Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
- 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.
- 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, will not print any messages other than error messages.

#### jobstep

Specifies the name of a job step to be checkpointed using the form host.jobid.stepid where:

- host: the name of the schedd machine to which the job was submitted (default is the local machine)
- jobid: the job ID assigned to the job when it was submitted using the llsubmit command (jobid is required)
- stepid: the step ID assigned to the job when it was submitted using the llsubmit command (stepid is required)

### Description

The **llckpt** command should be used to save the state of the job in the event it does not complete. 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.

## Examples

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
```

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, 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: Checkpoint of job step iron.12.1 completed. No status information is available.
```

## llclass - Query class information

### Purpose

Returns information about classes.

### Syntax

```
llclass [-?] [-H] [-v] [-l] [classlist]
```

### Flags

- ?** Provides a short usage message.
- H** Provides intended help information.
- v** Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
- 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.

#### *classlist*

Is a blank-delimited list of classes for which you are requesting status. If no *classlist* is 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.

### Examples

This example generates a long listing for classes named *silver* and *gold*:

```
llclass -l silver gold
```

### Results

**The Standard Listing:** The standard listing is generated when you do *not* specify **-l** with the **llclass** command. The following is sample output from the **llclass Parallel** command, where there are 24 initiators of class **Parallel** configured in the cluster, with one job step of class **Parallel** using 6 initiators currently running:

Name	MaxJobCPU d+hh:mm:ss	MaxProcCPU d+hh:mm:ss	Free Slots	Max Slots	Description
Parallel	2+02:45:00	05:30:00	18	24	Parallel job class

The standard listing includes the following fields:

**Name** The name of the class.

#### **MaxJobCPU**

The hard job CPU limit of job steps for the specified class. See "job\_cpu\_limit" on page 94 for a description of job CPU limit for serial and parallel job steps.

#### **MaxProcCPU**

The hard CPU limit for the processes of the job steps of the specified class.

#### **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.

### Max Slots

The number of configured initiators (slots) for the specified class in the LoadLeveler cluster.

### 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.

**The Long Listing:** The long listing is generated when you specify the **-l** option on the **lclass** command. The following is sample output from the **lclass -l Parallel** command, where there are 24 initiators of class Parallel configured in the cluster, with one job step of class Parallel using 6 initiators currently running:

```

===== Class Parallel =====
      Name: Parallel
      Priority: 70
      Exclude_Users:
      Include_Users: aliceb johnmt loadl rhclark srherb
      Exclude_Groups:
      Include_Groups: chemistry physics
                     Admin: loadl brownap alice
      NQS_class: F
      NQS_submit:
      NQS_query:
      Max_processors: -1
      Maxjobs: -1
      Resource_requirement: cons_res1(1) cons_res2(3)
      Class_comment: Parallel job class
      Checkpoint_directory:
        Ckpt_limit: undefined, undefined
      Wall_clock_limit: 10+10:30:01, 9+14:55:00 (901801 seconds, 831300 seconds)
      Job_cpu_limit: 2+02:45:00, 2+01:30:00 (182700 seconds, 178200 seconds)
      Cpu_limit: 05:30:00, 05:00:01 (19800 seconds, 18001 seconds)
      Data_limit: 5.500 gb, 4.400 gb (5905580032 bytes, 4724464025 bytes)
      Core_limit: 8.000 gb, 8.000 gb (8589934592 bytes, 8589934592 bytes)
      File_limit: 1.500 tb, 1.200 tb (1649267441664 bytes, 1319413953331 bytes)
      Stack_limit: 400.000 mb, 300.000 mb (419430400 bytes, 314572800 bytes)
      Rss_limit: 3.000 pb, 2.000 pb (3377699720527872 bytes, 2251799813685248 bytes)
      Nice: 10
      Free_slots: 18
      Maximum_slots: 24
      Execution_factor: 1
      Max_total_tasks: 30
      Preempt_class: ALL { large } ENOUGH { small medium }
      Start_class: ( No_Class < 3 ) && ( 85ba < 10 )

```

The long listing includes these fields:

### Admin

The list of administrators for 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.

### Ckpt\_limit

Hard and soft checkpoint limits of a job step of the specified class.

### Checkpoint\_directory

The name of the directory containing the checkpointing files of job steps of the specified class.

## llclass

### **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.

### **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.

### **Execution\_factor**

Used only for Gang scheduling, Execution\_factor sets the relative processing time for 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. See "job\_cpu\_limit" on page 94 for a description of job CPU limit for serial and parallel job steps.

### **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 only for Gang scheduling, 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.

**Name** The name of the class

**Nice** The *nice* value of jobs of the specified class.

**NQS\_class**

Indicates whether this class is a gateway for an NQS system.

**NQS\_query**

The NQS queues to query where the job has been dispatched.

**NQS\_submit**

The NQS queue where the job will be submitted.

**Preempt\_class**

Used only for Gang scheduling, 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.

**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 only for Gang scheduling, 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.

## 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.

## llctl - Control LoadLeveler daemons

### Purpose

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 intended help information.
- v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
- q Specifies quiet mode: print no messages other than error messages.
- g Indicates that the command applies globally to all machines 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:

#### **purge** *list\_of\_machines*

Forces a schedd to delete any queued transaction to the machines in the *list\_of\_machines*. If all jobs on the listed machines have completed, and there are no messages pending to that machine, this option is not necessary.

This option is intended for recovery and cleanup after a machine has permanently crashed or was inadvertently removed from the LoadLeveler cluster before all activity on it was quiesced. Do not use this option unless the specified *list\_of\_machines* are guaranteed not to return to the LoadLeveler cluster.

If you need to return the machine to the cluster later, you must clear all files from the spool and execute directory of the machine which was deleted.

#### **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 76.

#### **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?" in the *IBM LoadLeveler for AIX: Diagnosis and Messages Guide*.

### 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

Starts the LoadLeveler daemons on the specified machine. You must have rsh privileges to start LoadLeveler on a remote machine.

If you are using DCE on AIX 5.1, you need the proper DCE credentials for the existing authentication method in order to run a command or function that uses **rshell** (**rsh**). Otherwise, the **rshell** command may fail. You can use the **lsauthent** command to determine the authentication method. If **lsauthent** indicates that DCE authentication is in use, you must log in to DCE with the **dce\_login** command to obtain the proper credentials.

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.

If you are using DCE on AIX 5.1, you need the proper DCE credentials for the existing authentication method in order to run a command or function that uses **rshell** (**rsh**). Otherwise, the **rshell** command may fail. You can use the **lsauthent** command to determine the authentication method. If **lsauthent** indicates that DCE authentication is in use, you must log in to DCE with the **dce\_login** command to obtain the proper credentials.

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 **SP dsh** command to send **llctl** commands (omitting the **-g** flag) to multiple nodes in a parallel fashion. For more information on **dsh**, see *IBM RS/6000 Scalable POWERparallel Systems: Administration Guide*, (SH26-2486).
- 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, which causes the central manager to ignore (fence) the target schedd 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 node.
4. Invoke "llctl -h *host* purgeschedd" to purge all of the jobs scheduled by the schedd on the failed node.
5. Remove all of the files in the LoadLeveler spool directory for that node. Once the failed node is working again, remove the "schedd\_fenced = true" statement from the administration file, then reconfigure the central manager node.

## Examples

This example stops LoadLeveler on the machine named *iron*:

```
llctl -h iron stop
```

This example starts the LoadLeveler daemons on all members of the LoadLeveler cluster, starting with the central manager, as defined in the machine stanzas of the administration file:

```
llctl -g start
```

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
```

For the next three examples, suppose the classes *small*, *medium*, and *large* are available on the machine called *iron*.

This example drains the classes *medium* and *large* on the machine named *iron*.

```
llctl -h iron drain startd medium large
```

This example drains the classes *medium* and *large* on all machines.

```
llctl -g drain startd medium large
```

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
```

This example resumes the classes *medium* and *large* on the machine named *iron*.

```
llctl -h iron resume startd medium large
```

This example illustrates how to capture accounting information on a work shift called *day* on the machine *iron*:

## llctl

```
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 76.

Assume the machine *earth* has crashed while running jobs. Its hard disk needs to be replaced. You try to cancel the jobs that were running on that machine. The schedd marks the job Remove Pending until it gets confirmation from *earth* that the jobs were removed. Since *earth* will be reinstalled, you need to inform schedd that it should not wait for confirmation.

Assume the schedd is named *mars*, and the running jobs are named *mars.1.0* and *mars.1.1*. First you want to tell the negotiator to remove the jobs:

```
llcancel mars.1.0  
llcancel mars.1.1
```

Next, tell the schedd not to wait for confirmation from *earth* before marking the jobs removed:

```
llctl -h mars purge earth
```

## Results

The following shows the result of the **llctl -h mars purge earth** command:

```
llctl: Sent purge command to host mars
```



## lldcegrpmaint - LoadLeveler DCE group maintenance utility

### Purpose

This command extracts the names of the DCE groups associated with the DCE\_ADMIN\_GROUP and DCE\_SERVICES\_GROUP keywords from the LoadLeveler configuration file. It will create these groups if they do not already exist. This command also adds the DCE principal names of the LoadLeveler daemons to the group specified by the DCE\_SERVICES\_GROUP keyword.

### Syntax

```
lldcegrpmaint [-?] [-H] [-v] <config_pathname> <admin_pathname>
```

### Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

*config\_pathname*

Pathname of the LoadLeveler configuration file.

*admin\_pathname*

Pathname of the LoadLeveler administration file.

### Description

The lldcegrpmaint command is available to DCE administrators who have logged in to DCE as **cell\_admin**. The command performs the following functions:

1. Extracts the names of the DCE groups associated with the DCE\_ADMIN\_GROUP and DCE\_SERVICES\_GROUP keywords from the LoadLeveler global configuration file. These groups are known generically as the LoadL-admin group and the LoadL-services group. The LoadL-admin group contains the DCE principal names of users who have administrative authority for LoadLeveler. The LoadL-services group contains the DCE principal names of all the LoadLeveler daemons which run in the current LoadLeveler cluster. The lldcegrpmaint command will create these groups if they do not already exist.
2. Populates the LoadL-services group with the DCE principal names of the LoadLeveler daemons. These names are derived from the DCE hostnames associated with the dce\_host\_name keyword in the LoadLeveler administration file, and LoadLeveler related information defined in the /usr/lpp/ssp/config/spsec\_defaults file. In order for this step to work, the machine stanzas in the administration file must contain the DCE hostnames of the all the machines in the LoadLeveler cluster. The llexSDR command can be used to retrieve the DCE hostnames.

Before running the lldcegrpmaint command, a DCE administrator should make sure that basic DCE Security setup steps have been performed. If SMIT panels are used, the steps under the "RS/6000 SP Security" panel should be performed in sequence (from top to bottom) to properly update the DCE Registry. This measure is important for LoadLeveler, and for any other function that exploits DCE Security on the SP. For the purposes of the lldcegrpmaint command, the important actions are: (1) "Create dcehostnames" and (2) "Configure SP Trusted Services to use DCE Authentication."

## lldcegrpmaint

Note: lldcegrpmaint does not add the names associated with the LOADL\_ADMIN keyword in the configuration file to the LoadL-admin group. It is the administrator's responsibility to add appropriate DCE principals to this group.

## Examples

In this example, it is assumed that the DCE cell name is `/.../c163.ppd.pok.ibm.com` and that LoadLeveler configuration and administration files are named `/u/loadl/LoadL_config` and `/u/loadl/LoadL_admin`, respectively, and contain the statements:

```
DCE_ENABLEMENT=TRUE
DCE_ADMIN_GROUP=LoadL-admin4
DCE_SERVICES_GROUP=LoadL-services4
```

and

```
c163n02.ppd.pok.ibm.com: type = machine central_manager = true
machine_mode = general
schedd_host = true
dce_host_name = c163n02.ppd.pok.ibm.com

c163n03.ppd.pok.ibm.com: type = machine central_manager = false
machine_mode = general
schedd_host = true
dce_host_name = c163n03.ppd.pok.ibm.com
```

It is also assumed that there is no override specification in the file `/spdata/sys1/spsec/spsec_overrides` and that the file `/usr/lpp/ssp/config/spsec_defaults` contains the following:

```
SERVICE:LoadL/Master:kw:root:system
SERVICE:LoadL/Negotiator:kw:root:system
SERVICE:LoadL/Schedd:kw:root:system
SERVICE:LoadL/Startd:kw:root:system
SERVICE:LoadL/Starter:kw:root:system
SERVICE:LoadL/Kbdd:kw:root:system
SERVICE:LoadL/GSmonitor:kw:root:system
```

Executing the command:

```
lldcegrpmaint /u/loadl/LoadL_config /u/loadl/LoadL_admin
```

results in:

1. The creation of the DCE groups:

```
/.../c163.ppd.pok.ibm.com/LoadL-admin4
/.../c163.ppd.pok.ibm.com/LoadL-services4
```

2. The population of the DCE group LoadL-services4 with the DCE principals:

```
/.../c163.ppd.pok.ibm.com/LoadL/c163n02.ppd.pok.ibm.com/Master
/.../c163.ppd.pok.ibm.com/LoadL/c163n02.ppd.pok.ibm.com/Negotiator
/.../c163.ppd.pok.ibm.com/LoadL/c163n02.ppd.pok.ibm.com/Schedd
/.../c163.ppd.pok.ibm.com/LoadL/c163n02.ppd.pok.ibm.com/Startd
/.../c163.ppd.pok.ibm.com/LoadL/c163n02.ppd.pok.ibm.com/Starter
/.../c163.ppd.pok.ibm.com/LoadL/c163n02.ppd.pok.ibm.com/Kbdd
/.../c163.ppd.pok.ibm.com/LoadL/c163n02.ppd.pok.ibm.com/GSmonitor
/.../c163.ppd.pok.ibm.com/LoadL/c163n03.ppd.pok.ibm.com/Master
/.../c163.ppd.pok.ibm.com/LoadL/c163n03.ppd.pok.ibm.com/Negotiator
/.../c163.ppd.pok.ibm.com/LoadL/c163n03.ppd.pok.ibm.com/Schedd
/.../c163.ppd.pok.ibm.com/LoadL/c163n03.ppd.pok.ibm.com/Startd
/.../c163.ppd.pok.ibm.com/LoadL/c163n03.ppd.pok.ibm.com/Starter
/.../c163.ppd.pok.ibm.com/LoadL/c163n03.ppd.pok.ibm.com/Kbdd
/.../c163.ppd.pok.ibm.com/LoadL/c163n03.ppd.pok.ibm.com/GSmonitor
```

## llexSDR - Extract adapter information from the SDR

### Purpose

Extracts adapter information from the system data repository (SDR) and creates adapter and machine stanzas for each node in an RS/6000 SP partition. You can use the information in these stanzas in the LoadLeveler administration file. This command writes the stanzas to standard output.

### Syntax

```
llexSDR [-?] [-H] [-v] [-a adapter]
```

### Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
- a *adapter*  
Specifies that the interface name of the given *adapter* on each node is used as the label (machine stanza name) of the generated machine stanza. If you do not specify an *adapter*, the label used is the **initial\_hostname** field of the Node class in the SDR.

### Description

In the SDR, the Node class contains an entry for each node in the SP partition. The Adapter class contains an entry for each adapter configured on a node. This command extracts the information in the Adapter class and creates an adapter stanza. This command also creates a machine stanza which identifies the node and the adapters attached to the node. The generated machine stanza also includes the **spacct\_exclude\_enable** keyword, whose value is obtained from the `spacct_exclude_enable` attribute in the SP class of the SDR. For more information on adapter stanzas, see “Step 5: Specify adapter stanzas” on page 332. For more information on machine stanzas, see “Step 1: Specify machine stanzas” on page 310.

The partition for which information is extracted is either the default partition or that specified with the `SP_NAME` environment variable. For the control workstation, the default partition is the default system partition. For an SP node, the default partition is the partition to which the node belongs.

You must issue this command on a machine with the `ssp.clients` file set installed. If you issue this command from a non-SP workstation, you must set `SP_NAME` to the IP address of the appropriate SDR instance for the partition.

### Examples

The following example creates adapter and machine stanzas for all nodes in a partition:

```
llexSDR
```

The following example creates machine stanzas with each node's `css0` interface name as the label:

```
llexSDR -a css0
```

## llexSDR

## Results

You may need to alter or add information to the stanzas produced by this command when you incorporate the stanzas into the administration file. For example, administrators may want to have each **network\_type** field use a value that reflects the type of nodes installed on the network. Users will need to know the values used for **network\_type** so that they can specify an appropriate value in their job command files.

Also, the output of this command includes fully-qualified machine names. If your existing administration file uses short names, you may need to change either the command output or your existing administration file so that you use either all fully-qualified names or all short names.

This is sample output for the **llexSDR** command:

```
#llexSDR: System Partition = "c97s" on Wed Aug 29 16:43:13 2001
```

```
c98n05.ppd.pok.ibm.com: type = machine
  adapter_stanzas = c97san04.ppd.pok.ibm.com c97s2n04.ppd.pok.ibm.com
                  c97sn04.ppd.pok.ibm.com c98n05.ppd.pok.ibm.com
  spacct_exclude_enable = false
  dce_host_name = c98n05.ppd.pok.ibm.com
  alias = c97san04.ppd.pok.ibm.com c97s2n04.ppd.pok.ibm.com
         c97sn04.ppd.pok.ibm.com
```

```
c97san04.ppd.pok.ibm.com: type = adapter
  adapter_name = m10
  network_type = multilink
  interface_address = 9.114.59.196
  interface_name = c97san04.ppd.pok.ibm.com
  multilink_list = css0,css1
```

```
c97s2n04.ppd.pok.ibm.com: type = adapter
  adapter_name = css1
  network_type = switch
  interface_address = 9.114.59.4
  interface_name = c97s2n04.ppd.pok.ibm.com
  multilink_address = 9.114.59.196
  switch_node_number = 3
  css_type = SP_Switch2_Adapter
```

```
c97sn04.ppd.pok.ibm.com: type = adapter
  adapter_name = css0
  network_type = switch
  interface_address = 9.114.59.132
  interface_name = c97sn04.ppd.pok.ibm.com
  multilink_address = 9.114.59.196
  switch_node_number = 3
  css_type = SP_Switch2_Adapter
```

```
c98n05.ppd.pok.ibm.com: type = adapter
  adapter_name = en0
  network_type = ethernet
  interface_address = 9.114.59.70
  interface_name = c98n05.ppd.pok.ibm.com
```

```
.
.
.
```

```
c97n05.ppd.pok.ibm.com: type = machine
  adapter_stanzas = c97san02.ppd.pok.ibm.com c97s2n02.ppd.pok.ibm.com
                  c97sn02.ppd.pok.ibm.com c97n05.ppd.pok.ibm.com
  spacct_exclude_enable = false
```

```

dce_host_name = c97n05.ppd.pok.ibm.com
alias = c97san02.ppd.pok.ibm.com c97s2n02.ppd.pok.ibm.com
      c97sn02.ppd.pok.ibm.com

c97san02.ppd.pok.ibm.com: type = adapter
  adapter_name = m10
  network_type = multilink
  interface_address = 9.114.59.194
  interface_name = c97san02.ppd.pok.ibm.com
  multilink_list = css0,css1

c97s2n02.ppd.pok.ibm.com: type = adapter
  adapter_name = css1
  network_type = switch
  interface_address = 9.114.59.2
  interface_name = c97s2n02.ppd.pok.ibm.com
  multilink_address = 9.114.59.194
  switch_node_number = 1
  css_type = SP_Switch2_Adapter

c97sn02.ppd.pok.ibm.com: type = adapter
  adapter_name = css0
  network_type = switch
  interface_address = 9.114.59.130
  interface_name = c97sn02.ppd.pok.ibm.com
  multilink_address = 9.114.59.194
  switch_node_number = 1
  css_type = SP_Switch2_Adapter

c97n05.ppd.pok.ibm.com: type = adapter
  adapter_name = en0
  network_type = ethernet
  interface_address = 9.114.59.66
  interface_name = c97n05.ppd.pok.ibm.com

```

This is sample output for the llexSDR -a css0 command:

```
#llexSDR: System Partition = "c97s" on Wed Aug 29 17:24:07 2001
```

```

c97sn04.ppd.pok.ibm.com: type = machine
  adapter_stanzas = c97san04.ppd.pok.ibm.com c97s2n04.ppd.pok.ibm.com
                  c97sn04.ppd.pok.ibm.com c98n05.ppd.pok.ibm.com
  spacct_exclude_enable = false
  alias = c97san04.ppd.pok.ibm.com c97s2n04.ppd.pok.ibm.com
        c98n05.ppd.pok.ibm.com

c97san04.ppd.pok.ibm.com: type = adapter
  adapter_name = m10
  network_type = multilink
  interface_address = 9.114.59.196
  interface_name = c97san04.ppd.pok.ibm.com
  multilink_list = css0,css1

c97s2n04.ppd.pok.ibm.com: type = adapter
  adapter_name = css1
  network_type = switch
  interface_address = 9.114.59.4
  interface_name = c97s2n04.ppd.pok.ibm.com
  multilink_address = 9.114.59.196
  switch_node_number = 3
  css_type = SP_Switch2_Adapter

c97sn04.ppd.pok.ibm.com: type = adapter
  adapter_name = css0
  network_type = switch
  interface_address = 9.114.59.132

```

## IlexSDR

```
interface_name = c97sn04.ppd.pok.ibm.com
multilink_address = 9.114.59.196
switch_node_number = 3
css_type = SP_Switch2_Adapter

c98n05.ppd.pok.ibm.com: type = adapter
adapter_name = en0
network_type = ethernet
interface_address = 9.114.59.70
interface_name = c98n05.ppd.pok.ibm.com
```

## llfavorjob - Reorder system queue by job

### Purpose

Sets specified jobs to a higher system priority than all jobs that are not favored. This command also *unfavors* previously favored job(s), 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** Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
- 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 jobs of the form *host.jobid.stepid* where:

- *host* is the name of the schedd machine to which the job was submitted (delimited by dot). The default is the local machine.
- *jobid* is the job ID assigned to the job by LoadLeveler when it was submitted using the **lsubmit** command. *jobid* is required.
- *stepid* (delimited by dot) is the job step ID assigned to the job by LoadLeveler when it was submitted using the **lsubmit** command. The default is to include all members of the job.

### 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

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 iron.12.4 zinc.8.2
```

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
```

---

## llfavoruser - Reorder system queue by user

### Purpose

Sets a user's job(s) 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 job(s), 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** Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
- q** Specifies quiet mode: print no messages other than error messages.
- u** Unfavors previously favored users, reordering their job(s) according to their original priority level(s). If **-u** is **not** specified, the user's job(s) 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

This example grants highest priority to all queued jobs submitted by users `ellen` and `fred` according to the **sysprio** expression:

```
llfavoruser ellen fred
```

This example unfavors all queued jobs submitted by users `ellen` and `fred`:

```
llfavoruser -u ellen fred
```



## llhold - Hold or release a submitted job

### Purpose

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] [-u <userlist>] [-h <hostlist>] [<joblist>]
```

### Flags

- ?** Provides a short usage message.
- H** Provides extended help information.
- v** Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
- q** Specifies quiet mode: print no messages other than error messages.
- s** Puts job(s) in system hold. Only a LoadLeveler administrator can use this option.  
 If neither **-s** nor **-r** is specified, LoadLeveler puts the job(s) in user hold.
- r** Releases a job from hold. A job in user hold is released unless it is also in system hold, where it remains. A job in system hold is released unless it is also in user hold, where it remains.  
 Only a LoadLeveler administrator can release jobs from system hold. Only an administrator or the owner of a job can release it from user hold.  
 If neither **-s** nor **-r** is specified, LoadLeveler puts the job(s) in user hold.
- 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 jobs of the form *host.jobid.stepid* where:
  - *host* is the name of the schedd machine to which the job was submitted (delimited by dot). The default is the local machine.
  - *jobid* is the job ID assigned to the job when it was submitted using the **lsubmit** command. *jobid* is required.

## llhold

- *stepid* (delimited by dot) is the step ID assigned to the job by LoadLeveler when it was submitted using the **llsubmit** command. The default is to include all steps of the job.

## 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 non-administrator attempts to move a job into or out of system hold.
- a non-administrator attempts to move a job submitted by someone else into or out of user hold.

## Examples

This example places job 23, job step 0 and job 19, job step 1 on hold:

```
llhold 23.0 19.1
```

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
```

This example places all jobs from users abe, barbara, and carol2 in system hold:

```
llhold -s -u abe barbara carol2
```

This example releases from a hold state all jobs on machines bronze, iron, and steel:

```
llhold -r -h bronze iron steel
```

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
```

## 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.
```

## llinit - Initialize machines in the LoadLeveler cluster

### Purpose

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 sub-directories 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.
- 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 loadl home directory.

## 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
```

## 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.
```

## llmatrix - Query Gang matrix

### Purpose

Returns the Gang matrix information in the LoadLeveler cluster when Gang scheduling is used.

### Syntax

```
llmatrix [-? | -H | -v | [-r] [-s] <hostlist> ]
```

### Flags

- ? Provides a short usage message.
- H Provides help information.
- v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
- r Indicates raw mode for output. Each column of the Gang matrix will occupy one line with the output fields separated by an exclamation point (!).
- s Specifies that the short hostnames will be used in the output.

#### *hostlist*

A blank-delimited list of machines for which you are requesting Gang matrix information.

### Description

If you do not specify a hostlist, the system displays the complete Gang matrix.

Gang matrix output information is formatted into columns. Each column is identified by the machine name and processor number. Each row in the matrix is called a time-slice and contains job step names. Job steps in an active time-slice run for the time period specified by GANG\_MATRIX\_TIME\_SLICE. When that time-slice expires, the job steps in the next time-slice are run. At the end of the last time-slice, the cycle is restarted at the first time-slice.

In the matrix output:

- "\*" following a job step name indicates the active time-slice
- "NULL" for a job step name indicates that no job step is there

**Note:** The processor number does not correspond to a particular physical processor.

### Examples

This example displays the complete Gang matrix:

```
llmatrix
```

This example displays the portion of the Gang matrix for machine c163n03 in raw mode:

```
llmatrix -r c163n03
```

### Results

**llmatrix** returns the following exit values:

- 0 (command successfully returns Gang matrix information)

## llmatrix

- -1 (an error occurred)

The following listing is sample output for the **llmatrix -s** command:

```
GANG_MATRIX_TIME_SLICE = 60 seconds

Column: Machine Name / Processor Number
        Job Step Names

Column: c209f1n05 / Processor0
        c209f1n05.11.0
        c209f1n05.14.0*
        c209f1n05.13.0

Column: c209f1n05 / Processor1
        c209f1n05.11.0
        c209f1n05.14.0*
        c209f1n05.13.0

Column: c209f1n05 / Processor2
        c209f1n05.12.0
        c209f1n05.14.0*
        NULL
```

Figure 19. *llmatrix -s* output (Part 1 of 2)

```
Column: c209f1n05 / Processor3
        c209f1n05.12.0
        c209f1n05.14.0*
        NULL

Column: c209f1n01 / Processor0
        c209f1n05.11.0
        c209f1n05.14.0*
        c209f1n05.13.0

Column: c209f1n01 / Processor1
        c209f1n05.11.0
        c209f1n05.14.0*
        c209f1n05.13.0

Column: c209f1n01 / Processor2
        c209f1n05.12.0
        c209f1n05.14.0*
        NULL

Column: c209f1n01 / Processor3
        c209f1n05.12.0
        c209f1n05.14.0*
        NULL
```

Figure 19. *llmatrix -s* output (Part 2 of 2)

The following listing is sample output for the **llmatrix -s -r c209f1n05 c209f1n01** command:

```
c209f1n05!Processor0!c209f1n05.11.0,c209f1n05.14.0*,c209f1n05.13.0
c209f1n05!Processor1!c209f1n05.11.0,c209f1n05.14.0*,c209f1n05.13.0
c209f1n05!Processor2!c209f1n05.12.0,c209f1n05.14.0*,NULL
c209f1n05!Processor3!c209f1n05.12.0,c209f1n05.14.0*,NULL
c209f1n01!Processor0!c209f1n05.11.0,c209f1n05.14.0*,c209f1n05.13.0
c209f1n01!Processor1!c209f1n05.11.0,c209f1n05.14.0*,c209f1n05.13.0
c209f1n01!Processor2!c209f1n05.12.0,c209f1n05.14.0*,NULL
c209f1n01!Processor3!c209f1n05.12.0,c209f1n05.14.0*,NULL
```

Figure 20. llmatrix -s -r output

## llmodify - Change attributes of a submitted job step

### Purpose

Changes the attributes or characteristics of a submitted job.

### Syntax

```
llmodify { -? | -H | -v | [-q] { -x <execution_factor> | -c <consumable_cpus> | -m
<consumable_real_memory>} <jobstep> }
```

### Flags

- ? Provides a short usage message.
  - H Provides extended help information.
  - v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
  - q Specifies quiet mode: print no messages other than error messages.
  - x <execution\_factor>  
For Gang scheduling only, specifies the execution factor value. Valid values are 1 through 3 and 99.
    - 99 – puts the job step into non-preemptable state. All other jobs running on the same nodes will be preempted until this job finish running.
    - 1 to 3 – specifies the relative share of CPU time the job step will receive when Gang scheduling is used.
  - c <consumable\_cpus>  
Specifies the consumable CPU value.  
  
Allows the ConsumableCpus resource requirement to be reset to the specified value. This value can be any integer equal or greater than zero (0) and should follow the rules for the **resources** keyword in the job command file. A value of zero (0) indicates that all previous resource requirements will be removed. It is not possible to literally request zero (0) resources.
  - m <consumable\_real\_memory>  
Specifies the consumable real memory value.  
  
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 equal or greater than zero (0) and should follow the rules for the **resources** keyword in the job command file. A value of zero (0) indicates that all previous resource requirements will be removed. It is not possible to literally request zero (0) resources.
- jobstep*  
Is the name of a job step to be modified using the form of *host.jobid.stepid*
- *host* : The name of the schedd machine to which the job was submitted. The default is the local machine.
  - *jobid* : The ID assigned to the job when it was submitted using the llsubmit command. A jobid is required.
  - *stepid* : The step ID assigned to the job when it was submitted using the llsubmit command. A stepid is required.



## Description

To determine if the request was successful, issue the **llq -x -l** command and check the resource requirement variable.

The following comments apply to the -x option (execution factor):

- Only LoadLeveler administrators have authority to use this option
- A job step must be in the "Running" state before the execution factor can be set to 99 (prevents the job step from being preempted)
- When you set the execution factor of a job step to 99, all other jobs sharing nodes with the job step are preempted until the job step finishes or the execution factor is lowered

The initial value of `execution_factor` for a job step is the value specified for its job class. This command will override the class value for a specific job step.

The following comments apply to the -c and -m options (consumable CPUs and consumable real memory):

- Can be used by either the LoadLeveler administrator or the job step owner
- A job step must be in one of the following states:
  - Idle
  - Deferred
  - User Hold
  - System Hold
  - Not Queued
  - Vacate
  - Vacate Pending
  - Rejected
  - Reject Pending
- At the time that the job step is modified, LoadLeveler does not check to make certain that the modified values for resource requirements can be met.

## Examples

This example puts the job step `c163n07.12.0` into a non-preemptable state and preempts all other job steps running on the same nodes:

```
llmodify -x 99 c163n07.12.0
```

This example requests that the execution factor of job step `c163n07.12.0` be changed to 3:

```
llmodify -x 3 c163n07.12.0
```

## Results

The following shows a sample system response for `llmodify -x 3 c163n07.12.0`:  
llmodify: request has been sent to LoadLeveler.

**llmodify** returns the following exit values:

- 0 (success)
- -1 (error)

---

# llpreempt - Preempt a submitted job step

## Purpose

Places a specified job step in the (user-initiated) preempted state. The job step will stay in that state until the action is undone with the -u flag. After the undo operation, the job step resumes its normal state controlled by the LoadLeveler scheduling and preemption rules.

## Syntax

```
llpreempt { -? | -H | -v | [-q] [-u] <jobstep> }
```

## Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
- q Specifies quiet mode: print no messages other than error messages.
- u Undo the previous preempt action.

### *jobstep*

Is the name of a job step to be preempted or resumed. Uses the form *host.jobid.stepid* where:

- *host*: The name of the schedd machine to which the job step was submitted. The default is the local machine.
- *jobid*: The job ID assigned to the job when it was submitted using the **llsubmit** command. A *jobid* is required.
- *stepid*: The step ID assigned to the job when it was submitted using the **llsubmit** command. A *stepid* is required.

## Description

This is a LoadLeveler administrator command used for Gang and external schedulers only. Regular users do not have authority to execute this command.

## Examples

This example requests that job step c163n07.12.0 be preempted:

```
llpreempt c163n07.12.0
```

This example requests that job step c163n07.12.0 be resumed:

```
llpreempt -u c163n07.12.0
```

## Results

The following shows a sample system response for the **llpreempt** command:

```
llpreempt: request has been sent to LoadLeveler.
```

## llprio - Change the user priority of submitted job steps

### Purpose

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] [+<integer> | -<integer> | -p <priority>] <joblist>
```

### Flags

- ?** Provides a short usage message.
- H** Provides extended help information.
- v** Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
- q** Specifies quiet mode: print no messages other than error messages.
- +** | **-** *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 of the form *host.jobid.stepid* where:
  - *host* is the name of the schedd machine to which the job step was submitted (delimited by dot). The default is the local machine.  
If the job step was submitted from a submit-only machine, this is the name of the machine where the schedd daemon that sent the job to the negotiator resides.
  - *jobid* is the job ID assigned to the job when it was submitted using the **llsubmit** command. *jobid* is required.
  - *stepid* (delimited by dot) is the job step ID assigned to the job when it was submitted using the **llsubmit** command.

### 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.

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

## llprio

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.

See “Setting and changing the priority of a job” on page 44 for more information.

## Examples

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
```

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
```

## Results

The following shows a sample system response for the **llprio -p 100 silver.18.4** command:

```
llprio: Priority command has been sent to the central manager.
```

## llq - Query job status

### Purpose

Returns information about job steps in the LoadLeveler queues.

### Syntax

```
llq [-?] [-H] [-v] [-x] [-s] [-I] [-w] [joblist] [-u userlist] [-h hostlist] [-c classlist]
[-f category_list] [-r category_list]
```

### Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
- 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 -I, CPU usage for active jobs is reported in the short format.  
  
**Note:** Using both the -I and -x options without a joblist specification can produce a very long report and excessive network traffic.
- 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.
- I Specifies that a long listing be generated for each job for which status is requested. Fields included in the long listing are shown in “Results” on page 175.  
  
If -I is *not* specified, then the standard listing is generated as shown in “Results” on page 175.
- 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.

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

#### *joblist*

Is a blank-delimited list of jobs of the form *host.jobid.stepid* where:

- *host* is the name of the schedd machine to which the job was submitted (delimited by dot). The default is the local machine.  
If the job was submitted from a submit-only machine, this is the name of the machine where the schedd daemon that sent the job to the negotiator resides.
- *jobid* is the job id assigned to the job when it was submitted using the **llsubmit** command.
- *stepid* (delimited by dot) Is the step id assigned to the job when it was submitted using the **llsubmit** command. The default is to include all the job steps of the job associated with the *jobid*.

#### **-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 queried. When used alone, only the user's jobs monitored on the schedd machine are queried.

#### **-h** *hostlist*

Is a blank-delimited list of machines. If the **-s** flag is not specified, all jobs monitored 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 Idle state. When issued with the **-u** option, the *userlist* is used to further select jobs for querying.

#### **-c** *classlist*

Is a blank-delimited list of classes. When used with **-h**, only those jobs monitored on the machines in the *hostlist* 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:

<b>%a</b>	Account number
<b>%c</b>	Class
<b>%cc</b>	Completion code
<b>%dc</b>	Completion date
<b>%dd</b>	Dispatch Date
<b>%dh</b>	Hold date
<b>%dq</b>	Queue date
<b>%gl</b>	LoadLeveler group
<b>%gu</b>	UNIX group
<b>%h</b>	Hostname (first hostname if more than one machine is allocated to the job step)
<b>%id</b>	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  
**%sn** Step name  
**%st** Status

**-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.

If the **-u** or **-h** options are not specified, and if no *jobid* is specified, then all jobs are queried.

The **-u** and **-h** options override the *jobid* parameters.

## Examples

This example generates a long listing for job 8, job step 2 submitted to machine *gold*:

```
llq -l gold.8.2
```

This example generates a standard listing for all job steps of job name 12 submitted to the local machine:

```
llq 12
```

## Results

**Standard listing:** The standard listing is generated when you do *not* specify the **-l** option with the **llq** command. The following is sample output from the **llq -h mars** command, 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:

**Id** Job identifier presented in the format: *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

## llq

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

Userid of the job submitter.

### Submitted

Date and time of job submission.

**ST** Current job status (state). Job status can be:

<b>C</b>	Completed
<b>CA</b>	Canceled
<b>CK</b>	Checkpointing
<b>CP</b>	Complete Pending
<b>D</b>	Deferred
<b>E</b>	Preempted
<b>EP</b>	Preempt Pending
<b>H</b>	User Hold
<b>HS</b>	User Hold and System Hold
<b>I</b>	Idle
<b>MP</b>	Resume Pending
<b>NR</b>	Not Run
<b>NQ</b>	Not Queued
<b>P</b>	Pending
<b>R</b>	Running
<b>RM</b>	Removed
<b>RP</b>	Remove Pending
<b>S</b>	System Hold
<b>ST</b>	Starting
<b>SX</b>	Submission Error
<b>TX</b>	Terminated
<b>V</b>	Vacated
<b>VP</b>	Vacate Pending
<b>X</b>	Rejected
<b>XP</b>	Reject Pending

For a detailed explanation of job states, see “Job states” on page 134.

**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. See “llprio - Change the user priority of submitted job steps” on page 171

**Class** Job class.

### 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.

**Customized, formatted standard listing:** A customized and formatted standard listing is generated when you specify llq with the -f flag. The following is sample output from this command:

```
llq -f %id %c %dq %dd %g1 %h
```



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

**Customized, unformatted standard listing:** A customized and unformatted (raw) standard listing is generated when you specify **llq** with the **-r** flag. Output fields are separated by an exclamation point (!). The following is sample output from this command:

```
llq -r %id %c %dq %dd %gl %h
```

```
116.pok.ibm.com.2.0!No_Class!04/08/2001 09:19!04/08/2001 09:21!No_Group!116.pok.ibm.com
116.pok.ibm.com.1.0!No_Class!04/08/2001 09:19!04/08/2001 09:21!No_Group!116.pok.ibm.com
116.pok.ibm.com.3.0!No_Class!04/08/2001 09:19!04/08/2001 09:21!No_Group!115.pok.ibm.com
```

**WLM CPU and real memory statistics listing:** If the LoadLeveler interface to AIX Workload Manager (WLM) is enabled, then the **-w** option can be used to obtain CPU and real memory statistics of job steps in running state. The following is the output of "llq -w c209f1n05.13.0" where c209f1n05.13.0 is a CPU intensive parallel job step currently running on the 2 nodes c209f1n05 and c209f1n01:

```
===== 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
```

The output listing associated with the **-w** option includes these fields:

#### Resource

The resource being enforced by WLM. This is either CPU or Real Memory.

#### snapshot

Current CPU or Real Memory consumption as a percentage of the total resources available.

**total** Total CPU time consumed in milliseconds. CPU resource only.

#### high water

The highest number of resident memory pages used. Real Memory resource only.

**The long listing:** The long listing is generated when you specify the **-l** option with the **llq** command. This section contains sample output for two **llq** commands: one querying a serial job and one querying a parallel job. Following the sample output is an explanation of all possible fields displayed by the **llq** command.

## llq

The following is sample output for the **llq -l** command for the serial job step **c209f1n01.ppd.pok.ibm.com.2.0**:

```
===== Job Step c209f1n01.ppd.pok.ibm.com.2.0 =====
Job Step Id: c209f1n01.ppd.pok.ibm.com.2.0
Job Name: c209f1n01.ppd.pok.ibm.com.2
Step Name: job_step_1
Structure Version: 10
  Owner: loadl
  Queue Date: Wed Jul 25 15:49:17 EDT 2001
  Status: Running
Execution Factor: 1
  Dispatch Time: Wed Jul 25 15:49:17 EDT 2001
Completion Date:
Completion Code:
  User Priority: 50
  user_sysprio: 0
  class_sysprio: 35
  group_sysprio: 70
System Priority: -33
  q_sysprio: -33
  Notifications: Complete
Virtual Image Size: 24 kb
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:
Restart From Ckpt: no
Restart Same Nodes: no
  Restart: yes
Hold Job Until:
  Cmd: /tmp/LL_V2/TEST/serial_90_60
  Args: arg_01 arg_02 arg_3
  Env:
  In: /dev/null
  Out: job1.c209f1n01.2.0.out
  Err: job1.c209f1n01.2.0.err
Initial Working Dir: /tmp/LL_V2/TEST
Dependency:
  Resources: ConsumableMemory(50.000 mb) ConsumableVirtualMemory(85.000 mb) ConsumableCpus(1)
Requirements: (Arch == "R6000") && (OpSys == "AIX51") && (Memory > 128)
Preferences: (Machine == { "c209f1n01.ppd.pok.ibm.com" "c209f1n05.ppd.pok.ibm.com" })
```

Figure 21. llq -l output for a serial job step (Part 1 of 2)

```

        && (Feature == "ESSL")
    Step Type: Serial
    Min Processors:
    Max Processors:
    Allocated Host: c209f1n01.ppd.pok.ibm.com
    Node Usage: shared
    Submitting Host: c209f1n01.ppd.pok.ibm.com
    Notify User: loadl@c209f1n01.ppd.pok.ibm.com
    Shell: /bin/ksh
    LoadLeveler Group: chemistry
    Class: large
    Ckpt Hard Limit: undefined
    Ckpt Soft Limit: undefined
    Cpu Hard Limit: 02:30:00 (9000 seconds)
    Cpu Soft Limit: 02:30:00 (9000 seconds)
    Data Hard Limit: 5.500 gb (5905580032 bytes)
    Data Soft Limit: 4.400 gb (4724464025 bytes)
    Core Hard Limit: 8.000 gb (8589934592 bytes)
    Core Soft Limit: 8.000 gb (8589934592 bytes)
    File Hard Limit: 1.500 tb (1649267441664 bytes)
    File Soft Limit: 1.200 tb (1319413953331 bytes)
    Stack Hard Limit: 400.000 mb (419430400 bytes)
    Stack Soft Limit: 300.000 mb (314572800 bytes)
    Rss Hard Limit: 3.000 pb (3377699720527872 bytes)
    Rss Soft Limit: 2.000 pb (2251799813685248 bytes)
    Step Cpu Hard Limit: 04:00:30 (14430 seconds)
    Step Cpu Soft Limit: 04:00:30 (14430 seconds)
    Wall Clk Hard Limit: 00:11:40 (700 seconds)
    Wall Clk Soft Limit: 00:10:00 (600 seconds)
    Comment: Test job 1 of Serial test suite 3.
    Account: 99999
    Unix Group: loadl
    NQS Submit Queue:
    NQS Query Queues:
    Negotiator Messages:
    Adapter Requirement:
    Step Cpus: 1
    Step Virtual Memory: 85.000 mb
    Step Real Memory: 50.000 mb
    Step Adapter Memory: 0 bytes

```

Figure 21. llq -l output for a serial job step (Part 2 of 2)

The following listing is sample output for the **llq -l -x c209f1n01.1.0** command, where c209f1n01.1.0 is a parallel, non-checkpointing job step:

## llq

```
===== Job Step c209f1n05.ppd.pok.ibm.com.1.0 =====
  Job Step Id: c209f1n05.ppd.pok.ibm.com.1.0
  Job Name: c209f1n05.ppd.pok.ibm.com.1
  Step Name: parallel_job_step_1
  Structure Version: 10
  Owner: loadl
  Queue Date: Wed Jul 25 17:49:51 EDT 2001
  Status: Running
  Execution Factor: 1
  Dispatch Time: Wed Jul 25 17:49:51 EDT 2001
  Completion Date:
  Completion Code:
  User Priority: 50
  user_sysprio: 0
  class_sysprio: 45
  group_sysprio: 0
  System Priority:
  q_sysprio:
  Notifications: Complete
  Virtual Image Size: 387 kb
  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:
  Restart From Ckpt: no
  Restart Same Nodes: no
  Restart: yes
  Hold Job Until:
    Env: MANPATH=/usr/local/man:/usr/share/man: ...
    In: /dev/null
    Out: poe5_1.c209f1n05.1.0.out
    Err: poe5_1.c209f1n05.1.0.err
  Initial Working Dir: /tmp/TEST/PARA
```

Figure 22. llq -l -x output for a parallel, non-checkpointing job step (Part 1 of 4)

```

Dependency:
Resources: ConsumableMemory(75.000 mb) ConsumableVirtualMemory(125.000 mb) ConsumableCpus(1)
Step Type: General Parallel
Node Usage: shared
Submitting Host: c209f1n05.ppd.pok.ibm.com
Notify User: loadl
Shell: /bin/ksh
LoadLeveler Group: No_Group
Class: Parallel
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: 4.250 pb (4785074604081152 bytes)
Data Soft Limit: 1.500 tb (1649267441664 bytes)
Core Hard Limit: 2.250 tb (2473901162496 bytes)
Core Soft Limit: 1.250 tb (1374389534720 bytes)
File Hard Limit: 1.200 eb (1383505805528216384 bytes)
File Soft Limit: 1.100 eb (1268213655067531680 bytes)
Stack Hard Limit: 40.000 mb (41943040 bytes)
Stack Soft Limit: 30.000 mb (31457280 bytes)
Rss Hard Limit: 1.200 eb (1383505805528216384 bytes)
Rss Soft Limit: 5.500 pb (6192449487634432 bytes)
Step Cpu Hard Limit: 3+08:00:00 (288000 seconds)
Step Cpu Soft Limit: 23:59:59 (86399 seconds)
Wall Clk Hard Limit: 01:40:00 (6000 seconds)
Wall Clk Soft Limit: 01:40:00 (6000 seconds)
Comment: Test job 1 of Parallel test suite 5.
Account: 99999
Unix Group: loadl
User Space Windows: 8
NQS Submit Queue:
NQS Query Queues:
Negotiator Messages:
Adapter Requirement: (css0,LAPI,shared,US),(css0,MPI,shared,US)
Step Cpus: 4
Step Virtual Memory: 500.000 mb
Step Real Memory: 300.000 mb
Step Adapter Memory: 8.000 mb (8388608 bytes)

```

Figure 22. llq -l -x output for a parallel, non-checkpointing job step (Part 2 of 4)

## llq

```
----- Detail for c209f1n05.ppd.pok.ibm.com.1.0 -----  
Running Host: c209f1n05.ppd.pok.ibm.com  
Machine Speed: 1.000000  
Starter User Time: 00:00:00.230000  
Starter System Time: 00:00:00.190000  
Starter Total Time: 00:00:00.420000  
Starter maxrss: 1972  
Starter ixrss: 8788  
Starter idrss: 13468  
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 nvcs: 82  
Starter nivcs: 56  
Step User Time: 00:01:20.460000  
Step System Time: 00:00:00.790000  
Step Total Time: 00:01:21.250000  
Step maxrss: 4312  
Step ixrss: 52544  
Step idrss: 9308828  
Step isrss: 0  
Step minflt: 6941  
Step majflt: 0  
Step nswap: 0  
Step inblock: 0  
Step oublock: 0  
Step msgsnd: 0  
Step msgrcv: 0  
Step nsignals: 0  
Step nvcs: 507
```

Figure 22. llq -l -x output for a parallel, non-checkpointing job step (Part 3 of 4)

```

Step nivcs: 8515
-----
Node
----

Name      :
Requirements : (Arch == "R6000") && (OpSys == "AIX51") && (Memory > 128)
Preferences : (Machine == { "c209f1n01.ppd.pok.ibm.com" "c209f1n05.ppd.pok.ibm.com" })
            && (Feature == "ESSL")

Node minimum : 2
Node maximum : 2
Node actual   : 2
Allocated Hosts : c209f1n05.ppd.pok.ibm.com:RUNNING:css0(1,LAPI,US,1M),css0(2,MPI,US,1M),
                  css0(3,LAPI,US,1M),css0(4,MPI,US,1M)
                  + c209f1n01.ppd.pok.ibm.com:RUNNING:css0(1,LAPI,US,1M),css0(2,MPI,US,1M),
                  css0(3,LAPI,US,1M),css0(4,MPI,US,1M)

Master Task
-----

Executable : /bin/poe
Exec Args   : /tmp/TEST/PARA/ivp_60 -eulib us -ilevel 6 -labelio yes -pmdlog yes
Num Task Inst: 1
Task Instance: c209f1n05:-1

Task
----

Num Task Inst: 4
Task Instance: c209f1n05:0:css0(1,LAPI,US,1M),css0(2,MPI,US,1M)
Task Instance: c209f1n05:1:css0(3,LAPI,US,1M),css0(4,MPI,US,1M)
Task Instance: c209f1n01:2:css0(1,LAPI,US,1M),css0(2,MPI,US,1M)
Task Instance: c209f1n01:3:css0(3,LAPI,US,1M),css0(4,MPI,US,1M)

```

Figure 22. llq -l -x output for a parallel, non-checkpointing job step (Part 4 of 4)

The long listing includes these fields:

**Job Step ID**

The job step identifier.

**Job Name**

The name of the job.

**Step Name**

The name of the job step

**Structure Version**

An internal version identifier.

**Owner**

The userid of the user submitting the job.

**Queue Date**

The date and time that LoadLeveler received the job.

**Status**

The status (state) of the job. A job's status can be:

- Canceled
- Checkpointing
- Completed
- Complete Pending
- Deferred
- Idle
- Not Queued

Not Run  
 Pending  
 Preempted (user-initiated)  
 Preempted (system-initiated)  
 Preempt Pending (user-initiated)  
 Preempt Pending (system-initiated)  
 Rejected  
 Reject Pending  
 Removed  
 Remove Pending  
 Resume Pending  
 Running  
 Starting  
 Submission Error  
 System Hold  
 System and User Hold  
 Terminated  
 User Hold  
 Vacated  
 Vacate Pending

**Note:** For a detailed explanation of these job states, see “Job states” on page 134 .

**Execution Factor**

The weight factor of the relative processing time when using Gang scheduling.

**Dispatch Time**

The time the job was dispatched.

**Completion Date**

Date and time job completed or exited.

**Completion Code**

The status returned by the wait3 UNIX system call.

**User Priority**

The priority of the job step, as specified by the user in the job command, or changed by the **llprio** command.

**user\_sysprio**

The user system priority of the job step, where the value is defined in the administration file.

**class\_sysprio**

The class priority of the job step, where the value is defined in the administration file.

**group\_sysprio**

The group priority of the job step, where the value is defined in the administration file.

**System Priority**

The overall system priority of the job step, where the value is defined by the SYSPRIO expression in the configuration file.

**q\_sysprio**

The adjusted system priority of the job step (see “How does a job's priority affect dispatching order?” on page 45).



## Notifications

The notification status for the job step, where:

### **always**

Indicates notification is sent through the mail for all four notification categories below.

### **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.

## 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.

## Restart

Restart status (yes or no)

## Checkpointable

Indicates if LoadLeveler considers the job step checkpointable (yes, no, or interval).

## Ckpt Start Time

The start time of the current checkpoint in progress. Blank if no checkpoint running.

## Good Ckpt Time/Date

Time and date stamp of the last successful checkpoint.

## Ckpt Elapse Time

Amount of time taken to perform the last successful checkpoint.

## Fail Ckpt Time/Date

Time and date stamp of the last failed checkpoint.

## Ckpt Accum Time

Accumulated time, in seconds, the job step has spent checkpointing.

## Checkpoint File

Location of the directory and file name to be used for checkpoint data.

## 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).

## Hold Job Until

Job step is deferred until this date and time.

- Cmd** The name of the executable associated with the executable keyword (if specified) or the name of the job command file.
- Args** Arguments that were passed to the executable.
- Env** Environment variables to be set before executable runs. Appears only when the **-x** option is specified.
- In** The file to be used for stdin.
- Out** The file to be used for stdout.
- Err** The file to be used for stderr.

**Initial Working Dir**

The directory from which the job step is run. The relative directory from which the stdio files are accessed, if appropriate.

**Dependency**

Job step dependencies as specified at job submission.

**Requirements**

Job step requirements as specified at job submission.

**Preferences**

Job step preferences as specified at job submission.

**Task\_geometry**

Reflects the settings for the task\_geometry keyword in the job command file.

**Resources**

Reflects the settings for the resources keyword in the job command file.

**Blocking**

Reflects the settings for the blocking keyword in the job command file.

**Step Type**

Type of job step:

- Serial
- General parallel
- PVM3

**Min Processors**

The minimum number of processors needed for this job step.

**Max Processors**

The maximum number of processors that can be used for this job step.

**Allocated Hosts**

The machines that have been allocated for this job step.

**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.

**Submitting Host**

The name of the machine to which the job is submitted.

**Notify User**

The user to be notified by mail of a job's status.

**Shell** The shell to be used when the job step runs.

**LoadLeveler Group**

The LoadLeveler group associated with the job step.

**Class** The class of the job step as specified at job submission.

**Ckpt Hard Limit**

Checkpoint hard limit as specified at job step submission.

**Ckpt Soft Limit**

Checkpoint soft limit as specified at job step submission.

**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.

**Core Hard Limit**

Core hard limit as specified at job submission.

**Core Soft Limit**

Core soft limit as specified at job submission.

**File Hard Limit**

File hard limits as specified at job submission.

**File Soft Limit**

File soft limit as specified at job submission.

**Stack Hard Limit**

Stack hard limit as specified at job submission.

**Stack Soft Limit**

Stack soft limit as specified at job submission.

**Rss Hard Limit**

RSS hard limit as specified at job step submission.

**Rss Soft Limit**

RSS soft limit as specified at job step submission.

**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.

**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.

**NQS Submit Queue**

The name of the NQS pipe queue to which the NQS job will be routed.

**NQS Query Queues**

The NQS queue names you can use to monitor the job.

**Comment**

The comment specified by the comment keyword in the job command file.

**Account**

The account number specified in the job command file.

**Unix Group**

The effective UNIX group name.

**DCE Principal**

The DCE principal name associated with the process that submitted the job to LoadLeveler.

**User Space Windows**

The number of switch adapter windows assigned to the job step.

**Negotiator Messages**

Informational messages for the job step if it is in the Idle or NotQueued state.

**Adapter Requirement**

Reflects the settings of the network keyword in the job command file.

**Step Cpus**

The total ConsumableCpus for the job step.

**Step Virtual Memory**

The total ConsumableVirtualMemory for the job step.

**Step Real Memory**

The total ConsumableMemory for the job step.

**Step Adapter Memory**

The total adapter pinned memory for the job step.

When -x and -l options are specified, **llq** also displays the information listed below. If several LoadL\_starter processes are used for running this job step, then the values reported are either cumulative totals or the maximum values. The same is true for the processes of the job step.

**Starter maxrss/Step maxrss**

Maximum resident set size utilized. Maximum value.

**Starter ixrss/Step ixrss**

Size of the text segment. Maximum value.

**Starter idrss/Step Starter idrss**

Size of the data segment. Maximum value.

**Starter isrss/Step isrss**

Integral unshared stack used. Maximum value.

**Starter minflt/Step minflt**

Number of page faults (reclaimed). Cumulative total.

**Starter majflt/Step majflt**

Number of page faults (I/O required). Cumulative total.

**Starter nswap/Step nswap**

Number of times swapped out. Cumulative total.

**Starter inblock/Step inblock**

Number of times file system performed input. Cumulative total.

**Starter oublock/Step oublock**

Number of times file system performed output. Cumulative total.

**Starter msgsnd/Step msgsnd**

Number of IPC messages sent. Cumulative total.

**Starter msgrcv/Step msgrcv**

Number of IPC messages received. Cumulative total.

**Starter nsignals/Step nsignals**

Number of signals delivered. Cumulative total.

**Starter nvcsww/Step nvcsww**

Number of context switches due to voluntarily giving up processor.  
Cumulative total.

**Starter nivcsww/Step nivcsww**

Number of involuntary context switches. Cumulative total.

**Starter User Time/Step User Time**

CPU user time of Starter/Step processes. 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.

**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.

**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.

Other fields displayed for parallel jobs are:

**(Node) Name**

Blank value. Reserved for future use.

**(Node) Requirements**

Job step requirements as specified at job submission.

**(Node) Preferences**

Job step preferences as specified at job submission.

**(Node) Node minimum**

Minimum number of machines of this Node type required to run 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 actual**

Actual number of machines of this Node type that are used in the running of this job step.

**(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 + ... +  
hostname:task status:adapter usage, ... ,adapter usage
- The adapter usage information has the format: adapter name(adapter window ID, network protocol, mode, adapter window memory)

**(Node/Master Task) Executable**

The executable associated with the master task.

**(Node/Master Task) Exec Args**

The arguments passed to the master task executable.

**(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)

**(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
- Adapter usage information has the format: adapter name(adapter window ID, network protocol, mode, adapter window memory)

## llstatus - Query machine status

### Purpose

Returns status information about machines in the LoadLeveler cluster. It does not provide status on any NQS machine.

### Syntax

```
llstatus [-?] [-H][-R][-F] [-v] [-l] [-f category_list] [-r category_list] [hostlist]
```

### Flags

- ? Provides a short usage message.
- H Provides extended help information.
- R Lists all of the machine consumable resources associated with all of the machines in the LoadLeveler cluster (when specified alone). When a host list is specified, the option only displays machine consumable resources associated with the specified hosts. 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.
- v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
- l Specifies that a long listing be generated for each machine for which status is requested. If -l is *not* specified, the standard list, described below, is generated.
- 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

## llstatus

### **-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.

### *hostlist*

Is a blank-delimited list of machines for which status is requested.

## Description

If no *hostlist* is specified, all machines are queried.

If you have more than a few machines configured for LoadLeveler, consider redirecting the output to a file when using the **-l** flag.

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.

If you define consumable resources in the administration file, then **llstatus** displays this information when either the **-R** or **-I** option is specified. For the predefined ConsumableCpus resource, the "total" values reported by **llstatus** can be the values defined in the administration file or the values evaluated by the startd daemons. The startd values are used if the administration file values are set to "all." In this case, **llstatus** appends a plus (+) sign to the resource name in the output reports.

## Examples

This example requests a long status listing for machines named silver and gold:

```
llstatus -l silver gold
```

## Results

**The Standard Listing:** The standard listing is generated when you do *not* specify the **-l** option with the **llstatus** command. The following is sample output from the **llstatus** command, where there are two nodes in the cluster.

Name	Schedd	InQ	Act	Startd	Run	LdAvg	Idle	Arch	OpSys
k10n09.ppd.pok.ibm.com	Avail	3	1	Run	1	2.72	0	R6000	AIX51
k10n12.ppd.pok.ibm.com	Avail	0	0	Idle	0	0.00	365	R6000	AIX51
R6000/AIX51	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 GANG scheduler is in use

All machines on the machine\_list are present.

The standard listing includes the following fields:



**Name** Hostname of the machine.

**Schedd**

State of the schedd daemon, which can be one of the following:

Down  
 Drned (Drained)  
 Drning (Draining)  
 Avail (Available)

For a detailed explanation of these states, see “The schedd daemon” on page 129.

**InQ** Number of job steps in the job queue of this schedd machine.

**Act** Number of job steps dispatched by the schedd daemon on this machine.

**Startd** State of the startd daemon, which can be:

Busy  
 Down  
 Drned (Drained)  
 Drning (Draining)  
 Flush  
 Idle  
 None  
 Run (Running)  
 Suspnd (Suspend)

For a detailed explanation of these states, see “The startd daemon” on page 130.

**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.

**LdAvg**

Berkeley one-minute load average on this machine.

**Idle** The number of seconds since keyboard or mouse activity in a login session was detected. Highest number displayed is 9999.

**Arch** The hardware architecture of the machine as listed in the configuration file.

**OpSys**

The operating system on this machine.

**Total Machines**

The standard listing includes the following summary fields:

**machines**

The number of machines in the cluster that have made a status report to the Central Manager.

**jobs** The number of job steps in LoadLeveler job queues.

**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.

**Consumable Resources Listing:** The **llstatus** command, issued with the **-R** option, generates a listing of all of the consumable resources associated with all of the machines in the LoadLeveler cluster. When a host list is specified, this option will only display resources associated with the specified hosts. The following is sample output from this command:

## llstatus

### llstatus -R

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(4,4)+ 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.

Figure 23. Sample llstatus -R command output

**Floating Consumable Resources Listing:** The **llstatus** command, issued with the **-F** option, 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. The following is sample output from this command:

### llstatus -F

Floating Resource	Available	Total
EDA_licenses	20	29
Frame5	15	20
WorkBench6	5	7
XYZ_software	6	6

**Customized, Formatted Standard Listing:** A customized and formatted standard listing is generated when you specify **llstatus** with the **-f** option. The following is sample output from this command:

### llstatus -f %n %scs %inq %m %v %sts %l %o

Name	Schedd	InQ	Memory	FreeVMemory	Startd	LdAvg	OpSys
l15.pok.ibm.com	Avail	0	128	22708	Run	0.23	AIX51
l16.pok.ibm.com	Avail	3	224	16732	Run	0.51	AIX51
R6000/AIX51		2 machines	3 jobs	3 running			
Total Machines		2 machines	3 jobs	3 running			

The Central Manager is defined on l15.pok.ibm.com

The GANG scheduler is in use

All machines on the machine\_list are present.

**Customized, Unformatted Standard Listing:** A customized and unformatted (raw) standard listing is generated when you specify **llstatus** with the **-r** flag. Output fields are separated by an exclamation point (!). The following is sample output from this command:

### llstatus -r %n %scs %inq %m %v %sts %l %o

l15.pok.ibm.com!Avail!0!128!22688!Running!0.14!AIX51
l16.pok.ibm.com!Avail!3!224!16668!Running!0.37!AIX51

**The Long Listing:** The long listing is generated when you specify the **-l** option with the **llstatus** command. Following the sample output is an explanation of all possible fields displayed by the **llstatus** command.

The following is sample output from the **llstatus -l c209f1n05** command:

```
=====
Name                = c209f1n05.ppd.pok.ibm.com
Machine             = c209f1n05.ppd.pok.ibm.com
Arch                = R6000
OpSys               = AIX51
SYSPRIO             = (0 - QDate)
MACHPRIO            = ((Memory + FreeRealMemory) - ((LoadAvg * 1000) + CustomMetric))
VirtualMemory       = 491560 kb
Disk                = 519484 kb
KeyboardIdle        = 0
Tmp                 = 519484 kb
LoadAvg             = 1.802475
ConfiguredClasses   = Parallel(12) 85ba(2) misc(2) tiny(1) No_Class(7) small(14) large(1) medium(1)
AvailableClasses    = Parallel(8) 85ba(2) misc(2) tiny(1) No_Class(7) small(14) medium(1)
DrainingClasses     =
DrainedClasses      =
Pool                = 1 7
FabricConnectivity  = 1
Adapter             = en0(ethernet,c209f1n05.ppd.pok.ibm.com,9.114.99.66,)
                   css0(switch,c209f1n05.ppd.pok.ibm.com,9.114.99.130,,1,8/16,120M/128M,1,READY)
                   csss(striped,,,,1,8/16,120M/128M,1,READY)
Feature             = OSL ESSL
Max_Starters        = 50
Memory              = 1024 mb
FreeRealMemory      = 484 mb
PagesFreed          = 0
PagesScanned        = 0
PagesPagedIn        = 0
PagesPagedOut       = 0
ConsumableResources = ConsumableCpus(0,4)+ ConsumableMemory(724.000 mb,1.000 gb) spice2g6(360,360)
ConfigTimeStamp     = Fri Jul 27 11:44:29 EDT 2001
```

Figure 24. Sample output from **llstatus -l c209f1n05** (Part 1 of 2)

## llstatus

```
Cpus                = 4
Speed              = 1.000000
Subnet             = 9.114.99
MasterMachPriority  = 0.000000
CustomMetric       = 1
StartdAvail        = 1
State              = Running
EnteredCurrentState = Fri Jul 27 11:46:22 EDT 2001
START              = ((LoadAvg < 5.000000) && ((tm_hour > 8) && (tm_hour < 17)))
SUSPEND            = F
CONTINUE           = T
VACATE             = F
KILL               = F
Machine Mode       = general
Running            = 5
ScheddAvail        = 1
ScheddState        = Avail
ScheddRunning      = 2
Pending            = 0
Starting           = 0
Idle               = 5
Unexpanded         = 0
Held               = 0
Removed            = 0
RemovedPending     = 0
Completed          = 0
TotalJobs          = 7
Running steps      = c209f1n05.ppd.pok.ibm.com.5.0 c209f1n05.ppd.pok.ibm.com.6.0
                   c209f1n05.ppd.pok.ibm.com.7.0
TimeStamp          = Fri Jul 27 11:46:22 EDT 2001
```

Figure 24. Sample output from `llstatus -l c209f1n05` (Part 2 of 2)

The long listing includes these 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,  
available\_adapter\_windows/total\_adapter\_windows,  
available\_device\_memory/total\_device\_memory,  
adapter\_fabric\_connectivity, adapter\_state)
- 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 machines. Each element of this list has the format: resource\_name(available, total).

**CONTINUE**

The expression, defined following C conventions in the configuration file, that evaluates to true or false (T/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**

A boolean vector representing the current state of connectivity between machine's switch adapters and the SP switch.

**Feature**

Set of all features on this machine.

**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/F). This determines whether jobs running on this machine should be sent the SIGKILL signal.

**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.

## llstatus

### **MasterMachPriority**

The machine priority for the parallel master node.

### **Max\_Starters**

Maximum number of initiators that can be used simultaneously on this machine.

### **Memory**

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.

### **PagesPaged In**

Pages paged in from paging space per second. This value corresponds to the "pi" value of the vmstat command output.

### **PagesPagedOut**

Pages paged out to paging space per second. This value corresponds to the "po" value of the vmstat command output.

### **PagesScanned**

Pages scanned by the page-replacement algorithm per second. This value corresponds to the "sr" value of the vmstat command output.

### **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.

### **Removed**

The number of job steps in this state on this schedd machine.

### **Remove Pending**

The number of job steps in this state on this schedd machine.

### **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.

**Speed** Speed associated with the machine.

**START**

The expression, defined following C conventions in the configuration file, that evaluates to true or false (T/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 a detailed explanation of these states, see “The startd daemon” on page 130.

**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/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.

**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/F). This determines whether suspended jobs are vacated on this machine.

**Virtual Memory**

Available swap space (free paging space) in kilobytes, on this machine.

## lsubmit - Submit a job

### Purpose

Submits a job to LoadLeveler to be dispatched based upon job requirements in the job command file.

You can submit both LoadLeveler jobs and NQS jobs. To submit NQS jobs, the job command file must contain the shell script to be submitted to the NQS node.

### Syntax

```
lsubmit [-H] [-?] [-v] [-q] [<cmdfile> | - ]
```

### Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
- q Specifies quiet mode: print no messages other than error messages.

#### *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.

### Related Information

- Users with **uid** or **gid** equal to 0 are not allowed to issue the **lsubmit** 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 information on writing a program to filter job scripts when they are submitted, see "Filtering a job script" on page 285.
- The **lsubmit** command will display an error and fail to submit the job if the **resources** keyword in the job command file doesn't match the resources to be enforced and LoadLeveler is set to check for the **resources** specification. For more details on assigning and enforcing consumable resources, see "Step 4: Define consumable resources" on page 341.

### Examples

In this example, a job command file named *qtrlyrun.cmd* is submitted:

```
lsubmit qtrlyrun.cmd
```

### Results

The following shows the results of the **lsubmit qtrlyrun.cmd** command issued from the machine **earth**:



llsubmit: The job "earth.505" has been submitted.

Note that 505 is the job ID generated by LoadLeveler.

## lsummary - Return job resource information for accounting

### Purpose

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 more details, refer to "Step 9: Define job accounting" on page 349.

### Syntax

```
lsummary [-?] [-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>]
[<filename>]
```

### Flags

- ? Provides a short usage message.
- H Provides extended help information.
- v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
- 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), lsummary will report information about every job which contains at least one step of the specified class. For the short format, lsummary 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), lsummary reports information about every job which contains at least one step of the specified group. For the

## lsummary

short format, **lsummary** 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:

#### **resource**

Provides CPU usage for all submitted jobs, including those that did not run. This is the default.

#### **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.

#### **throughput**

Selects all throughput reports.

#### **numeric**

Reports CPU times in seconds rather than hours, minutes, and seconds

### **-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*

The job for which accounting data is reported. *host* is the name of the machine to which the job was submitted. The default is the local machine. *jobid* is the job ID assigned to the job when it was submitted using the **lsubmit** command. The entire *host.jobid* string is required.

### *filename*

The file containing the accounting data. If not specified, the default is the local history file on the machine from which the command was issued. You can use the **llacctmrg** command to produce such a file.

## Related Information

In order to create an accounting report with the **lsummary** command, you must have read access to a history file. If a history file name is not specified as an argument, **lsummary** 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

## lsummary

the **linit** 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

The following example requests summary reports (standard listing) of all the jobs submitted on your machine between the days of September 12, 1999 and October 12, 1999:

```
lsummary -s 09/12/1999 to 10/12/1999
```

## Results

**The Standard Listing:** The standard listing is generated when you do not specify **-l**, **-r**, or **-d** with **lsummary**. 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.

The following is an example of the standard listing:

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
osl2	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:

**Name** User ID submitting jobs.

**Class** Class specified or defaulted for the jobs.

**Group** User's login group.

**Account**

Account number specified for the jobs.

**Jobs** Count of the total number of jobs submitted by this user, class, group, or account.

**Steps** Count of the total number of job steps submitted by this user, class, group, or account.

## Job CPU

Total CPU time consumed by user's jobs.

## Starter CPU

Total CPU time consumed by LoadLeveler starter processes on behalf of the user jobs.

## Leverage

Ratio of job CPU to starter CPU.

**The -r Listing:** The following is sample output from the **llsummary -r throughput** command. 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:

## AvgQueueTime

Average amount of time the job spent queued before running for this user, class, group, or account.

## AvgRealTime

Average amount of accumulated wall clock time for jobs associated with this user, class, group, or account.

## AvgCPUTime

Average amount of accumulated CPU time for jobs associated with 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.

## MinCPUime

Time of the job with the least amount of CPU time for this user, class, group, or account.

The **MaxQueueTime**, **MaxRealTime**, and **MaxCPUTime** fields display the time of the job with the greatest amount of queue, wall clock, and CPU time, respectively. The **ALL** line for the Average listing displays the average time for all users, classes,

## lsummary

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.

**The Long Listing:** If you specify both the **-x** and **-l** options when running the **lsummary** command, the generated output you receive will resemble the listing below. In this sample, c209f1n05.ppd.pok.ibm.com.2 is a parallel job consisting of one job step with four tasks. Two of these tasks are running on c209f1n05 and two tasks are running on c209f1n01.

**Note:** Job statistics for the running hosts are available separately.

In this sample, you will find system defined events named started and completed. The listing also shows two installation defined events named user\_event\_1 and user\_event\_2. These two installation defined events are the result of accounting snapshots made while the job was running. A LoadLeveler administrator made the snapshots by issuing the commands, **llctl -g capture user\_event\_1** and **llctl -g capture user\_event\_2**.

```
===== Job c209f1n05.ppd.pok.ibm.com 2 =====
      Job Id: c209f1n05.ppd.pok.ibm.com 2
      Job Name: c209f1n05.ppd.pok.ibm.com.2
      Structure Version: 210
      Owner: loadl
      Unix Group: loadl
      Submitting Host: c209f1n05.ppd.pok.ibm.com
      Submitting Userid: 1064
      Submitting Groupid: 222
      Number of Steps: 1
----- Step c209f1n05.ppd.pok.ibm.com.2.0 -----
      Job Step Id: c209f1n05.ppd.pok.ibm.com.2.0
      Step Name: parallel_job_step_1
      Queue Date: Fri Jul 27 17:52:59 EDT 2001
      Dependency:
      Status: Completed
      Dispatch Time: Fri Jul 27 17:52:59 EDT 2001
      Start Time: Fri Jul 27 17:53:00 EDT 2001
      Completion Date: Fri Jul 27 17:55:12 EDT 2001
      Completion Code: 0
      Start Count: 1
      User Priority: 50
      user_sysprio: 50
      class_sysprio: 45
      group_sysprio: 0
      Notifications: Complete
      Virtual Image Size: 800 kb
      Checkpointable: no
      Good Ckpt Time/Date:
      Ckpt Accum Time: 0
      Checkpoint File:
```

Figure 25. Output generated by **lsummary -x -l** command (Part 1 of 8)

```

Restart From Ckpt: no
Restart Same Nodes: no
  Restart: yes
  Hold Job Until:
    Cmd: /bin/poe
    Args: /tmp/PARA_50/ivp_cpu_60_60_sleep_60 -eulib us -ilevel 6 -labelio yes -pmdlog yes
    Env: MANPATH=/usr/local/man:/usr/share/man:/usr/lpp/LoadL/full/man; LANG=en_US; ...
    In: /dev/null
    Out: poe5_1.c209f1n05.2.0.out
    Err: poe5_1.c209f1n05.2.0.err
Initial Working Dir: /tmp/PARA_50
  Requirements: (Arch == "R6000") && (OpSys == "AIX51") && (Memory > 128)
  Preferences: (Machine == { "c209f1n01.ppd.pok.ibm.com" "c209f1n05.ppd.pok.ibm.com" }) && (Feature == "ESSL")
  Step Type: General Parallel
  Min Processors: 2
  Max Processors: 2
Alloc. Host Count: 2
  Allocated Host: c209f1n05.ppd.pok.ibm.com
                  c209f1n01.ppd.pok.ibm.com
  Node Usage: shared
  Notify User: loadl
  Shell: /bin/ksh
LoadLeveler Group: No_Group
  Class: Parallel
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: 4.250 pb (4785074604081152 bytes)
Data Soft Limit: 1.500 tb (1649267441664 bytes)
Core Hard Limit: 2.250 tb (2473901162496 bytes)
Core Soft Limit: 1.250 tb (1374389534720 bytes)
File Hard Limit: 1.200 eb (1383505805528216384 bytes)
File Soft Limit: 1.100 eb (1268213655067531680 bytes)

```

Figure 25. Output generated by `llsummary -x -l` command (Part 2 of 8)

## llsummary

```
Stack Hard Limit: 40.000 mb (41943040 bytes)
Stack Soft Limit: 30.000 mb (31457280 bytes)
Rss Hard Limit: 1.200 eb (1383505805528216384 bytes)
Rss Soft Limit: 5.500 pb (6192449487634432 bytes)
Step Cpu Hard Limit: 3+08:00:00 (288000 seconds)
Step Cpu Soft Limit: 23:59:59 (86399 seconds)
Wall Clk Hard Limit: 01:40:00 (6000 seconds)
Wall Clk Soft Limit: 01:40:00 (6000 seconds)
Comment: Test job 3 of Parallel test suite 6.
Account: 99999
NQS Submit Queue:
NQS Query Queues:
Job Tracking Exit:
Job Tracking Args:
Task geometry:
Resources: ConsumableMemory(75 bytes) ConsumableCpus(1)
Blocking: UNSPECIFIED
Adapter Requirement:
Step Cpus: 4
Step Virtual Memory: 0.000 mb
Step Real Memory: 300.000 mb
Step Adapter Memory: 8.000 mb (8388608 bytes)
----- Detail for c209f1n05.ppd.pok.ibm.com.2.0 -----
Running Host: c209f1n05.ppd.pok.ibm.com
Machine Speed: 1.000000
Event: System
Event Name: started
Time of Event: Fri Jul 27 17:53:00 EDT 2001
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
...
...
```

Figure 25. Output generated by llsummary -x -l command (Part 3 of 8)



```

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 nivcsw: 0
Event: Installation Defined
Event Name: user_event_1
Time of Event: Fri Jul 27 17:53:29 EDT 2001
Starter User Time: 00:00:00.110000
Starter System Time: 00:00:00.140000
Starter Total Time: 00:00:00.250000
Starter maxrss: 2000
Starter ixrss: 9324
...
...
Starter nivcsw: 29
Step User Time: 00:00:49.920000
Step System Time: 00:00:00.420000
Step Total Time: 00:00:50.340000
Step maxrss: 4312
Step ixrss: 59888
...
...
Step nivcsw: 5235
Event: Installation Defined
Event Name: user_event_2
Time of Event: Fri Jul 27 17:53:55 EDT 2001
Starter User Time: 00:00:00.110000
Starter System Time: 00:00:00.140000
Starter Total Time: 00:00:00.250000
Starter maxrss: 2000
Starter ixrss: 9324
...
...

```

Figure 25. Output generated by llsummary -x -l command (Part 4 of 8)

## llsummary

```
Starter nivcsw: 29
Step User Time: 00:01:41.940000
Step System Time: 00:00:00.450000
Step Total Time: 00:01:42.390000
  Step maxrss: 4312
  Step ixrss: 122408
  ...
  ...
  Step nivcsw: 10692
  Event: System
    Event Name: completed
    Time of Event: Fri Jul 27 17:55:11 EDT 2001
Starter User Time: 00:00:00.110000
Starter System Time: 00:00:00.150000
Starter Total Time: 00:00:00.260000
  Starter maxrss: 2000
  Starter ixrss: 9324
  Starter idrss: 16812
  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 nvcs: 52
  Starter nivcsw: 30
  Step User Time: 00:02:00.310000
  Step System Time: 00:00:00.580000
  Step Total Time: 00:02:00.890000
```

Figure 25. Output generated by llsummary -x -l command (Part 5 of 8)

```

Step maxrss: 4312
Step ixrss: 288516
Step idrss: 51400928
Step isrss: 0
Step minflt: 4919
Step majflt: 4
Step nswap: 0
Step inblock: 0
Step oublock: 0
Step msgsnd: 0
Step msgrcv: 0
Step nsignals: 2
Step nvcs: 899
Step nivcs: 12634
Running Host: c209f1n01.ppd.pok.ibm.com
Machine Speed: 1.000000
Event: System
Event Name: started
Time of Event: Fri Jul 27 17:53:00 EDT 2001
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 nivcs: 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
...
...

```

Figure 25. Output generated by `llsummary -x -l` command (Part 6 of 8)

## lsummary

```
Step nivcsw: 0
  Event: Installation Defined
    Event Name: user_event_1
    Time of Event: Fri Jul 27 17:53:29 EDT 2001
  Starter User Time: 00:00:00.120000
  Starter System Time: 00:00:00.070000
  Starter Total Time: 00:00:00.190000
    Starter maxrss: 1940
    Starter ixrss: 8432
    ...
    ...
  Starter nivcsw: 25
  Step User Time: 00:00:50.110000
  Step System Time: 00:00:00.250000
  Step Total Time: 00:00:50.360000
    Step maxrss: 4288
    Step ixrss: 40368
    ...
    ...
  Step nivcsw: 5455
  Event: Installation Defined
    Event Name: user_event_2
    Time of Event: Fri Jul 27 17:53:55 EDT 2001
  Starter User Time: 00:00:00.120000
  Starter System Time: 00:00:00.070000
  Starter Total Time: 00:00:00.190000
    Starter maxrss: 1940
    Starter ixrss: 8432
    ...
    ...
  Starter nivcsw: 25
  Step User Time: 00:01:42.280000
  Step System Time: 00:00:00.250000
  Step Total Time: 00:01:42.530000
    Step maxrss: 4288
    Step ixrss: 82128
    ...
    ...
```

Figure 25. Output generated by `lsummary -x -l` command (Part 7 of 8)

```

Step nivcsw: 10878
  Event: System
  Event Name: completed
  Time of Event: Fri Jul 27 17:55:12 EDT 2001
Starter User Time: 00:00:00.120000
Starter System Time: 00:00:00.070000
Starter Total Time: 00:00:00.190000
  Starter maxrss: 2004
  Starter ixrss: 8432
  Starter idrss: 12428
  Starter isrss: 0
  Starter minflt: 606
  Starter majflt: 0
  Starter nswap: 0
  Starter inblock: 0
  Starter oublock: 0
  Starter msgsnd: 0
  Starter msgrcv: 0
  Starter nsignals: 2
  Starter nivcsw: 40
  Starter nivcsw: 25
Step User Time: 00:02:00.420000
Step System Time: 00:00:00.330000
Step Total Time: 00:02:00.750000
  Step maxrss: 4292
  Step ixrss: 196376
  Step idrss: 51428956
  Step isrss: 0
  Step minflt: 3511
  Step majflt: 0
  Step nswap: 0
  Step inblock: 0
  Step oublock: 0
  Step msgsnd: 0
  Step msgrcv: 0
  Step nsignals: 2
  Step nivcsw: 771
  Step nivcsw: 12823

```

Figure 25. Output generated by `llsummary -x -l` command (Part 8 of 8)

For an explanation of these fields, see the description of the output fields for the long listing of the **llq** command.



---

## Chapter 15. Application Programming Interfaces (APIs)

LoadLeveler provides several Application Programming Interfaces (API) that you can use. LoadLeveler's APIs allow application programs written by customers to interact with the LoadLeveler environment by using specific data or functions that are a part of LoadLeveler. These interfaces can be subroutines within a library or installation exits.

This appendix provides details on the following APIs, their subroutines, and required keywords:

- "Accounting API"
- "Error Handling API" on page 259
- "Checkpointing API" on page 218
- "Submit API" on page 268
- "Data Access API" on page 223
- "Parallel Job API" on page 260
- "Workload Management API" on page 270
- "Query API" on page 265
- "User exits" on page 282

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** 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.

---

### Accounting API

The LoadLeveler Accounting API provides a user exit for account validation and a subroutine for extracting accounting data. Job accounting information saved in a history file can also be queried by using the Data Access API.

#### Account validation user exit

LoadLeveler provides the **llacctval** executable to perform account validation.

##### Purpose

**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 non-zero 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

## Accounting API

**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 non-zero, **llsubmit** does not submit the job.

You can replace **llacctval** with your own accounting user exit (see below).

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 non-zero number.

## Report generation subroutine

LoadLeveler provides the **GetHistory** subroutine to generate accounting reports.

### Purpose

**GetHistory** processes local or global LoadLeveler history files.

### Library

LoadLeveler API library **libllapi.a**

### 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

**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.

## Checkpointing API

This section describes routines used for checkpointing jobs running under LoadLeveler. "Step 14: Enable checkpointing" on page 356 describes how to checkpoint your jobs in various ways. 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\_init\_ckpt

#### Purpose

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 environment variable **LOADL\_CKPT\_FILE** 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 environment variable **LOADL\_STEP\_ID**
- 2 Cannot retrieve the checkpoint file name from the environment variable **LOADL\_CKPT\_FILE**, 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\_ckpt

**Note:** Before you consider using the Checkpoint/Restart function refer to the LoadL.README file in /usr/lpp/LoadL/READMEs for information on availability and support of this function.

## Purpose

Initiates a checkpoint on a specific job step.

## Library

LoadLeveler API library **libllapi.a**

## 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*

The id of the job step to be checkpointed. Uses the following formats: "*host.jobid.stepid*," "*jobid.stepid*". Where:

- *host*: is the name of the machine to which the job was submitted (the default is the local machine)
- *jobid*: is the job ID assigned to the job by LoadLeveler
- *stepid*: is the job step ID assigned to a job step by LoadLeveler

**enum ckpt\_type** *ckptType*

The action to be taken after the checkpoint successfully completes. The values for **enum ckpt\_type** are:

**CKPT\_AND\_CONTINUE**

Allow the job to continue after the checkpoint

**CKPT\_AND\_TERMINATE**

Terminate the job after the checkpoint

**CKPT\_AND\_HOLD**

Puts the job on user hold after the checkpoint

**Note:** If checkpoint is not successful, the job continues on return regardless of these settings.

**enum wait\_option** *waitType*

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**

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*

AIX structure containing error info from **ckpt**.

**int** *ckpt\_rc*

Return code from checkpoint

**int** *soft\_limit*

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***

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.

## 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
- 5** 64-bit API not supported when DCE is enabled
- 6** Configuration file errors
- 7** DCE identity cannot be established
- 8** No DCE credentials
- 9** DCE credentials life time less than 300 seconds

## ll\_set\_ckpt\_callbacks

### Purpose

Used by an application to register callbacks which will be invoked when a job step is checkpointed, resumed and restarted.

## ll\_set\_ckpt\_callbacks

### 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 non-negative 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

### Purpose

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.

---

## Data Access API

This API gives you access to LoadLeveler objects and allows you to retrieve specific data from those objects. You can use this API to query the negotiator daemon for information about its current set of jobs, machines, and gang matrices. This API can also be used to:

- Query a LoadLeveler history file for job accounting information
- Query the startd and schedd daemons for selected Workload Manager and job information

The Data Access API consists of the following subroutines: **ll\_query**, **ll\_set\_request**, **ll\_reset\_request**, **ll\_get\_objs**, **ll\_get\_data**, **ll\_next\_obj**, **ll\_free\_objs**, and **ll\_deallocate**.

## Using the data access API

To use this API, you need to call the data access subroutines in the following order:

- Call **ll\_query** to initialize the query object. See “ll\_query subroutine” on page 224 for more information.
- Call **ll\_set\_request** to filter the objects you want to query. See “ll\_set\_request subroutine” on page 224 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 228 for more information.
    - Call **ll\_get\_data** to retrieve specific data from an object. See “ll\_get\_data subroutine” on page 233 for more information.
  - Call **ll\_next\_obj** to retrieve the next object in the list. See “ll\_next\_obj subroutine” on page 251 for more information.
- Call **ll\_free\_objs** to free the list of objects you received. See “ll\_free\_objs subroutine” on page 251 for more information.
- Call **ll\_deallocate** to end the query. See “ll\_deallocate subroutine” on page 252 for more information.

To see code that uses these subroutines, refer to “Examples of using the Data Access API” on page 252. For more information on LoadLeveler objects, see “Understanding the LoadLeveler job object model” on page 230.

### ll\_query subroutine

#### Purpose

The **ll\_query** subroutine 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**

#### Syntax

```
#include "llapi.h"
```

```
LL_element * ll_query(enum QueryType query_type);
```

#### Parameters

*query\_type*

Can be:

- **JOBS** (to query job information)
- **MACHINES** (to query machine information)
- **CLUSTER** (to query cluster information)
- **WLMSTAT** (to query AIX Workload Manager)
- **MATRIX** (to query Gang Matrix scheduling information)

#### 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\_get\_data**, **ll\_set\_request**, **ll\_reset\_request**, **ll\_get\_objs**, **ll\_free\_objs**, **ll\_next\_obj**, **ll\_deallocate**.

### ll\_set\_request subroutine

#### Purpose

The **ll\_set\_request** subroutine 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**

#### 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*

When *query\_type* (in **ll\_query**) is **JOBS**, *query\_flags* can be the following:

#### **QUERY\_ALL**

Query all jobs.

#### **QUERY\_JOBID**

Query by job ID.

#### **QUERY\_STEPID**

Query by step ID.

#### **QUERY\_USER**

Query by user ID.

#### **QUERY\_GROUP**

Query by LoadLeveler group.

#### **QUERY\_CLASS**

Query by LoadLeveler class.

#### **QUERY\_HOST**

Query by machine name.

#### **QUERY\_STARTDATE**

Query by job start dates. History file query only.

#### **QUERY\_ENDDATE**

Query by job end dates. History file query only.

When *query\_type* (in **ll\_query**) is **MACHINES**, *query\_flags* can be the following:

#### **QUERY\_ALL**

Query all machines.

#### **QUERY\_HOST**

Query by machine names.

When *query\_type* (in **ll\_query**) is **CLUSTER**, *query\_flags* can be the following:

#### **QUERY\_ALL**

Query cluster information from central manager.

When *query\_type* (in **ll\_query**) is **WLMSTAT**, *query\_flags* can be the following:

#### **QUERY\_STEPID**

Query by step ID.

When *query\_type* (in **ll\_query**) is **MATRIX**, *query\_flags* can be the following:

#### **QUERY\_ALL**

Query all machines.

#### **QUERY\_HOST**

Query by machine names.

### *object\_filter*

Specifies search criteria. The value you specify for *object\_filter* is related to the value you specify for *query\_flags*:

- If you specify **QUERY\_ALL**, you do not need an *object\_filter*.
- If you specify **QUERY\_JOBID**, the *object\_filter* must contain a list of job IDs (in the form *host.jobid*).
- If you specify **QUERY\_STEPID**, the *object\_filter* must contain a list of step IDs (in the form *host.jobid.stepid*).
- If you specify **QUERY\_USER**, the *object\_filter* must contain a list of user IDs.

## Data Access API

- If you specify **QUERY\_CLASS**, the *object\_filter* must contain a list of LoadLeveler class names.
- If you specify **QUERY\_GROUP**, the *object\_filter* must contain a list of LoadLeveler group names.
- If you specify **QUERY\_HOST**, the *object\_filter* 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.
- If you specify **QUERY\_STARTDATE** or **QUERY\_ENDDATE**, the object filter must contain a list of two start dates or two end dates having the format MM/DD/YYYY.

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*:

- If you specify **JOBS**, *data\_filter* can be **ALL\_DATA** (the default), which returns the entire object, or **Q\_LINE**, which returns the same information returned by the **llq -f** flag. For more information, see “llq - Query job status” on page 173.

**Note:** If you query a history file for job information, always specify **ALL\_DATA**.

- If you specify **MACHINES**, *data\_filter* can be **ALL\_DATA** (the default) which returns the entire object, or **STATUS\_LINE** which returns the same information returned by the **llstatus -f** flag. For more information, see “llstatus - Query machine status” on page 191.
- If you specify **MATRIX**, **WLM\_STAT**, or **CLUSTER**, then *data\_filter* must be **ALL\_DATA** (the default).

## Description

*query\_element*, *query\_flags*, *object\_filter*, and *data\_filter* are the input fields for this subroutine.

You can request a combination of object filters by calling **ll\_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
- **QUERY\_HOST** and **QUERY\_USER**: the result is the intersection of both queries
- **QUERY\_HOST** and **QUERY\_CLASS**: the result is the intersection of both queries
- **QUERY\_HOST** and **QUERY\_GROUP**: the result is the intersection of both queries
- **QUERY\_STARTDATE** and **QUERY\_ENDDATE**: the result is the intersection of both queries.

That is, to query jobs owned by certain users and on a 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\_get\_data**, **ll\_query**, **ll\_reset\_request**, **ll\_get\_objs**, **ll\_free\_objs**, **ll\_next\_obj**, **ll\_deallocate**.

## ll\_reset\_request subroutine

### Purpose

The **ll\_reset\_request** subroutine resets the request data to NULL for the *query\_element* you specify.

### Library

LoadLeveler API library **libllapi.a**

### 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\_get\_data**, **ll\_set\_request**, **ll\_query**, **ll\_get\_objs**, **ll\_free\_objs**, **ll\_next\_obj**, **ll\_deallocate**.

## ll\_get\_objs subroutine

### Purpose

The **ll\_get\_objs** subroutine 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**

### 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};
```

The following indicates which daemons respond to which query flags. When *query\_type* (in **ll\_query**) is **JOBS**, the *query\_flags* (in **ll\_set\_request**) listed in the left-hand column are responded to by the daemons listed in the right-hand column:

<b>QUERY_ALL</b>	negotiator (LL_CM), schedd (LL_SCHEDD), or history file (LL_HISTORY_FILE)
<b>QUERY_JOBID</b>	negotiator (LL_CM) or schedd (LL_SCHEDD)
<b>QUERY_STEPIID</b>	negotiator (LL_CM)
<b>QUERY_USER</b>	negotiator (LL_CM)
<b>QUERY_GROUP</b>	negotiator (LL_CM)
<b>QUERY_CLASS</b>	negotiator (LL_CM)
<b>QUERY_HOST</b>	negotiator (LL_CM)
<b>QUERY_STARTDATE</b>	history file (LL_HISTORY_FILE)
<b>QUERY_ENDDATE</b>	history file (LL_HISTORY_FILE)

When *query\_type* (in **ll\_query**) is **MACHINES**, the *query\_flags* (in **ll\_set\_request**) listed in the left-hand column are responded to by the daemons listed in the right-hand column:

<b>QUERY_ALL</b>	negotiator (LL_CM)
<b>QUERY_HOST</b>	negotiator (LL_CM)

When *query\_type* (in **ll\_query**) is **CLUSTER**, the *query\_flags* (in **ll\_set\_request**) listed in the left-hand column are responded to by the daemons listed in the right-hand column:

QUERY_ALL	negotiator (LL_CM)
-----------	--------------------

When *query\_type* (in **ll\_query**) is **WLMSTAT**, the *query\_flags* (in **ll\_set\_request**) listed in the left-hand column are responded to by the daemons listed in the right-hand column:

QUERY_STEPID	startd (LL_STARTD)
--------------	--------------------

When *query\_type* (in **ll\_query**) is **MATRIX**, the *query\_flags* (in **ll\_set\_request**) listed in the left-hand column are responded to by the daemons listed in the right-hand column:

QUERY_ALL	negotiator (LL_CM)
QUERY_HOST	negotiator (LL_CM)

#### *hostname*

Specifies the *hostname* where the **schedd** or **startd** daemon is queried. If you specify NULL, the **schedd** daemon on the local machine is queried. To contact the negotiator daemon, you do not need to specify a *hostname*. If *query\_daemon* is LL\_HISTORY\_FILE, *hostname* is the name of the history file.

#### *number\_of\_objs*

Is a pointer to an integer representing the number of objects received from the daemon.

**Note:** For **MATRIX** queries:

- Only one object is returned when the call is successful
- **ll\_next\_obj** is not needed

#### *error\_code*

Is a pointer to an integer representing the error code issued when the function returns a NULL value. 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)

## Data Access API

- 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)
- 13 DCE identity of calling program can not be established
- 14 No DCE credentials
- 15 DCE credentials within 300 secs of expiration
- 16 64-bit API is not supported when DCE is enabled

### Related information

Subroutines: **ll\_get\_data**, **ll\_set\_request**, **ll\_query**, **ll\_get\_objs**, **ll\_free\_objs**, **ll\_next\_obj**, **ll\_deallocate**.

## Understanding the LoadLeveler job object model

The **ll\_get\_data** subroutine of the data access API allows you to access the LoadLeveler job model. The LoadLeveler job model consists of objects that have attributes and connections to other objects. An attribute is a characteristic of the object and generally has a primitive data type (such as integer, float, or character). The job name, submission time and job priority are examples of attributes.

Objects are connected to one or more other objects via relationships. An object can be connected to other objects through more than one relationship, or through the same relationship. For example, A Job object is connected to a Credential object and to Step objects through two different relationships. A Job object can be connected to more than one Step object through the same relationship of "having a Step." When an object is connected through different relationships, different specifications are used to retrieve the appropriate object.

When an object is connected to more than one object through the same relationship, there are Count, GetFirst and GetNext specifications associated with the relationship. The Count operation returns the number of connections. You must use the GetFirst operation to initialize access to the first such connected object. You must use the GetNext operation to get the remaining objects in succession. You can not use GetNext after the last object has been retrieved.

You can use the **ll\_get\_data** subroutine to access both attributes and connected objects. See "ll\_get\_data subroutine" on page 233 for more information.

The root of the job model is the Job object, as shown in Figure 26 on page 232. The job is queried for information about the number of steps it contains and the time it was submitted. The job is connected to a single Credential object and one or more Step objects. Elements for these objects can be obtained from the job.

You can query the Credential object to obtain the ID and group of the submitter of the job.

The Step object represents one executable unit of the job (all the tasks that are executed together). It contains information about the execution state of the step, messages generated during execution of the step, the number of nodes in the step, the number of unique machines the step is running on, the time the step was dispatched, the execution priority of the step, the unique identifier given to the step by LoadLeveler, the class of the step and the number of processes running for the step (task instances). The Step is connected to one or more Switch Table objects, one or more Machine objects and one or more Node objects. The list of Machines represents all of the hosts where one or more nodes of the step are running. 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 Step object is connected to one Switch Table object for each of the protocols (MPI and/or LAPI) used by the Step.

Each Node object manages a set of executables that share common requirements and preferences. The Node can be queried for the number of tasks it manages, and is connected to one or more Task objects.

## Data Access API

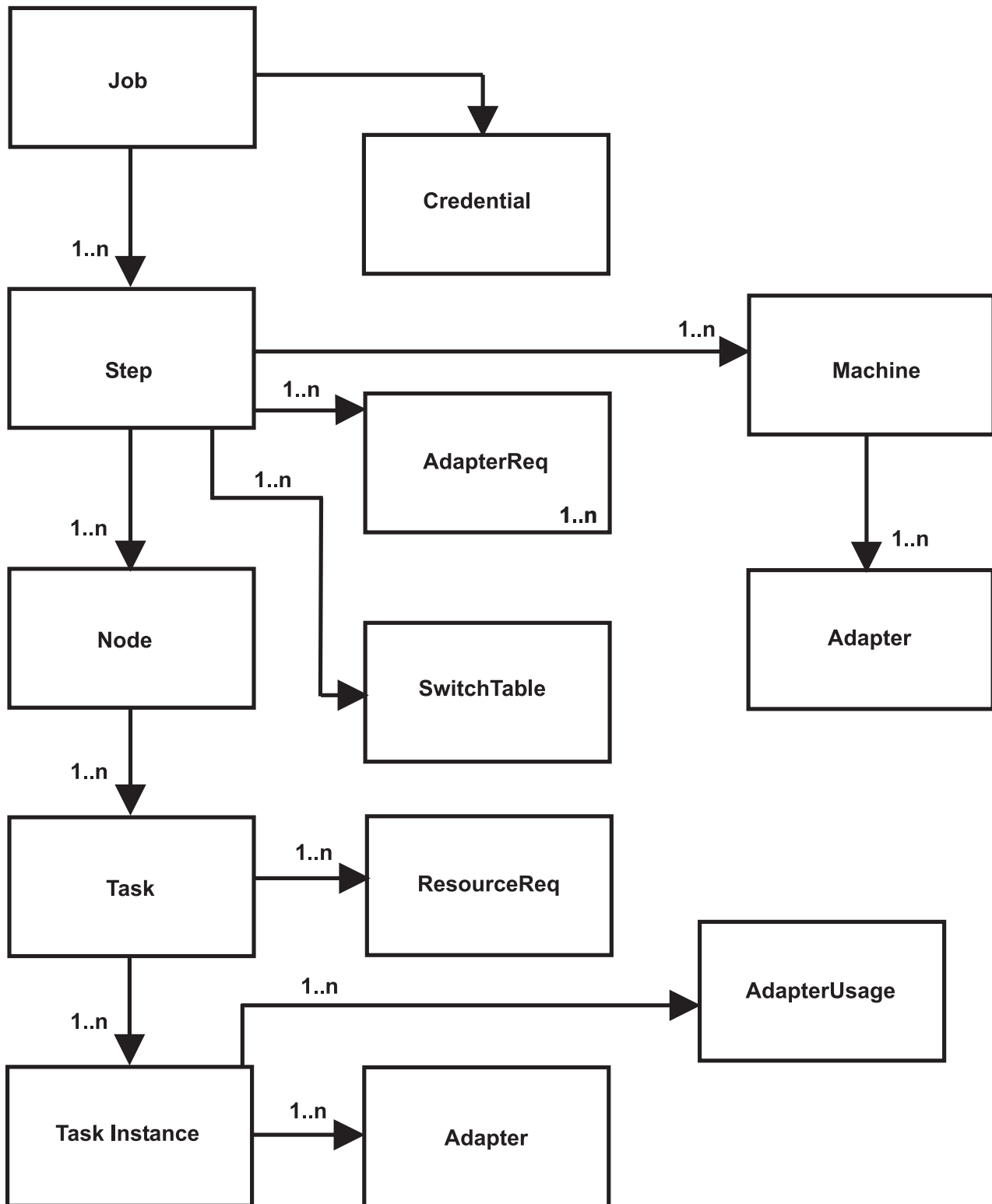


Figure 26. LoadLeveler job object model

The Task object represents one or more copies of the same executable. The Task object can be queried for the executable, the executable arguments, and the number of instances of the executable.



Table 15 on page 234 describes 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 (**LL\_Job**) and the value returned (**GetFirstStep**).

This table is sorted alphabetically by object; within each object the specifications are also sorted alphabetically.

When using the 2.1 release API of **ll\_get\_data**, you must use the new 2.1 release keywords. For instance, you can not use the **min\_processors** and **max\_processors** from the 1.3.0 release with the 2.1 release API **ll\_get\_data**. You must use the new keyword, **node**.

## ll\_get\_data subroutine

Before you use this subroutine, make sure you are familiar with “Understanding the LoadLeveler job object model” on page 230.

### Purpose

The **ll\_get\_data** subroutine returns data from a valid **LL\_element**.

### Library

LoadLeveler API library **libllapi.a**

### 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 or by the **ll\_get\_data** subroutine. For example: Job, Machine, Step, etc.

*specification*

Specifies the data field within the data object you want to read.

*resulting\_data\_type*

Is a pointer to where you want the data stored. If this parameter is equal to **NULL**, then an error has occurred and the value could not be stored.

### 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 Table 15 on page 234. 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.

## Data Access API

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.

When using the 2.1 release API of **ll\_get\_data**, you must use the new 2.1 release keywords. For instance, you can not use the **min\_processors** and **max\_processors** from the 1.3.0 release with the 2.1 release API **ll\_get\_data**. You must use the new keyword, **node**.

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\_query**, **ll\_set\_request**, **ll\_reset\_request**, **ll\_get\_objs**, **ll\_next\_obj**, **ll\_free\_objs**, **ll\_deallocate**.

Table 15. Specifications for *ll\_get\_data* subroutine

Object	Specification	Resulting Data Type	Description
Adapter	LL_AdapterAvailWindowCount	int*	A pointer to an integer indicating the number of windows not in use.
Adapter	LL_AdapterCommInterface	int*	A pointer to a string containing the adapter's communication interface.
Adapter	LL_AdapterInterfaceAddress	int*	A pointer to a string containing the adapter's interface IP address.
Adapter	LL_AdapterMaxWindowSize	int*	A pointer to the integer indicating the maximum allocatable window memory.
Adapter	LL_AdapterMemory	int*	A pointer to the integer indicating the amount of total adapter memory.
Adapter	LL_AdapterMinWindowSize	int*	A pointer to the integer indicating the minimum allocatable window memory.
Adapter	LL_AdapterName	int*	A pointer to a string containing the adapter name.
Adapter	LL_AdapterTotalWindowCount	int*	A pointer to the integer indicating the number of windows on the adapter.
Adapter	LL_AdapterUsageMode	int*	A pointer to a string containing the mode used for css IP or US.
Adapter	LL_AdapterUsageProtocol	int*	A pointer to a string containing the task's protocol.
Adapter	LL_AdapterUsageWindow	int*	A pointer to a string containing the window assigned to the task.

Table 15. Specifications for *ll\_get\_data* subroutine (continued)

Object	Specification	Resulting Data Type	Description
Adapter	LL_AdapterUsageWindowMemory	int*	A pointer to the integer indicating the number of bytes used by the window.
AdapterReq	LL_AdapterReqCommLevel	int*	A pointer to the integer indicating the adapter's communication level.
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.
Cluster	LL_ClusterGetFirstResource	LL_element*	A pointer to the element associated with the first resource.
Cluster	LL_ClusterGetNextResource	LL_element*	A pointer to the element associated with the next resource.
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_ClusterDefinedResourceCount	int*	A pointer to an integer indicating the number of consumable resources defined in the cluster.
Cluster	LL_ClusterEnforcedResources	char**	A pointer to an array of characters indicating the number of enforced resources
Cluster	LL_ClusterEnforcedResourceCount	int*	A pointer to an integer indicating the number of enforced resources
Cluster	LL_ClusterEnforceSubmission	int*	A pointer to a boolean integer indicating resources required at time of submission
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.
Cluster	LL_ClusterSchedulingResourceCount	int*	A pointer to an integer indicating the number of consumable resources considered by the scheduler for the cluster.
Column	LL_ColumnMachineName	char*	A pointer to a string containing the machine name for the Gang Matrix column.
Column	LL_ColumnProcessorNumber	int*	A pointer to an integer indicating the processor number of the Gang Matrix column.
Column	LL_ColumnRowCount	int*	A pointer to an integer indicating the number of rows of this column.
Column	LL_ColumnStepNames	char**	A pointer to an array containing the list of job step names associated with the column. The array ends with a null string.

## Data Access API

Table 15. Specifications for *ll\_get\_data* subroutine (continued)

Object	Specification	Resulting Data Type	Description
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.
Credential	LL_CredentialUserName	char*	A pointer to a string containing the user ID of the user submitting the job.
DispUsage	LL_DispatchUsageEventUsageCount	int*	Count of Event Usages
DispUsage	LL_DispatchUsageGetFirstEventUsage	LL_element*	First Event Usage
DispUsage	LL_DispatchUsageGetNextEventUsage	LL_element*	Next Event Usage
DispUsage	LL_DispatchUsageStarterIdrss64	int64_t*	Starter idrss value of dispatch
DispUsage	LL_DispatchUsageStarterInblock64	int64_t*	Starter inblock value of dispatch
DispUsage	LL_DispatchUsageStarterIsrss64	int64_t*	Starter isrss value of dispatch
DispUsage	LL_DispatchUsageStarterIxrss64	int64_t*	Starter ixrss value of dispatch
DispUsage	LL_DispatchUsageStarterMajflt64	int64_t*	Starter majflt value of dispatch
DispUsage	LL_DispatchUsageStarterMaxrss64	int64_t*	Starter maxrss value of dispatch
DispUsage	LL_DispatchUsageStarterMinflt64	int64_t*	Starter minflt value of dispatch
DispUsage	LL_DispatchUsageStarterMsgrcv64	int64_t*	Starter msgrcv value of dispatch
DispUsage	LL_DispatchUsageStarterMsgsnd64	int64_t*	Starter msgsnd value of dispatch
DispUsage	LL_DispatchUsageStarterNivcs64	int64_t*	Starter nivcs value of dispatch
DispUsage	LL_DispatchUsageStarterNsignals64	int64_t*	Starter nsignals value of dispatch
DispUsage	LL_DispatchUsageStarterNswap64	int64_t*	Starter nswap value of dispatch
DispUsage	LL_DispatchUsageStarterNvcs64	int64_t*	Starter nvcs value of dispatch
DispUsage	LL_DispatchUsageStarterOublock64	int64_t*	Starter oublock value of dispatch
DispUsage	LL_DispatchUsageStarterSystemTime64	int64_t*	Starter system time of dispatch
DispUsage	LL_DispatchUsageStarterUserTime64	int64_t*	Starter user time of dispatch
DispUsage	LL_DispatchUsageStepIdrss64	int64_t*	Step idrss value of dispatch
DispUsage	LL_DispatchUsageStepInblock64	int64_t*	Step inblock value of dispatch
DispUsage	LL_DispatchUsageStepIsrss64	int64_t*	Step isrss value of dispatch
DispUsage	LL_DispatchUsageStepIxrss64	int64_t*	Step ixrss value of dispatch
DispUsage	LL_DispatchUsageStepMajflt64	int64_t*	Step majflt value of dispatch
DispUsage	LL_DispatchUsageStepMaxrss64	int64_t*	Step maxrss value of dispatch
DispUsage	LL_DispatchUsageStepMinflt64	int64_t*	Step minflt value of dispatch
DispUsage	LL_DispatchUsageStepMsgrcv64	int64_t*	Step msgrcv value of dispatch
DispUsage	LL_DispatchUsageStepMsgsnd64	int64_t*	Step msgsnd value of dispatch
DispUsage	LL_DispatchUsageStepNivcs64	int64_t*	Step nivcs value of dispatch
DispUsage	LL_DispatchUsageStepNsignals64	int64_t*	Step nsignals value of dispatch

Table 15. Specifications for *ll\_get\_data* subroutine (continued)

Object	Specification	Resulting Data Type	Description
DispUsage	LL_DisUsageStepNswap64	int64_t*	Step nswap value of dispatch
DispUsage	LL_DisUsageStepNvcsw64	int64_t*	Step nvcsw value of dispatch
DispUsage	LL_DisUsageStepOublock64	int64_t*	Step oublock value of dispatch
DispUsage	LL_DisUsageStepSystemTime64	int64_t*	Step system time of dispatch.
DispUsage	LL_DisUsageStepUserTime64	int64_t*	Step user time of dispatch
EventUsage	LL_EventUsageEventId	int*	Event id
EventUsage	LL_EventUsageEventName	char*	Event name
EventUsage	LL_EventUsageEventTimestamp	int*	Event timestamp
EventUsage	LL_EventUsageStarterIdrss64	int64_t*	Starter idrss value of event
EventUsage	LL_EventUsageStarterInblock64	int64_t*	Starter inblock value of event
EventUsage	LL_EventUsageStarterIsrss64	int64_t*	Starter isrss value of event
EventUsage	LL_EventUsageStarterIxrss64	int64_t*	Starter ixrss value of event
EventUsage	LL_EventUsageStarterMajflt64	int64_t*	Starter majflt value of event
EventUsage	LL_EventUsageStarterMaxrss64	int64_t*	Starter maxrss value of event
EventUsage	LL_EventUsageStarterMinflt64	int64_t*	Starter minflt value of event
EventUsage	LL_EventUsageStarterMsgrcv64	int64_t*	Starter msgrcv value of event
EventUsage	LL_EventUsageStarterMsgsnd64	int64_t*	Starter msgsnd value of event
EventUsage	LL_EventUsageStarterNivcsw64	int64_t*	Starter nivcsw value of event
EventUsage	LL_EventUsageStarterNsignals64	int64_t*	Starter nsignals value of event
EventUsage	LL_EventUsageStarterNswap64	int64_t*	Starter nswap value of event
EventUsage	LL_EventUsageStarterNvcsw64	int64_t*	Starter nvcsw value of event
EventUsage	LL_EventUsageStarterOublock64	int64_t*	Starter oublock value of event
EventUsage	LL_EventUsageStarterSystemTime64	int64_t*	Starter system time of event
EventUsage	LL_EventUsageStarterUserTime64	int64_t*	Starter user time of event
EventUsage	LL_EventUsageStepIdrss64	int64_t*	Step idrss value of event
EventUsage	LL_EventUsageStepInblock64	int64_t*	Step inblock value of event
EventUsage	LL_EventUsageStepIsrss64	int64_t*	Step isrss value of event
EventUsage	LL_EventUsageStepIxrss64	int64_t*	Step ixrss value of event
EventUsage	LL_EventUsageStepMajflt64	int64_t*	Step majflt value of event
EventUsage	LL_EventUsageStepMaxrss64	int64_t*	Step maxrss value of event
EventUsage	LL_EventUsageStepMinflt64	int64_t*	Step minflt value of event
EventUsage	LL_EventUsageStepMsgrcv64	int64_t*	Step msgrcv value of event
EventUsage	LL_EventUsageStepMsgsnd64	int64_t*	Step msgsnd value of event
EventUsage	LL_EventUsageStepNivcsw64	int64_t*	Step nivcsw value of event
EventUsage	LL_EventUsageStepNsignals64	int64_t*	Step nsignals value of event
EventUsage	LL_EventUsageStepNswap64	int64_t*	Step nswap value of event
EventUsage	LL_EventUsageStepNvcsw64	int64_t*	Step nvcsw value of event
EventUsage	LL_EventUsageStepOublock64	int64_t*	Step oublock value of event
EventUsage	LL_EventUsageStepSystemTime64	int64_t*	Step system time of event

## Data Access API

Table 15. Specifications for *ll\_get\_data* subroutine (continued)

Object	Specification	Resulting Data Type	Description
EventUsage	LL_EventUsageStepUserTime64	int64_t*	Step user time of event
Job	LL_JobCredential	LL_element*	A pointer to the element associated with the job credential.
Job	LL_JobGetFirstStep	LL_element*	A pointer to the element associated with the first step of the job, to be used in subsequent <b>ll_get_data</b> calls.
Job	LL_JobGetNextStep	LL_element*	A pointer to the element associated with the next step.
Job	LL_JobName	char*	A pointer to a character string containing the job name.
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_JobVersionNum	int*	A pointer to an integer indicating the job's version number
Machine	LL_MachineAdapterList	char**	A pointer to an array containing the list 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.
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_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_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.

Table 15. Specifications for `ll_get_data` subroutine (continued)

Object	Specification	Resulting Data Type	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_MachineDrainingClassList	char**	A pointer to an array containing the draining class list on the machine. The array ends with a NULL string.
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_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*	A pointer to the element associated with the machine's first adapter.
Machine	LL_MachineGetFirstResource	LL_element*	A pointer to the element associated with the machine's first resource.
Machine	LL_MachineGetNextAdapter	LL_element*	A pointer to the element associated with the machine's next adapter.
Machine	LL_MachineGetNextResource	LL_element*	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_MachineLoadAverage	double*	A pointer to a double containing the load average on the machine.
Machine	LL_MachineMaxTasks	int*	A pointer to an integer indicating the maximum number of tasks this machine can run at one time.
Machine	LL_MachineMachineMode	char*	A pointer to a string containing the configured machine mode.
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.
Machine	LL_MachinePagesFreed	int*	A pointer to an integer indicating the number of pages freed per second by the page replacement algorithm.

## Data Access API

Table 15. Specifications for *ll\_get\_data* subroutine (continued)

Object	Specification	Resulting Data Type	Description
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_MachineRealMemory	int*	A pointer to an integer indicating the physical memory in MBs on the machine.
Machine	LL_MachineRealMemory64	int64_t*	A pointer to a 64-bit integer indicating the physical memory in MBs on the machine.
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.



Table 15. Specifications for *ll\_get\_data* subroutine (continued)

Object	Specification	Resulting Data Type	Description
Machine	LL_MachineStartExpr	char*	A pointer to a string containing the machine's start control expression.
Machine	LL_MachineStartdRunningJobs	int*	A pointer to an integer containing the number of running jobs known by the startdd daemon.
Machine	LL_MachineStartdState	char*	A pointer to a string containing the state of the startdd daemon.
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_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.
MachUsage	LL_MachUsageDispUsageCount	int*	Count of Dispatch Usages
MachUsage	LL_MachUsageGetFirstDispUsage	LL_element*	First Dispatch Usage
MachUsage	LL_MachUsageGetNextDispUsage	LL_element*	Next Dispatch Usage
MachUsage	LL_MachUsageMachineName	char*	Machine name
MachUsage	LL_MachUsageMachineSpeed	double*	Machine speed
Matrix	LL_MatrixColumnCount	int*	A pointer to an integer indicating the number of columns connected to the matrix.
Matrix	LL_MatrixGetFirstColumn	LL_element*	A pointer to the element associated with the first column of the matrix, to be used in subsequent <i>ll_get_data</i> calls.
Matrix	LL_MatrixGetNextColumn	LL_element*	A pointer to the element associated with the next column.
Matrix	LL_MatrixRowCount	int*	A pointer to an integer indicating the number of rows of the longest columns.
Matrix	LL_MatrixTimeSlice	int*	A pointer to an integer indicating the value for the GANG_MATRIX_TIME_SLICE.
Node	LL_NodeGetFirstTask	LL_element*	A pointer to the element associated with the first task for this node.

## Data Access API

Table 15. Specifications for *ll\_get\_data* subroutine (continued)

Object	Specification	Resulting Data Type	Description
Node	LL_NodeGetNextTask	LL_element*	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.
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_ResourceName	char*	A pointer to a string containing the resource name.
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.
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.
Step	LL_StepAccountNumber	char*	A pointer to a string containing the account number specified by the user submitting the job.
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"> <li>• Returns -1 if unlimited is specified</li> <li>• Returns 0 if blocking is unspecified</li> </ul>
Step	LL_StepClassSystemPriority	int*	A pointer to an integer indicating the class priority of the job step.

Table 15. Specifications for `ll_get_data` subroutine (continued)

Object	Specification	Resulting Data Type	Description
Step	LL_StepCheckpointable	int*	A pointer to an integer indicating if checkpointing was enabled via the <b>checkpoint</b> keyword (0=disabled, 1=enabled).
Step	LL_StepCheckpointing	Boolean	If <b>True</b> , indicates that a checkpoint is currently being taken for the step.
Step	LL_StepCkptAccumTime	int*	A pointer to an integer indicating the amount of accumulated time, in seconds, that the job step has spent checkpointing.
Step	LL_StepCkptFailStartTime	time_t*	A pointer to a time_t structure indicating the start time of the last unsuccessful checkpoint.
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.
Step	LL_StepCkptGoodStartTime	time_t*	A pointer to a time_t structure indicating the start time of the last successful checkpoint.
Step	LL_StepCkptRestart	int*	A pointer to an integer indicating the value specified by the user for the <b>restart_from_ckpt</b> keyword (0= no, 1= yes).
Step	LL_StepCkptRestartSameNodes	int*	A pointer to a string indicating the value specified by the user for the <b>restart_on_same_nodes</b> keyword (0= no, 1= yes).
Step	LL_StepCkptTimeHardLimit	int*	A pointer to an integer indicating the hard limit set by the user in the <b>ckpt_time_limit</b> keyword.
Step	LL_StepCkptTimeHardLimit64	int64_t*	A pointer to a 64-bit integer indicating the hard limit set by the user in the <b>ckpt_time_limit</b> keyword.
Step	LL_StepCkptTimeSoftLimit	int*	A pointer to an integer indicating the soft limit set by the user in <b>ckpt_time_limit</b> keyword.
Step	LL_StepCkptTimeSoftLimit64	int64_t*	A pointer to a 64-bit integer indicating the soft limit set by the user in <b>ckpt_time_limit</b> keyword.
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.

## Data Access API

Table 15. Specifications for *ll\_get\_data* subroutine (continued)

Object	Specification	Resulting Data Type	Description
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 <b>core_limit</b> keyword.
Step	LL_StepCoreLimitHard64	int64_t*	A pointer to a 64-bit integer indicating the core hard limit set by the user in the <b>core_limit</b> keyword.
Step	LL_StepCoreLimitSoft	int*	A pointer to an integer indicating the core soft limit set by the user in the <b>core_limit</b> keyword.
Step	LL_StepCoreLimitSoft64	int64_t*	A pointer to a 64-bit integer indicating the core soft limit set by the user in the <b>core_limit</b> keyword.
Step	LL_StepCpuLimitHard	int*	A pointer to an integer indicating the CPU hard limit set by the user in the <b>cpu_limit</b> keyword.
Step	LL_StepCpuLimitHard64	int64_t*	A pointer to a 64-bit integer indicating the CPU hard limit set by the user in the <b>cpu_limit</b> keyword.
Step	LL_StepCpuLimitSoft	int*	A pointer to an integer indicating the CPU soft limit set by the user in the <b>cpu_limit</b> keyword.
Step	LL_StepCpuLimitSoft64	int64_t*	A pointer to a 64-bit integer indicating the CPU soft limit set by the user in the <b>cpu_limit</b> keyword.
Step	LL_StepCpuStepLimitHard	int*	A pointer to an integer indicating the CPU step hard limit set by the user in the <b>job_cpu_limit</b> 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 <b>job_cpu_limit</b> keyword.
Step	LL_StepCpuStepLimitSoft	int*	A pointer to an integer indicating the CPU step soft limit set by the user in the <b>job_cpu_limit</b> 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 <b>job_cpu_limit</b> keyword.
Step	LL_StepDataLimitHard	int*	A pointer to an integer indicating the data hard limit set by the user in the <b>data_limit</b> keyword.
Step	LL_StepDataLimitHard64	int64_t*	A pointer to a 64-bit integer indicating the data hard limit set by the user in the <b>data_limit</b> keyword.
Step	LL_StepDataLimitSoft	int*	A pointer to an integer indicating the data soft limit set by the user in the <b>data_limit</b> keyword.

Table 15. Specifications for *ll\_get\_data* subroutine (continued)

Object	Specification	Resulting Data Type	Description
Step	LL_StepDataLimitSoft64	int64_t*	A pointer to a 64-bit integer indicating the data soft limit set by the user in the <b>data_limit</b> keyword.
Step	LL_StepDispatchTime	time_t*	A pointer to a time_t structure indicating the time the negotiator dispatched the job.
Step	LL_StepEnvironment	char*	A pointer to a string containing the environment variables set by the user in the executable.
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_StepExecutionFactor	int*	A pointer to an integer indicating the execution_factor of the job step.
Step	LL_StepFileLimitHard	int*	A pointer to an integer indicating the file hard limit set by the user in the <b>file_limit</b> keyword.
Step	LL_StepFileLimitHard64	int64_t*	A pointer to a 64-bit integer indicating the file hard limit set by the user in the <b>file_limit</b> keyword.
Step	LL_StepFileLimitSoft	int*	A pointer to an integer indicating the file soft limit set by the user in the <b>file_limit</b> keyword.
Step	LL_StepFileLimitSoft64	int64_t*	A pointer to a 64-bit integer indicating the file soft limit set by the user in the <b>file_limit</b> keyword.
Step	LL_StepGetFirstAdapterReq	LL_element*	A pointer to the element associated with the first adapter requirement.
Step	LL_StepGetFirstMachine	LL_element*	A pointer to the element associated with the first machine in the step.
Step	LL_StepGetFirstMachUsage	LL_element*	First Mach Usage
Step	LL_StepGetFirstNode	LL_element*	A pointer to the element associated with the first node of the step.
Step	LL_StepGetFirstSwitchTable	LL_element*	A pointer to the element associated with the first switch table for this step.
Step	LL_StepGetMasterTask	LL_element*	A pointer to the element associated with the master task of the step.
Step	LL_StepGetNextAdapterReq	LL_element*	A pointer to the element associated with the next adapter requirement.
Step	LL_StepGetNextMachine	LL_element*	A pointer to the element associated with the next machine of the step.
Step	LL_StepGetNextMachUsage	LL_element*	Next Mach Usage of step
Step	LL_StepGetNextNode	LL_element*	A pointer to the element associated with the next node of the step.

## Data Access API

Table 15. Specifications for *ll\_get\_data* subroutine (continued)

Object	Specification	Resulting Data Type	Description
Step	LL_StepGetNextSwitchTable	LL_element*	A pointer to the element associated with the next switch table for this step.
Step	LL_StepGroupSystemPriority	int*	A pointer to an integer indicating the group priority of a job step.
Step	LL_StepHoldType	int*	A pointer to an integer indicating the hold state of the step (user, system, etc). The value returned is in the HoldType enum.
Step	LL_StepHostList	char**	A pointer to an array containing the list of hosts in the <b>host.list</b> file associated with the step. The array ends with a null string.
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.
Step	LL_StepImageSize64	int64_t*	A pointer to a 64-bit integer indicating the image size of the executable.
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.
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.
Step	LL_StepMachUsageCount	int*	Count of Machine Usages
Step	LL_StepMessages	char*	A pointer to a string containing a list of messages from LL
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, NOT_SHARED, or SLICE_NOT_SHARED). The value returned is in the enum Usage.

Table 15. Specifications for `ll_get_data` subroutine (continued)

Object	Specification	Resulting Data Type	Description
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_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. Only the CM has the current value for LL_StepQueueSystemPriority.
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"> <li>• 1 indicates yes</li> <li>• 0 indicates no</li> </ul>
Step	LL_StepRssLimitHard	int*	A pointer to an integer indicating the RSS hard limit set by the user in the <b>rss_limit</b> keyword.
Step	LL_StepRssLimitHard64	int64_t*	A pointer to a 64-bit integer indicating the RSS hard limit set by the user in the <b>rss_limit</b> keyword.
Step	LL_StepRssLimitSoft	int*	A pointer to an integer indicating the RSS soft limit set by the user in the <b>rss_limit</b> keyword.
Step	LL_StepRssLimitSoft64	int64_t*	A pointer to a 64-bit integer indicating the RSS soft limit set by the user in the <b>rss_limit</b> keyword.
Step	LL_StepShell	char*	A pointer to a character string containing the shell name used by the executable.
Step	LL_StepStackLimitHard	int*	A pointer to an integer indicating the stack hard limit set by the user in the <b>stack_limit</b> keyword.
Step	LL_StepStackLimitHard64	int64_t*	A pointer to a 64-bit integer indicating the stack hard limit set by the user in the <b>stack_limit</b> keyword.
Step	LL_StepStackLimitSoft	int*	A pointer to an integer indicating the stack soft limit set by the user in the <b>stack_limit</b> keyword.
Step	LL_StepStackLimitSoft64	int64_t*	A pointer to a 64-bit integer indicating the stack soft limit set by the user in the <b>stack_limit</b> keyword.
Step	LL_StepStartCount	int*	A pointer to an integer indicating the number of times the step has been started.

## Data Access API

Table 15. Specifications for `ll_get_data` subroutine (continued)

Object	Specification	Resulting Data Type	Description
Step	LL_StepStartDate	time_t*	A pointer to a time_t structure indicating the value the user specified in the <b>startdate</b> keyword.
Step	LL_StepStarterIdrss64	int64_t*	Starter idrss value
Step	LL_StepStarterInblock64	int64_t*	Starter inblock value
Step	LL_StepStarterIsrss64	int64_t*	Starter isrss value
Step	LL_StepStarterIxrss64	int64_t*	Starter ixrss value
Step	LL_StepStarterMajflt64	int64_t*	Starter majflt value
Step	LL_StepStarterMaxrss64	int64_t*	Starter maxrss value
Step	LL_StepStarterMinflt64	int64_t*	Starter minflt value
Step	LL_StepStarterMsgrcv64	int64_t*	Starter msgrcv value
Step	LL_StepStarterMsgsnd64	int64_t*	Starter msgsnd value
Step	LL_StepStarterNivcs64	int64_t*	Starter nivcs64 value
Step	LL_StepStarterNsignals64	int64_t*	Starter nsignals value
Step	LL_StepStarterNswap64	int64_t*	Starter nswap value
Step	LL_StepStarterNvcs64	int64_t*	Starter nvcs64 value
Step	LL_StepStarterOublock64	int64_t*	Starter oublock value
Step	LL_StepStarterSystemTime64	int64_t*	Starter system time
Step	LL_StepStarterUserTime64	int64_t*	Starter user time
Step	LL_StepState	int*	A pointer to an integer indicating the state of the Step (Idle, Pending, Starting, etc.). The value returned is in the StepState enum.
Step	LL_StepStepIdrss64	int64_t*	Step idrss value
Step	LL_StepStepInblock64	int64_t*	Step inblock value
Step	LL_StepStepIsrss64	int64_t*	Step isrss value
Step	LL_StepStepIxrss64	int64_t*	Step ixrss value
Step	LL_StepStepMajflt64	int64_t*	Step majflt value
Step	LL_StepStepMaxrss64	int64_t*	Step maxrss value
Step	LL_StepStepMinflt64	int64_t*	Step minflt value
Step	LL_StepStepMsgrcv64	int64_t*	Step msgrcv value
Step	LL_StepStepMsgsnd64	int64_t*	Step msgsnd value
Step	LL_StepStepNivcs64	int64_t*	Step nivcs64 value
Step	LL_StepStepNsignals64	int64_t*	Step nsignals value
Step	LL_StepStepNswap64	int64_t*	Step nswap value
Step	LL_StepStepNvcs64	int64_t*	Step nvcs64 value
Step	LL_StepStepOublock64	int64_t*	Step oublock value
Step	LL_StepStepSystemTime64	int64_t*	Step system time
Step	LL_StepStepUserTime64	int64_t*	Step user time



Table 15. Specifications for `ll_get_data` subroutine (continued)

Object	Specification	Resulting Data Type	Description
Step	LL_StepSystemPriority	int*	A pointer to an integer indicating the overall system priority of the job step. Only the CM has the current value for LL_StepSystemPriority.
Step	LL_StepTaskGeometry	char*	A pointer to a string containing the values specified in the <code>task_geometry</code> statement by the user in the job command file. The syntax is the same as specified in the statement <code>, {(task id, task id, ...) (task id, task id, ...) ...}</code> . If unspecified, a null string is returned.
Step	LL_StepTaskInstanceCount	int*	A pointer to an integer indicating the number of task instances in the step. This is only available from the schedd daemon.
Step	LL_StepTasksPerNode Requested	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.
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 <code>, [min],[max]</code> , where min contains the minimum number of nodes requested and max contains the maximum nodes requested. If unspecified, a null string is returned.
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.
Step	LL_StepUserSystemPriority	int*	A pointer to an integer indicating the user system priority of the job step.
Step	LL_StepWallClockLimitHard	int*	A pointer to an integer indicating the wall clock hard limit set by the user in the <b>wall_clock_limit</b> 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 <b>wall_clock_limit</b> keyword.
Step	LL_StepWallClockLimitSoft	int*	A pointer to an integer indicating the wall clock soft limit set by the user in the <b>wall_clock_limit</b> 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 <b>wall_clock_limit</b> keyword.

## Data Access API

Table 15. Specifications for *ll\_get\_data* subroutine (continued)

Object	Specification	Resulting Data Type	Description
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	A pointer to the element associated with the first resource requirement.
Task	LL_TaskGetFirstTaskInstance	LL_element*	A pointer to the element associated with the first task instance.
Task	LL_TaskGetNextResourceRequirement	LL_element*	A pointer to the element associated with the next resource requirement.
Task	LL_TaskGetNextTaskInstance	LL_element*	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.
Task Instance	LL_TaskInstanceAdapterCount	int*	A pointer to the integer indicating the number of adapters.
Task Instance	LL_TaskInstanceGetFirstAdapter	LL_element*	A pointer to the element associated with the first adapter.
Task Instance	LL_TaskInstanceGetFirstAdapterUsage	LL_element*	A pointer to the element associated with the first adapter usage.
Task Instance	LL_TaskInstanceGetNextAdapter	LL_element*	A pointer to the element associated with the next adapter.
Task Instance	LL_TaskInstanceGetNextAdapterUsage	LL_element*	A pointer to the element associated with the next adapter usage.
Task Instance	LL_TaskInstanceMachineName	char*	A pointer to the string indicating the machine assigned to a task.
Task Instance	LL_TaskInstanceTaskID	int*	A pointer to the integer indicating the task ID.
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_WlmStatMemorySnapshotUsage	int*	A pointer to real memory 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.

## ll\_next\_obj subroutine

### Purpose

The **ll\_next\_obj** subroutine returns the next object in the *query\_element* list you specify.

### Library

LoadLeveler API library **libllapi.a**

### 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\_get\_data**, **ll\_set\_request**, **ll\_query**, **ll\_get\_objs**, **ll\_free\_objs**, **ll\_deallocate**.

## ll\_free\_objs subroutine

### Purpose

The **ll\_free\_objs** subroutine 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**

### 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.

## Data Access API

### Error values

-1      You specified a *query\_element* that is not valid.

### Related information

Subroutines: **ll\_get\_data**, **ll\_set\_request**, **ll\_query**, **ll\_get\_objs**, **ll\_reset\_request**, **ll\_free\_objs**.

## ll\_deallocate subroutine

### Purpose

The **ll\_deallocate** subroutine deallocates the *query\_element* allocated by the **ll\_query** subroutine.

### Library

LoadLeveler API library **libllapi.a**

### 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\_get\_data**, **ll\_set\_request**, **ll\_query**, **ll\_get\_objs**, **ll\_reset\_request**, **ll\_next\_obj**, **ll\_free\_objs**.

## 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 27. Obtaining machine, job, and cluster information with the Data Access API (Part 1 of 4)

## Data Access API

```
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 27. Obtaining machine, job, and cluster information with the Data Access API (Part 2 of 4)

```

*
* 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 27. Obtaining machine, job, and cluster information with the Data Access API (Part 3 of 4)

## Data Access API

```
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 27. 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.



```

#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[] = { "07/23/2001", "07/25/2001", NULL };
    char *end_dates[] = { "07/23/2001", "08/01/2001", 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 28. Extracting job accounting information from a history file (Part 1 of 3)

## Data Access API

```
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 28. Extracting job accounting information from a history file (Part 2 of 3)

```

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 28. 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

This routine converts a LoadLeveler error object to an error message string. As an option, you can print the error message string to stdout or stderr.

## Data Access API

### Library

LoadLeveler API library libllapi.a

### 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.

*print\_to*

1 - print error message to stdout

2 - print error message to stderr

Any other value - no error message printed

### Description

It is 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 pointer to an error message string.

### Error values

The **ll\_error** API returns a NULL if the error object is NULL.

---

## Parallel Job API

If you are using any of the parallel operating environments already supported by LoadLeveler, you do not have to use the parallel API. However, if you have another application environment that you want to use, you need to use the subroutines described here to interface with LoadLeveler.

The parallel job API consists of two subroutines. **ll\_get\_hostlist** acquires the list of LoadLeveler selected parallel nodes, and **ll\_start\_host** starts the parallel task under the LoadLeveler starter.

The following section describes how parallel job submission works. Understanding this will help you to better understand the parallel API.

## Interaction between LoadLeveler and the parallel API

This API does not give you access to any new LoadLeveler functions from Version 2 Release 1.0, or later releases.

Program applications which use the parallel APIs to interface with LoadLeveler are supported under a job type called **parallel**. When a user submits a job specifying the keyword **job\_type** equal to **parallel**, the LoadLeveler API job control flow is as follows:

The negotiator selects nodes based on the resources you request. Once the nodes have been obtained, the negotiator contacts the schedd to start the job. The schedd marks the job pending and contacts the affected startds to start their starter processes.

One machine becomes the **Master Starter**. The Master Starter is one of the selected parallel nodes. After all starters are started and have completed initialization, the Master Starter starts the executable specified in the job command file. The executable referred to as the **Parallel Master** uses this API to start tasks on remote nodes. A `LOADLBATCH` environment variable is set to YES so that the Parallel Master can distinguish between callers.

The Parallel Master must:

- Obtain the machine list through the **ll\_get\_hostlist** API.
- Start a task on all allocated machines through the **ll\_start\_host** API. It is mandatory that one and only one task be started on each machine. Each task is considered a Parallel Slave. Acquiring the task name, path and arguments is the responsibility of the Parallel Master. The user may pass this information through the **arguments** or **environment** keywords in the job command file.

When the Parallel Master starts, the job is marked Running. Once the Parallel Master and all tasks exit, the job is marked Complete.

### Termination paths

The Parallel Master is expected to cleanup and exit when:

- All of the Parallel Slaves have exited.
- A negative value is returned by either the **ll\_get\_hostlist** or **ll\_start\_host** subroutine.
- A SIGCONT, followed by a SIGTERM, is received. A possible reason for this is that LoadLeveler receives a job cancel request.  
The SIGTERM is also sent to all parallel tasks.
- A SIGCONT, followed by a SIGUSR1, is received. Reasons for this include:
  - The Parallel Master receives a VACATE or FLUSH request.
  - LoadLeveler receives a stop LoadLeveler daemons command.

The SIGUSR1 is also sent to all parallel tasks.

A SIGKILL is issued to any process which does not exit within two minutes of receiving a termination signal.

Note that a SIGUSR1 indicates the job must terminate but will be restarted, while a SIGTERM indicates the job must terminate but will not be restarted.

## ll\_get\_hostlist subroutine

### Purpose

This subroutine obtains a list of machines from the Master Starter machine so that the Parallel Master can start the Parallel Slaves. The Parallel Master is the LoadLeveler executable specified in the job command file and the Parallel Slaves are the processes started by the Parallel Master through the **ll\_start\_host** API.

**Note:** This API is obsolete and is supported for backward compatibility only.

### Library

LoadLeveler API library **libllapi.a**

### Syntax

```
int ll_get_hostlist(struct JM_JOB_INFO* jobinfo);
```

## Parallel Job API

### Parameters

*jobinfo* is a pointer to the JM\_JOB\_INFO structure defined in **llapi.h**. No fields are required to be filled in. **ll\_get\_hostlist** allocates storage for an array of JM\_NODE\_INFO structures and returns the pointer in the *jm\_min\_node\_info* pointer. It is the caller's responsibility to free this storage.

```
struct JM_JOB_INFO {
    int  jm_request_type;
    char  jm_job_description[50];
    enum  JM_ADAPTER_TYPE jm_adapter_type;
    int  jm_css_authentication;
    int  jm_min_num_nodes;
    struct JM_NODE_INFO *jm_min_node_info;
};

struct JM_NODE_INFO {
    char  jm_node_name [MAXHOSTNAMELEN];
    char  jm_node_address [50];
    int  jm_switch_node_number;
    int  jm_pool_id;
    int  jm_cpu_usage;
    int  jm_adapter_usage;
    int  jm_num_virtual_tasks;
    int  *jm_virtual_task_ids;
    enum  JM_RETURN_CODE jm_return_code;
};
```

The following data is filled in for the JM\_JOB\_INFO structure:

*jm\_min\_num\_nodes*

Is the number of elements in the array of JM\_NODE\_INFO structures. It is the number of hosts allocated to a job.

*jm\_min\_node\_info*

Is the pointer to the array of JM\_NODE\_INFO structures. The first entry in this array describes the node which is mapped to task 0. The second entry is mapped to task 1, and so on.

The following data is filled in for each JM\_NODE\_INFO structure:

*jm\_node\_name*

Is the name of the node.

*jm\_node\_address*

Is the address corresponding to the adapter requested.

*jm\_switch\_node\_number*

Is the relative node number set only for job running on the SP switch adapter. For all other jobs it is set to -1.

### Description

The Parallel Master must:

- Issue error messages as appropriate.
- Exit when **ll\_get\_hostlist** returns with a negative return value. The Parallel Master exit status is included in the job mail returned to the user.

### Return values

This subroutine returns a zero to indicate success.

### Error values

**-2**       Cannot get LoadLeveler step ID from environment.

- 5 Cannot make socket. This means that the UNIX stream socket could not be created. This socket is needed to establish communications with the starter for both of the API's functions.
- 6 Cannot connect to host.
- 8 Cannot get hostlist.
- 10 DCE identity can not be determined
- 11 No DCE credentials
- 12 DCE credentials within 300 secs of expiration
- 13 64-bit API not supported when DCE is enabled

## ll\_start\_host subroutine

### Purpose

This subroutine starts a task on a selected machine.

### Library

LoadLeveler API library **libllapi.a**

### Syntax

```
int ll_start_host(char *host, char *start_cmd);
```

### Parameters

*host*

Is the name of the node on which you want to start the task.

*start\_cmd*

Is the actual command to execute on the node, including flags and arguments.

### Description

This function must be invoked for all the machines returned from the **ll\_get\_hostlist** subroutine once and only once by the Parallel Master. Acquiring the **start\_cmd** is the responsibility of the Parallel Master. The user may pass this information through the **arguments** or **environment** keywords in the job command file.

The Parallel Master must:

- Issue error messages as appropriate.
- Exit when **ll\_start\_host** returns a negative value. The Parallel Master exit status is included in the job mail returned to the user.

### Return values

This subroutine returns an integer greater than one to indicate the socket connected to the Parallel Slave's standard I/O (stdio).

### Error values

- 2 Cannot get LoadLeveler step ID from environment
- 4 Nameserver cannot resolve host
- 6 Cannot connect to host
- 7 Cannot send PASS\_OPEN\_SOCKET command to remote startd
- 9 The command you specified failed.

## Parallel Job API

### Examples

A sample program called **para\_api.c** is provided in the **samples/lpara** subdirectory of the release directory, usually **/usr/lpp/LoadL/full**.

In order to run this example, you need to do the following:

1. Copy the sample Makefile and the sample program called **para\_api.c** to your home directory.
2. Update the **startCmd** variable in **para\_api.c** to reflect your home directory versus **/usr/lpp/LoadL/full/samples/lpara**. For example:  

```
char *startCmd = "/home/user/para_api -s";
```
3. Issue **make** to create the executable **para\_api**.
4. Update your job command file as follows:

```
#!/bin/ksh
# @ initialdir      = /home/user
# @ executable      = para_api
# @ output          = para_api.${cluster}.${process}.out
# @ error           = para_api.${cluster}.${process}.err
# @ job_type        = parallel
# @ min_processors  = 2
# @ max_processors  = 2
# @ queue
```

5. Submit the job command file to LoadLeveler.

The syntax to invoke the Parallel Master is:

```
para_api
```

The syntax to invoke the Parallel Slave is:

```
para_api -s
```

The Parallel Master does the following:

- Acquires the hostlist through the **ll\_get\_hostlist** API and prints out the returned fields.
- Starts a Parallel Slave task by executing the command specified in the **StartCmd** variable on all hosts returned in the hostlist.
- Acquires the socket connected to the Parallel Slave's standard I/O (stdio).
- Writes a command over the socket to verify stdin.
- Reads acknowledgments over the socket to verify stderr and stdout.
- Prints out host names and acknowledgments.

Example output follows:

```
num_nodes=2

name=host1.kgn.ibm.com address=9.115.8.162 switch_number=-1

name=host2.kgn.ibm.com address=9.115.8.164 switch_number=-1

Connected to host1.kgn.ibm.com at sock 3
Received acko "8000" and acke "10000" from host 0

Connected to host2.kgn.ibm.com at sock 4
Received acko "8001" and acke "10001" from host 1

<Master Exiting>
```

The Parallel Slave does the following:



- Reads command from stdin.
- Writes acknowledgment to stdout and stderr.

---

## Query API

This API provides information about the jobs and machines in the LoadLeveler cluster. You can use this in conjunction 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 270 for more information. These APIs exist for backward compatibility. It is recommended that you use the Data Access API when possible.

The query API consists of the following subroutines: **ll\_get\_jobs**, **ll\_free\_jobs**, **ll\_get\_nodes**, and **ll\_free\_nodes**.

### ll\_get\_jobs subroutine

#### Purpose

This subroutine, available to any user, 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

The **LL\_get\_jobs\_info** structure 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 subroutines 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.
- 16     DCE identity can not be determined
- 17     No DCE credentials
- 18     DCE credentials within 300 secs of expiration
- 19     64-bit API not supported when DCE is enabled

### 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\_free\_jobs subroutine

### Purpose

This subroutine, available to any user, 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 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\_get\_jobs**, **ll\_free\_nodes**, **ll\_get\_nodes**.

## ll\_get\_nodes subroutine

### Purpose

This subroutine, available to any user, 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

The **LL\_get\_node\_info** structure 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.
- 16     DCE identity can not be determined
- 17     No DCE credentials
- 18     DCE credentials within 300 secs of expiration
- 19     64-bit API not supported when DCE is enabled

### 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**.

## Query API

### ll\_free\_nodes subroutine

#### Purpose

This subroutine, available to any user, frees storage that was allocated by **ll\_get\_nodes**.

#### Library

LoadLeveler API library **libllapi.a**

#### Syntax

```
#include "llapi.h"

int ll_nodes_jobs(LL_get_nodes_info *ptr);
```

#### Parameters

*ptr* Specifies the address of the **LL\_get\_nodes\_info** structure to be freed.

#### Description

This subroutine frees the storage pointed to by the **LL\_get\_nodes\_info** pointer.

#### Return values

This subroutine 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\_get\_jobs**, **ll\_free\_nodes**, **ll\_get\_nodes**.

---

## Submit API

This API allows you to submit jobs to LoadLeveler. The submit API consists of the **llsubmit** subroutine, the **llfree\_job\_info** subroutine, and a user exit for monitoring programs.

### llsubmit subroutine

**llsubmit** is both the name of a LoadLeveler command used to submit jobs as well as the subroutine described here.

#### Purpose

The **llsubmit** subroutine submits jobs to LoadLeveler for scheduling.

#### 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 is the uid and gid used when the job is run.

**Return values**

0        The job was submitted successfully.

**Error values**

-1       Error, error messages written to stderr.

**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.

### Monitoring programs

#### 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 didn't 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 four subroutines:

- **ll\_control** subroutine
- **ll\_preempt** subroutine
- **ll\_start\_job** 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\_preempt** subroutine is used to either preempt a job step or resume a job step that has been preempted. The **ll\_start\_job**, and **ll\_terminate\_job** subroutines are intended to be used in conjunction with an external scheduler.

To use an external scheduler, you must specify the following keyword in the global LoadLeveler configuration file:

```
SCHEDULER_TYPE = API
```

Specifying API disables the default LoadLeveler scheduling algorithm. When you disable the default LoadLeveler scheduler, jobs do not start unless requested to do so by the **ll\_start\_job** subroutine.

You can toggle between the default LoadLeveler scheduler and an external scheduler.

If you are running the default LoadLeveler scheduler, this is how you can switch to an external scheduler:

1. In the configuration file, set **SCHEDULER\_TYPE = API**
2. On the central manager machine, issue the **llctl -g stop** and then **llctl -g start** or **llctl -g recycle** commands

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. Issue the **llctl -g stop** and then **llctl -g start** or **llctl -g recycle** commands

Note that the **ll\_start\_job** and **ll\_terminate\_job** subroutines automatically connect to an alternate central manager if they cannot contact the primary central manager.

An example of an external scheduler you can use is the Extensible Argonne Scheduling sYstem (EASY), developed by Argonne National Laboratory and available as public domain code.

You should use **ll\_start\_job** 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 265 for more information.

**Note:** The AIX Workload Manager (WLM) and the LoadLeveler Workload Management API are two distinct and unrelated components.

## ll\_control subroutine

### Purpose

This subroutine allows an application program to perform most of the functions that are currently available through the standalone commands: **llctl**, **llfavorjob**, **llfavoruser**, **llhold**, and **llprio**.

### Library

LoadLeveler API library **libllapi.a**

## Workload Management API

### 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_SCHEDD,  
    LL_CONTROL_PURGE_SCHEDD, LL_CONTROL_FLUSH, LL_CONTROL_SUSPEND,  
    LL_CONTROL_RESUME, LL_CONTROL_RESUME_STARTD, LL_CONTROL_RESUME_SCHEDD,  
    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_PRIO_ABS, LL_CONTROL_PRIO_ADJ };
```

**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 names. The job name that an element of **job\_list** points to is a character string with one of the following formats: "*host.jobid.stepid*," "*jobid.stepid*," "*jobid*". *host* is the name of the machine to which the job was submitted (the default is the local machine), *jobid* is the job ID assigned to the job by LoadLeveler, and *stepid* is the job step ID assigned to a job step by LoadLeveler (the default is to include all the steps of a job).

**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 on the **llctl** command, see "llctl - Control LoadLeveler daemons" on page 148.



### **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\_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\_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,

## Workload Management API

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 him/her.

**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 him/her.

### 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.
- 36 DCE identity can not be determined
- 37 No DCE credentials
- 38 DCE credentials within 300 secs of expiration
- 39 64-bit API not supported when DCE is enabled

### Related information

Commands: **llprio**, **llhold**, **llfavoruser**, **llfavorjob**, **llctl**.

## ll\_modify subroutine

### Purpose

The **ll\_modify** subroutine modifies the attributes of a submitted job step.

### Library

LoadLeveler API library **libllapi.a**

### Syntax

```
#include "llapi.h"
```

```
int ll_modify (int version, LL_element **errObj, LL_modify_param **param, char **joblist) ;
```

### Parameters

*version*

Input parameter that indicates the LoadLeveler API version (should have the same value as **LL\_API\_VERSION** in llapi.h).

## Workload Management API

### *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

### *joblist*

A NULL terminated array of job step names. Only one job step is allowed in the current implementation. Uses the following formats: "*host.jobid.stepid*," "*jobid.stepid*". Where:

- *host*: is the name of the machine to which the job was submitted (the default is the local machine)
- *jobid*: is the job ID assigned to the job by LoadLeveler
- *stepid*: is the job step ID assigned to a job step by LoadLeveler

## Description

**ll\_modify( )** is the API for the **llmodify** command.

In **enum LL\_modify\_op**, the EXECUTION\_FACTOR can be used with Gang scheduling only. Only LoadLeveler administrators have authority to modify this attribute of a job step.

## Return values

The following return values are defined in llapi.h:

### **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\_STATE**

Request failed, job step in wrong state

### **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

**API\_64BIT\_DCE\_ERR**

64-bit API not supported when DCE is enabled

**Related information**

llmodify command, ll\_error ( ) API

**Example**

```

/* 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, exec_factor = 99;
    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 = EXECUTION_FACTOR;
    mycmd.data = &exec_factor;
    cmdp[0] = &mycmd;
    cmdp[1] = NULL;

    /* change execution factor to 99 for the job step */
    printf("Change execution factor to %d\n", exec_factor);
    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\_preempt subroutine****Purpose**

The ll\_preempt subroutine enables you to preempt a running job step or to resume a job step that has already been preempted through the **ll\_preempt** command or the ll\_preempt subroutine (user-initiated). The ll\_preempt subroutine cannot resume a job step preempted through PREEMPT\_CLASS rules (system-initiated).

**Library**LoadLeveler API library, **libllapi.a**

## Workload Management API

### 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 Gang scheduling and 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

#### **API\_CANT\_AUTH**

Caller not authorized

#### **API\_CANT\_CONNECT**

Failed to connect to LoadLeveler

#### **API\_64BIT\_DCE\_ERR**

64-bit API not supported when DCE is enabled

## ll\_start\_job subroutine

### Purpose

This subroutine tells the LoadLeveler negotiator to start a job on the specified nodes.

### Library

LoadLeveler API library **libllapi.a**

**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**

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 in conjunction with the **ll\_get\_nodes** and **ll\_get\_jobs** 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.

**Return values**

This subroutines return 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.

## Workload Management API

- 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*.
- 16 DCE identity can not be determined
- 17 No DCE credentials
- 18 DCE credentials within 300 secs of expiration
- 19 64-bit API not supported when DCE is enabled

### 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\_terminate\_job**, **ll\_get\_nodes**.

## ll\_terminate\_job subroutine

### Purpose

This subroutine tells the negotiator to cancel the specified job step.

### Library

LoadLeveler API library **libllapi.a**

### 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.

**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**

You do not need to disable the default LoadLeveler scheduler in order to use this subroutine.

Only processes having the LoadLeveler administrator user ID can invoke this subroutine.

An external scheduler uses this subroutine in conjunction with the **ll\_get\_job** subroutine (of the job control API) and **ll\_start\_jobs** subroutine (of the query API). The external scheduler must use this subroutine to return errors from **ll\_start\_job** to interactive parallel jobs.

**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.
- 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.
- 17     No DCE credentials
- 18     DCE credentials within 300 secs of expiration
- 19     64-bit API not supported when DCE is enabled

**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\_start\_job**, **ll\_get\_nodes**.

**Usage notes**

It is important to know how LoadLeveler keywords and commands behave when you disable the default LoadLeveler scheduling algorithm. 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.

## Workload Management API

The following sections discuss some specific keywords and commands and how they behave when you disable the default LoadLeveler scheduling algorithm.

### Job command file keywords

**class** – This value is provided by the query APIs. Machines chosen by **ll\_start\_job** must 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.

**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 “How does a job’s priority affect dispatching order?” on page 45). 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** is ignored.

**master\_node\_requirement** is ignored.

**maxidle** is supported.

**maxjobs** is ignored.

**maxqueued** is supported.

**max\_jobs\_scheduled** is ignored.

**priority** is used to calculate the system priority (where appropriate).

**speed** is available through the query API.

### Configuration file keywords

**MACHPRIO** is calculated but is not used.

**SYSPRIO** is calculated and available to the query API.

**MAX\_STARTERS** is calculated, and if starting the job causes this value to be exceeded, **ll\_start\_job** returns an error.

**NEGOTIATOR\_PARALLEL\_DEFER** is ignored.

**NEGOTIATOR\_PARALLEL\_HOLD** is ignored.

**NEGOTIATOR\_RESCAN\_QUEUE** is ignored.

**NEGOTIATOR\_RECALCULATE\_SYSPRIO\_INTERVAL** works as before. Set this value to 0 if you do not want the SYSPRIOs of job objects recalculated.

---

## User exits

This section discusses separate user exits for the following:

- Handling DCE security credentials
- Handling an AFS token
- Filtering a job script
- Overriding the default mail notification method

**Note:** Other user exits are available with functions limited to specific APIs.

## 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 following keyword in your configuration file:

**DCE\_AUTHENTICATION\_PAIR** = *program1*, *program2*

Where *program1* and *program2* are LoadLeveler or installation supplied programs that are used to authenticate DCE security credentials. *program1* is the path to a program that obtains a handle (an opaque credentials object), at the time the job is submitted, which is used to authenticate to DCE. *program2* is the path to a LoadLeveler or an installation supplied program that uses the opaque credentials (handle) obtained by *program1* to authenticate to DCE before starting the job on the executing machines.

One pair of LoadLeveler programs may be specified to provide this function. You may choose from one of the following pairs provided with LoadLeveler. If you specify **DCE\_ENABLEMENT=TRUE**, LoadLeveler uses the default program pair:

```
DCE_AUTHENTICATION_PAIR = $(BIN)/lldelegate, $(BIN)/llimpersonate
```

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

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

**Note:** These programs are designed to work in pairs. You cannot specify a program from one pair and a program from the other pair.

If **DCE\_ENABLEMENT=FALSE** is specified, DCE credential forwarding will not take place by default in this case.

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.

### lldelegate and llimpersonate

The program pair, **lldelegate** and **llimpersonate**, forwards DCE credentials using a technique referred to as credential forwarding. This technique is implemented using DCE API calls to forward the user's credentials from the **lldelegate** process. The **submit** process invokes the **lldelegate** process (through the **llsubmit** command or the **submit** API) to the **llimpersonate** process invoked by the LoadLeveler **starter** process running on the machines which will execute the user's program. This method of credential forwarding depends on the user obtaining a forwardable credential prior to invoking **llsubmit** or a program using the **submit** API (such as POE). The user can obtain forwardable credentials by specifying the -f flag when invoking either **dce\_login** or **kinit**.

Specification of the **lldelegate/llimpersonate** pair requires that LoadLeveler use SP Security Services, and the ssp.clients 3.2.0.3 or higher SP Authenticated Clients Commands fileset of the PSSP install image. These command filesets require AIX 4.3.3.10 or higher and DCE 3.1. You must also configure LoadLeveler to exploit DCE security.

**Note:** This program pair is not fully functional unless DCE is at level 3.1.0.3 or higher. With DCE levels lower than 3.1.0.3, DCE credentials forwarded by this program pair are not capable of being subsequently forwarded by rsh commands.

## User exits

### **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 following keyword in your configuration file:

**AFS\_GETNEWTOKEN = *myprog***

Where *myprog* is a filter 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.

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**.

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 user exit.

The user 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. This program can, for example, modify defaults or perform site specific verification of parameters. To invoke the program, specify the following keyword in your configuration file:

**SUBMIT\_FILTER = *myprog***

Where *myprog* is called with the job file as the standard input. The standard output is submitted to LoadLeveler. If the program returns with a non-zero exit code, the job submission is canceled.

The following environment variables are set 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

## User exits

### Using your own mail program

You can write a program to override the LoadLeveler default mail notification method. You can use this program to, for example, 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 a list of users as arguments, and the mail message is taken from standard input. This syntax is as follows:

**MAIL = *program***

Where *program* specifies the path name of a local program you want to use.

### Writing prolog and epilog programs

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

Prolog and epilog programs fall into two categories: those that run as the LoadLeveler user ID, and those that run in a user's environment.

To specify prolog and epilog programs, specify the following keywords in the configuration file:

**JOB\_PROLOG = *pathname***

Where *pathname* is the full path name of the prolog program. This program runs under the LoadLeveler user ID.

**JOB\_EPILOG = *pathname***

Where *pathname* is the full path name of the epilog program. This program runs under the LoadLeveler user ID.

**JOB\_USER\_PROLOG = *pathname***

Where *pathname* is the full path name of the user prolog program. This program runs under the user's environment.

**JOB\_USER\_EPILOG = *pathname***

Where *pathname* is the full path name of the user epilog program. This program runs under the user's environment.

A user environment prolog or epilog runs with AFS authentication, DCE authentication, or both (if either is 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 below sets the environment variable **STAGE\_HOST** for the job:

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

## 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 46.

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
. /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")
```

## User exits

```
        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/loadl/.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
```

## 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 46. 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.

**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
```

## User exits

```
    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
```

## User exits

---

## Chapter 16. Procedures

---

### Using the Graphical User Interface

This section describes tasks a user may need to accomplish through the Graphical User Interface (GUI). Although this procedure is presented step-by-step, you do not have to follow the steps in the order listed. You may perform certain tasks before others without any difficulty however some tasks must be performed prior to others in order 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 section are listed in Table 16.

*Table 16. User tasks available from the GUI*

Task	For more information see page:
Building jobs	"Step 1: Building jobs"
Editing job command files	"Step 2: Edit the job command file" on page 301
Submitting job command files	"Step 3: Submit a job command file" on page 303
Obtaining job status and updating the display	"Step 4: Display, refresh, and obtain job status" on page 303
Sorting the Jobs window	"Step 5: Sort the Jobs window" on page 304
Change job priority	"Step 6: Change priorities of jobs in a queue" on page 305
Placing a job on hold	"Step 7: Hold a job" on page 305
Releasing a job on hold	"Step 8: Release a hold on a job" on page 305
Cancelling a job	"Step 9: Cancel a job" on page 305
Display and refresh machine status	"Step 12: Display and refresh machine status" on page 306
Sorting the Machines window	"Step 13: Sort the Machines window" on page 307
Finding the central manager location	"Step 14: Find the location of the central manager" on page 308
Finding the location of the public scheduling machines	"Step 15: Find the location of the public scheduling machines" on page 308
Specifying which jobs appear in the Jobs window	"Step 17: Specify which jobs appear in the Jobs window" on page 308
Specifying which machines appear in Machines window	"Step 18: Specify which machines appear in Machines window" on page 309
Saving LoadLeveler messages in a file	"Step 19: Save LoadLeveler messages in a file" on page 310

### Step 1: Building jobs

From the Jobs window:

## Using the GUI

**SELECT**

**File → Build a Job**

▲ The dialog box shown in Figure 29 appears:

Build a Job			
Tools Edit			Help
Executable			
Arguments			
Stdin	/dev/null		
Stdout	/dev/null		
Stderr	/dev/null		
Initialdir	/u/ptrimble		
Notify User	ptrimble@c94n03.ppd.pok.ibm.com		
Start Date mm/dd/yyyy			
Start Time hh:mm:ss			
Priority			
Image Size			
Class	No_Class	Chc	
Hold	user		▼
Account Number			
Environment			
Shell			
Group			▼

Job Type	Notification	Restart	Same Nodes
<input checked="" type="radio"/> Serial	<input checked="" type="radio"/> Always	<input checked="" type="radio"/> No	<input checked="" type="radio"/> No
<input checked="" type="radio"/> Parallel	<input checked="" type="radio"/> Complete	<input checked="" type="radio"/> Yes	<input checked="" type="radio"/> Yes
<input checked="" type="radio"/> PVM	<input checked="" type="radio"/> Error		
	<input checked="" type="radio"/> Never		
	<input checked="" type="radio"/> Start		

Checkpoint	Start From Ckpt
<input checked="" type="radio"/> No	<input checked="" type="radio"/> No
<input checked="" type="radio"/> Yes	<input checked="" type="radio"/> Yes
<input checked="" type="radio"/> Interval	

Nodes
Network
Requirements
Resources
Preferences
Limits
PVM
Checkpoint Fields

Submit
Save
Close

Figure 29. LoadLeveler build a job window

Complete those fields for which you want to override what is currently specified in your **skel.cmd** defaults file. A sample **skel.cmd** file is found in **/usr/LoadL/full/samples**. 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**.

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 <b>/dev/null</b> .
Stdout	Filename to use as standard output (stdout) by the program.  Optional. The default is <b>/dev/null</b> .
Stderr	Filename to use as standard error (stderr) by the program.  Optional. The default is <b>/dev/null</b> .
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 mm/dd/yyyy. 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 hh:mm:ss. The job will not start before this time.  Optional. The default is to run the job as soon as possible.  If you specify <i>StartTime</i> but not <i>StartDate</i> , the default <i>StartDate</i> is the current day. If you specify <i>StartDate</i> but not <i>StartTime</i> , the default <i>StartTime</i> is 00:00:00. This means that the job will start as soon as possible on the specified date.
Priority	Number between 0 and 100, inclusive.  Optional. The default is 50.  This is the user priority. For more information on this priority, refer to “Setting and changing the priority of a job” on page 44.
Image size	Number in kilobytes that reflects the maximum size you expect your program to grow to as it runs.  Optional.

## Using the GUI

Field	Input
Class	Class name. The job will only run on machines that support the specified class name. Your system administrator defines the class names.  Optional: <ul style="list-style-type: none"> <li>Press the Choices button to get a list of available classes.</li> <li>Press the Details button under the class list to obtain long listing information about classes.</li> </ul>
Hold	Hold status of the submitted job. Permitted values are: <b>user</b> User hold <b>system</b> System hold (only valid for LoadLeveler administrators) <b>usersys</b> User and system hold (only valid for LoadLeveler administrators)  <b>Note:</b> The default is a no-hold state.
Account Number	Number associated with the job. For use with the llacctmrg and llsummary commands for acquiring job accounting data.  Optional. Required only if the <b>ACCT</b> keyword is set to <b>A_VALIDATE</b> in the configuration file.
Environment	Specifies your initial environment variables when your job starts. Separate environment specifications with semicolons.  Optional.
Shell	The name of the shell to use for the job.  Optional. If not specified, the shell used in the owner's password file entry is used. If none is specified, <b>/bin/sh</b> is used.
Group	The LoadLeveler group name to which the job belongs.  Optional.
Step Name	The name of this job step.  Optional.
Node Usage	How the node is used. Permitted values are: <b>shared</b> The node can be shared with other tasks of other job steps. This is the default. <b>not shared</b> The node cannot be shared. <b>slice not shared</b> Job will not share nodes during its time-slice.
Dependency	A Boolean expression defining the relationship between the job steps.  Optional.
Comments	Comments associated with the job. These comments help to distinguish one job from another job.  Optional.
<b>Note:</b> 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 "Chapter 11. Job command file keywords" on page 85.	

See "Chapter 11. Job command file keywords" on page 85 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.



**Parallel** Specifies a non-PVM parallel job.  
**PVM** Specifies a PVM parallel 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 "checkpoint" on page 86 for more information.  
**Interval** Yes, checkpoint the job at intervals determined by LoadLeveler. See "checkpoint" on page 86 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**

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. Depending upon which model you choose, different

## Using the GUI

fields will be available; any unavailable fields will be greyed out. LoadLeveler will assign defaults for any fields that you leave blank.

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. For more information, see “node” on page 97.  Optional. The default is one.
Max # of Nodes	Tasks Per Node Model	Maximum number of nodes required for running the parallel job. For more information, see “node” on page 97.  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. For more information, see “tasks_per_node” on page 107.  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. For more information, see “total_tasks” on page 107.  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. For more information, see “blocking” on page 85
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. For more information, see “task_geometry” on page 106

**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.

Complete those fields for which you want to specify network information. For more information, see “network” on page 96.

Field	Input
MPI/LAPI	Choose one, both, or none of these boxes to specify the MPI (Message Passing Interface) protocol, the (LAPI Low-level Application Programming Interface) protocol, both protocols, or neither protocol.  Optional.
Adapter/Network	Select an adapter name or a network type from the list.  Required for each protocol you select.
Adapter Usage	Specifies that the adapter is either shared or not shared.  Optional. The default is shared.
Communication Mode	Specifies the mode in which an SP switch adapter is used, and can be either IP (internet Protocol) or US (User Space).  Optional. The default is IP.
Communication Level	Implies the amount of memory to be allocated to each window for User Space mode. Allocation can be Low, Average, or High.

**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. 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.

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.
Machines	Machine names. The job will only run on the specified machines.  Optional.
Features	Features. The job will only run on machines with specified features.  Optional.
LoadLeveler Version	Specifies the version of LoadLeveler, in dotted decimal format, on the machine where you want the job to run. For example: 2.1.0.0 specifies that your job will run on a machine running LoadLeveler Version 2.1.0.0 or higher.  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.
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 "resources" on page 103.

## Using the GUI

**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. 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 “Limit keywords” on page 323.

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

Field	Input
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** PVM to select a PVM job.

▲ The PVM dialog box appears.

Complete those fields for which you want to specify requirements.  
Defaults are used for those fields that you leave blank.

Field	Input
Min # of Processors	Minimum number of processors required for running the PVM job.  Optional. The default is one.
Max # of Processors	Maximum number of processors required for running the PVM job.  Optional. The default is one.
Parallel Path	The directory that defines where the PVM3 executables are located.
PVM	Specifies that an adapter is used for this PVM job.
Adapter/Network	Select an adapter name or a network type from the list.  Required.
Adapter Usage	Specifies that the adapter is either shared or not shared.  Optional. The default is shared.

**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. For detailed information on specific keywords, see "Chapter 11. Job command file keywords" on page 85.

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 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.

## Step 2: Edit the job command file

There are several ways that you can edit the job command file that you just built:

1. 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.

## Using the GUI

### 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.

2. 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 following table:

To	Select
Add a step to the job command file	Add a 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
<b>Note:</b> 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 the following table:

To	Select
Name the job	Set Job Name
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

To	Do This
Save the information you entered into a file which you can submit later	<div><b>SELECT</b></div> <div><b>Save</b></div> <div>▲ A window appears prompting you to enter a job filename.</div> <div>a job filename in the text entry field.</div> <div><b>ENTER</b></div> <div><b>SELECT</b></div> <div><b>OK</b></div> <div>▲ The window closes and the information you entered is saved in the file you specified.</div>
Submit the program immediately and discard the information you entered	<div><b>SELECT</b></div> <div><b>GO TO</b></div> <div><b>Submit</b></div> <div>Step 4</div>

If you already submitted your job, go to “Step 4: Display, refresh, and obtain job status”. Otherwise, go to “Step 3: Submit a job command file”.

### Step 3: Submit a job command file

After building a job command file, you can submit it to one or more machines for processing. In addition to scripts with LoadLeveler keywords, you can also submit scripts that contain NQS options. You cannot, however, in this release of LoadLeveler, combine NQS and LoadLeveler options.

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.

Go to the next step.

### Step 4: Display, refresh, and obtain 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:

## Using the GUI

### 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.

For more information on requests for job information, see “llq - Query job status” on page 173.

Go to the next step.

## Step 5: Sort 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

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.

Go to the next step.



## Step 6: Change priorities of jobs in a queue

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 44. 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.

Go to the next step.

## Step 7: Hold a job

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.

Go to the next step.

## Step 8: Release a 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.

Go to the next step.

## Step 9: Cancel 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**

## Using the GUI

▲ LoadLeveler cancels the job and the job information disappears from the Jobs window.

Go to the next step.

### Step 10: Modify consumable CPUs and consumable memory

Modifies the consumable CPUs or memory requirements of a non-running job.

**SELECT**

**Modify → Consumable CPUs**

or

**Modify → Consumable memory**

▲ A dialog box appears prompting you to enter a number representing the new value for consumable CPUs or consumable memory.

**TYPE IN**

The new value

**SELECT**

**OK**

▲ The dialog box closes and the value you specified takes effect.

### Step 11: Take 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.

### Step 12: Display and refresh 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 → 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

Go to the next step.

## Step 13: Sort 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

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.

Go to the next step.

## Using the GUI

### Step 14: Find 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.

Go to the next step.

### Step 15: Find 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.

Go to the next step.

### Step 16: Find 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
  - Gang
  - External (API)

### Step 17: Specify 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.

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.

To Display	Select Select →
All jobs submitted to a specific machine (or machines)	<b>By Machine</b> ▲ 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)	<b>By Group</b> ▲ 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	<b>By Job Id</b> 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.
<b>Note:</b> 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 ( <b>k10</b> ) to display all machines beginning with the characters "k10".	

**SELECT**      **Select → Show Selection** to show the selection parameters.

Go to the next step.

## Step 18: Specify which machines appear in Machines window

You can specify which machines will appear in the Machines window. The default is to view all of the machines in the LoadLeveler pool.

From the Machines window:

To	Select Select →
View all of the machines	<b>All</b>
View machines by operating system	<b>by OpSys</b> ▲ A window appears prompting you to enter the operating system of those machines you want to view.
View machines by hardware architecture	<b>by Arch</b> ▲ A window appears prompting you to enter the hardware architecture of those machines you want to view.

## Using the GUI

To	Select Select →
View machines by state	<b>by State</b>  ▲ 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.

Go to the next step.

### Step 19: Save 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.

---

## Customizing the administration file

This section tells you how to modify the administration file in a step-by-step manner. However, you do not have to perform the steps in the order they appear here.

### Step 1: Specify machine stanzas

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

## Customizing the administration file

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 on automatically creating machine stanzas for an SP system, see "llexSDR - Extract adapter information from the SDR" on page 155.

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

```
label: type = machine
adapter_stanzas = stanza_list
alias = machine_name
central_manager = true | false | alt
cpu_speed_scale = true | false
dce_host_name = dce hostname
machine_mode = batch | interactive | general
master_node_exclusive = true | false
master_node_exclusive is ignored by the Gang scheduler
max_adapter_windows = [all | none | <+> n | -n ]
max_jobs_scheduled = number
max_smp_tasks = number
name_server = list
pvm_root = pathname
pool_list = pool_numbers
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 30. Format of a machine stanza

You can specify the following keywords in a machine stanza:

**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. All adapter stanzas you define must be specified on this keyword.

**alias = machine\_name**

Where *machine\_name* is a blank-delimited list of one or more machine names. Depending upon your network configurations, you may need to add **alias** keywords for machines that have multiple interfaces.

## Customizing the administration file

Note: In general, if your cluster is configured with machine hostnames which match the hostnames 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:

```
bmw: type = machine
alias = rover
```

### **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 440. 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 “Step 10: Specify alternate central managers” on page 350.

#### **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 “Task 5: Specifying machines and their weights” on page 375.

#### **dce\_host\_name = dce hostname**

Where *dce hostname* is the DCE hostname of this machine. Execute either “**SDRGetObjects Node dcehostname,**” or “**llexstSDR**” to obtain a listing of DCE hostnames of nodes on an SP system.

#### **machine\_mode = batch | interactive | general**

Specifies the type of job this machine can run. 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.

#### **master\_node\_exclusive = true| false**

Where **true** specifies that this machine is used only as a master node for parallel jobs.

**Note:** **master\_node\_exclusive** is ignored by the Gang scheduler.

#### **max\_adapter\_windows = [ all | none | <+>n | -n ]**

This keyword specifies how many of a machine’s available adapter windows LoadLeveler can use. The default value is **all**, which specifies that LoadLeveler can reserve all of the windows which are not already reserved by other applications. The value **none** indicates that LoadLeveler can not use any windows (consequently, no user space jobs will be dispatched to that machine). A positive number (specified, with or without the plus sign), means that LoadLeveler can use no more than the specified number of windows; however, LoadLeveler may use less than the specified number if fewer windows are actually available on the machine’s adapter. A negative number means that LoadLeveler will use all but the specified number of the available windows (e.g., **-n** means that LoadLeveler will reserve *n* windows for use by other applications).

#### **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. The default is -1, which means there is no maximum.

#### **max\_smp\_tasks = number**

Specifies the maximum number of tasks that a machine can run concurrently. This value must be less than or equal to the number of processors in the

## Customizing the administration file

machine. If this keyword is not specified or a non-positive integer is specified, the default will be used which is the number of processors in the machine.

**Note:** This keyword is used by Gang scheduling only.

### **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.

### **pvm\_root = *pathname***

Where *pathname* specifies the location of the directory in which PVM is installed. The default pathname is **\$HOME/pvm3**.

### **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 may be any positive integers including zero. This keyword provides compatibility with function that was previously part of the Resource Manager.

### **resources = *name(count) name(count) ... name(count)***

Specifies quantities of the consumable resources initially available on the machine. 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, with three exceptions: **ConsumableCpus**, and **ConsumableMemory** must be specified with a value which is greater than zero, and **ConsumableVirtualMemory** must be specified with a value greater than 0, and greater than or equal to the **image\_size**. 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 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** may 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.

### **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 in conjunction 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?” in the *LoadLeveler Diagnosis and Messages Guide*.

### **schedd\_host = true | false**

Where **true** designates this as a public scheduling machine used to receive job submissions from submit-only machines, or for accepting jobs from machines which run startd but not schedd daemons. Submit-only machines do not run LoadLeveler jobs.

### **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.

### **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 “Step 7: Prioritize the order of executing machines maintained by the negotiator” on page 345. 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. The default is 1.0.

The following example illustrates how the **speed** keyword can be used for assigning weights to machines.

If your cluster consisted of five RISC System/6000 machines that you want to have the same weight, you would not have to specify this keyword in the administration file. By default, all machines would have a weight of 1.0. If, however, you add an SP system to your cluster for parallel job processing, you may want to update the local configuration file for each node of the SP system to charge differently for resource consumption on those nodes. You would need to set the **speed** keyword to something other than 1.0 to make the SP nodes have a different weight.

For information on how the **speed** keyword can be used to schedule machines, refer to “Step 7: Prioritize the order of executing machines maintained by the negotiator” on page 345.

### **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**.

## Customizing the administration file

### Examples of 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, has an alias associated with it, and can run parallel PVM jobs:

```
#
machine_c: type = machine
central_manager = true    # central manager runs here
schedd_host = true       # defines a public scheduler
alias = brianne
pvm_root = /u/brianne/load1/1.2.0/aix32/pvm3
```

## Step 2: Specify user stanzas

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 stanzas take the following format:

You can specify the following keywords in a user stanza:

```
label: type = user
account = list
default_class = list
default_group = group name
default_interactive_class = class name
maxidle = number
maxjobs = number
maxqueued = number
max_node = number
max_processors = number
max_total_tasks = number
priority = number
total_tasks = number
```

Figure 31. Format of a user stanza

#### **account =list**

Where *list* is a blank-delimited list of account numbers that identifies the account numbers a user may use when submitting jobs. The default is a null list.

#### **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.

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 = group\_name**

Where *group\_name* is the default group assigned to jobs submitted by the user. 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**.

If you specify **default\_group = Unix\_Group**, LoadLeveler sets the user's LoadLeveler group to his or her primary UNIX group (as defined in the `/etc/passwd` file).

### **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. You can specify only one default interactive class name.

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 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 "Example 2" on page 319 for more information on how LoadLeveler assigns a default interactive class to jobs.

### **maxidle = number**

Where *number* is the maximum number of idle jobs this user can have in queue. That is, *number* is the maximum number of jobs which the negotiator will consider for dispatch for the user. Jobs above this maximum are placed in the `NotQueued` state. This prevents individual users from dominating the number of jobs that are either running or are being considered to run. If the user stanza does not specify **maxidle** or if there is no user stanza at all, the maximum number of jobs that can be simultaneously in queue for the user 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.

For more information, see "Controlling the mix of idle and running jobs" on page 444.

### **maxjobs = number**

Where *number* is the maximum number of jobs this user can run at any time. If the user stanza does not specify **maxjobs** or if there is no user stanza at all,

## Customizing the administration file

the maximum jobs that can be simultaneously run by the user is defined in the default stanza. The default is -1, which means no limit is placed on the number of jobs that can simultaneously run for the user. Regardless of this limit, there is no limit to the number of jobs a user can submit.

For more information, see “Controlling the mix of idle and running jobs” on page 444.

### **maxqueued = *number***

Where *number* is the maximum number of jobs allowed in the queue for this user. This is the maximum number of jobs which can be either running or being considered to be dispatched by the negotiator for that user. Jobs above this maximum are placed in the NotQueued state. This prevents individual users from dominating the number of jobs that are either running or are being considered to run. If no **maxqueued** is specified in the user stanza, or if there is no user stanza, the maximum number of jobs that can simultaneously be in the queue is defined in the default stanza. The default is -1, which means that no limit is placed on the number of jobs that can simultaneously be in the job queue for that user. Regardless of this limit, there is no limit to the number of jobs a user can submit.

For more information, see “Controlling the mix of idle and running jobs” on page 444.

### **max\_node = *number***

Where *number* specifies the maximum number of nodes this user can request for a parallel job in a job command file using the **node** keyword. The default is -1, which means there is no limit. The **max\_node** keyword will not affect the use of the **min\_processors** and **max\_processors** keywords in the job command file.

### **max\_processors = *number***

Where *number* specifies the maximum number of processors this user can request for a parallel job in a job command file using the **min\_processors** and **max\_processors** keywords. The default is -1, which means there is no limit.

### **max\_total\_tasks = *number***

Specifies the maximum total number of tasks that the scheduler will allocate at any given time to run the jobs of this user. The default value for this keyword is -1 which is unlimited.

**Note:** This keyword is used by Gang scheduling only.

### **priority = *number***

Where *number* is a integer that specifies the priority for jobs submitted by the user. The default is 0. The number specified for priority is referenced as **UserSysprio** in the configuration file. **UserSysprio** can be used in the assignment of job priorities. If the variable **UserSysprio** does not appear in the SYSPRIO expression in the configuration file, the priority numbers for users specified here in the administration file have no effect. See “Step 6: Prioritize the queue maintained by the negotiator” on page 343 for more information about the **UserSysprio** keyword.

### **total\_tasks = *number***

Where *number* specifies the maximum number of tasks this user can request for a parallel job in a job command file using the **total\_tasks** keyword. The default is -1, which means there is no limit.



## Examples of user stanzas

**Example 1:** In this example, user fred is being provided with a user stanza. His jobs will have a user priority of 100. If he does not specify a job class in his 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
        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
```

## Step 3: Specify class stanzas

The information in a class stanza defines characteristics for that class. Class stanzas are optional. Class stanzas take the following format. Default values for keywords appear in bold.

## Customizing the administration file

```
label: type = class
admin= list
ckpt_dir = directory
ckpt_time_limit = hardlimit,softlimit
class_comment = "string"
default_resources = name(count) name(count)...name(count)
exclude_groups = list
exclude_users = list
execution_factor = number
include_groups = list
include_users = list
master_node_requirement = true | false
maxjobs = number
max_node = number
max_processors = number
max_total_tasks = number
nice = value
NQS_class = true | false
NQS_submit = name
NQS_query = queue names
priority = number
total_tasks = number
core_limit = hardlimit,softlimit
cpu_limit = hardlimit,softlimit
data_limit = hardlimit,softlimit
file_limit = hardlimit,softlimit
job_cpu_limit = hardlimit,softlimit
rss_limit = hardlimit,softlimit
stack_limit = hardlimit,softlimit
wall_clock_limit = hardlimit,softlimit
```

Figure 32. Format of a class stanza

You can specify the following keywords in a class stanza:

### **admin = list**

Where *list* is a blank-delimited list of administrators for this class. These administrators can hold, release, and cancel jobs in this class.

### **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 358.

### **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 (:) character. The length of the string cannot exceed 1024 characters.

### **default\_resources = name(count) name(count)...name(count)**



## Customizing the administration file

Specifies the default amount of resources consumed by a task of a job step, of this class, 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.

The administrator defines the name and count values for **default\_resources**. In addition, *name(count)* 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, with three exceptions: **ConsumableCpus**, and **ConsumableMemory** must be specified with a value which is greater than zero, and **ConsumableVirtualMemory** must be specified with a value greater than 0, and greater than or equal to the **image\_size** (units for image\_size are in kilobytes). If the count is not valid, then LoadLeveler will issue an error message, and will not submit the job. 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.

### **exclude\_groups = list**

Where *list* is a blank-delimited list of groups who are *not* allowed to submit jobs of that *class name*. 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. The default is that no groups are excluded.

### **exclude\_users = list**

Where *list* is a blank-delimited list of users who are *not* permitted to submit jobs of that *class name*. 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. The default is that no users are excluded.

### **execution\_factor = number**

Specifies how much processing time jobs of this class will receive relative to other jobs operating on the same node. For example, if job A has an *execution\_factor* of 2 and job B has an *execution\_factor* of 1, then LoadLeveler will allocate twice the number of rows in the Gang matrix (and therefore twice the amount of processing time) to job A as to job B. The range of values for this keyword are 1, 2, or 3 and the default is 1.

## Customizing the administration file

**Note:** This keyword is used by Gang scheduling only.

### **include\_groups = *list***

Where *list* is a blank-delimited list of groups who are allowed to submit jobs of that *class name*. If provided, this list limits groups 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. The default is to include all groups.

### **include\_users = *list***

Where *list* is a blank-delimited list of users who are permitted to submit jobs of that *class name*. If provided, this list limits users of that class to those on the list. 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. The default is to include all users.

### **master\_node\_requirement = true|false**

Where **true** specifies that parallel jobs in this class 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.

**Note:** **master\_node\_requirement** is ignored by Gang scheduler.

### **maxjobs = *number***

Where *number* is the maximum number of jobs that can run in this class. If the class stanza does not specify **maxjobs**, or if there is no class stanza at all, the maximum jobs that can be simultaneously run in this class is defined in the default stanza. The default is -1, which means that no limit is placed on the number of jobs a user can submit.

### **max\_node = *number***

Where *number* specifies the maximum number of nodes a user submitting jobs in this class can request for a parallel job in a job command file using the **node** keyword. The default is -1, which means there is no limit. The **max\_node** keyword will not affect the use of the **min\_processors** and **max\_processors** keywords in the job command file.

### **max\_processors = *number***

Where *number* specifies the maximum number of processors a user submitting jobs to this class can request for a parallel job in a job command file using the **min\_processors** and **max\_processors** keywords. The default is -1 which means that there is no limit.

### **max\_total\_tasks = *number***

Specifies the maximum total number of tasks that the scheduler will allocate at any given time to run the jobs of this class. The default value for this keyword is -1 which is unlimited.

**Note:** This keyword is used by Gang scheduling only.

### **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.

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.

### **NQS\_class = true|false**

When **true**, any job submitted to this class will be routed to an NQS machine.

### **NQS\_submit = name**

Where *name* is the name of the NQS pipe queue to which the job will be routed. When the job is dispatched to LoadLeveler, LoadLeveler will invoke the **qsub** command using the name of this queue. There is no default.

### **NQS\_query = queue names**

Where *queue names* is a blank-delimited list of queue names (including host names if necessary) to be used with the **qstat** command to monitor the job and with the **qdel** command to cancel the job. There is no default.

For more information on routing jobs to machines running NQS, refer to Figure 18 on page 79

### **priority = number**

Where *number* is an integer that specifies the priority for jobs in this class. The default is 0. The number specified for priority is referenced as **ClassSysprio** in the configuration file. You can use **ClassSysprio** when assigning job priorities. If the variable **ClassSysprio** does not appear in the SYSPRIO expression, then the priority specified here in the administration file is ignored. See “Step 6: Prioritize the queue maintained by the negotiator” on page 343 for more information about the **ClassSysprio** keyword.

### **total\_tasks = number**

Where *number* specifies the maximum number of tasks a user submitting jobs in this class can request for a parallel job in a job command file using the **total\_tasks** keyword. The default is -1, which means there is no limit.

## **Limit keywords**

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

Table 17. Types of limit keywords

Limit	How It Is Enforced
<b>ckpt_time_limit</b>	Per job step
<b>core_limit</b>	Per process
<b>cpu_limit</b>	Per process
<b>data_limit</b>	Per process
<b>file_limit</b>	Per process
<b>job_cpu_limit</b>	Per job step
<b>rss_limit</b>	Per process

## Customizing the administration file

Table 17. Types of limit keywords (continued)

Limit	How It Is Enforced
<b>stack_limit</b>	Per process
<b>wall_clock_limit</b>	Per job step

Individual keywords are described in “Specifying limits in the class stanza” on page 326. The following section gives you a general overview of limits.

**Overview of limits:** 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. For more information, see “Exceeding limits”.

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.

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” on page 94.

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.

**Exceeding limits:** Process limits are enforced by the operating system. Job step limits are enforced by LoadLeveler.

**Exceeding job step limits:** When a hard limit is exceeded LoadLeveler sends a *non-trappable* signal to the process (except in the case of a parallel job). When a soft limit is exceeded, LoadLeveler sends a *trappable* signal to the process. The following chart summarizes the actions that occur when a job step limit is exceeded:

Table 18. Exceeding 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
Parallel (non-PVM)	SIGXCPU issued to both the user program and to the parallel daemon	SIGTERM issued
PVM	SIGXCPU issued to the user program	<b>pvm_halt</b> invoked to shut down PVM

## Customizing the administration file

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.

*Exceeding 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.

Note that when a job forks a process which exceeds a per process limit, such as the CPU limit, the operating system (and 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.

**Syntax:** 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.

## Customizing the administration file

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.

See Table 19 for more information on specifying limits.

Table 19. Setting limits

If the hard limit:	Then the:
Is 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.
Is 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.
Is set in the job command file and is less than its respective job soft limit	The job is not submitted.
Is 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.
Is 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.

**Specifying limits in the class stanza:** You can specify the following limit keywords:

**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 abort 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.

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
```

### **core\_limit = *hardlimit,softlimit***

Specifies the hard limit, soft limit, or both for the size of a core file.

Examples:

```
core_limit = unlimited
core_limit = 30mb
```

For more information, see “Overview of limits” on page 324

### **cpu\_limit = *hardlimit,softlimit***

Specifies hard limit, soft limit, or both for the CPU time to be used by each individual process of a job step. 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 more information, see “Overview of limits” on page 324.

### **data\_limit = *hardlimit,softlimit***

Specifies hard limit, soft limit, or both for the data segment to be used by each process of the submitted job.

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 more information, see “Overview of limits” on page 324.

### **file\_limit = *hardlimit,softlimit***

Specifies the hard limit, soft limit, or both for the size of a file. For more information, see “Overview of limits” on page 324.

### **job\_cpu\_limit = *hardlimit,softlimit***

Specifies the maximum total CPU time to be used by all processes of a job step.

For example:

```
job_cpu_limit = 10000
```

For more information on this keyword, see:

- “job\_cpu\_limit” on page 94
- **JOB\_LIMIT\_POLICY** on page 75
- For general information on limits, see “Overview of limits” on page 324



## Customizing the administration file

**rss\_limit = hardlimit,softlimit**

Specifies the hard limit, soft limit, or both for the resident size. For more information, see “Overview of limits” on page 324.

**stack\_limit = hardlimit,softlimit**

Specifies the hard limit, soft limit, or both for the size of a stack. For more information, see “Overview of limits” on page 324.

**wall\_clock\_limit = hardlimit,softlimit**

Specifies the hard limit, soft limit, or both for the elapsed time for which a job can run. 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. This keyword is not supported for NQS jobs.

If you are running the Backfill or Gang 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 or Gang scheduler, see “Choosing a scheduler” on page 335.

For more general information on limits, see “Overview of limits” on page 324.

## Examples of 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
core_limit=10mb              # max 10 MB core file
file_limit=50mb              # max file size 50 MB
stack_limit=10mb             # max stack size 10 MB
rss_limit=35mb               # max resident set size 35 MB
include_users = bob sally    # authorized users
```

### *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 by each
ConsumableCpus(2) resA(8) floatinglicenseX(1) resB(1) # 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 to route jobs to NQS machines:**

```
nqs: type=class            # class for NQS jobs
NQS_class=true
NQS_submit=pipe_queue     # NQS pipe queue name
NQS_query=one two three   # list of queue names
```

You can use the class names in control expressions in both the global and local configuration file.

### **Example 6: Creating a class for PVM jobs:**

```
PVM3: type=class          # class for PVM jobs
priority=60               # ClassSysprio (max=100)
max_processors=15         # maximum number of processors
```

### **Example 7: 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
```

## Step 4: Specify group stanzas

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, which is explained in this section.

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

## Customizing the administration file

```
label: type = group
admin = list
exclude_users = list
include_users = list
maxidle = number
maxjobs = number
maxqueued = number
max_node = number
max_processors = number
max_total_tasks = number
priority = number
total_tasks = number
```

Figure 33. Format of a group stanza

You can specify the following keywords in a group stanza:

**admin = list**

Where *list* is a blank-delimited list of administrators for this group. These administrators can hold, release, and cancel jobs submitted by users in the group.

**exclude\_users =list**

Where *list* is a blank-delimited list of users that 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 group. The default is that no users will be excluded.

**include\_users =list**

Where *list* is a blank-delimited list of users that belong to the group. If provided, this list limits users of that group to those on the list. Do not specify both a list of included users and a list of excluded users. Only one of these can be used for any group. The default is that all users are included.

**maxidle = number**

Where *number* is the maximum number of idle jobs this group can have in queue. That is, *number* is the maximum number of jobs which the negotiator will consider for dispatch for this group. Jobs above this maximum are placed in the NotQueued state. This prevents groups from flooding the job queue. If the group stanza does not specify **maxidle** or if there is no group stanza at all, the maximum number of jobs that can be simultaneously in queue for the group is defined in the default stanza. The default is -1, which means that no limit is placed on the number of jobs that can be simultaneously idle for the group.

For more information, see “Controlling the mix of idle and running jobs” on page 444.

**maxjobs = number**

Where *number* is a maximum number of jobs this group can run at any time. If the group stanza does not specify the **maxjobs** or if there is no group stanza at all, the maximum number of jobs that can be simultaneously run the group is defined in the default stanza. The default is -1, which means that no limit is placed on the number of jobs that can be simultaneously run for the group. Regardless of the limit set to running jobs, there is no limit to the number of jobs that a group can submit.

For more information, see “Controlling the mix of idle and running jobs” on page 444.

**maxqueued = number**

Where *number* is the maximum number of jobs allowed in the queue for this

## Customizing the administration file

group. This prevents groups from flooding the job queue. Jobs above this maximum are placed in the NotQueued state. If no **maxqueued** is specified in the group stanza, or if there is no group stanza, the maximum number of jobs that can simultaneously be in the queue is defined in the default stanza. The default is -1, which means that no limit is placed on the number of jobs that can simultaneously be in the job queue for that group. Regardless of the limit set to the number of jobs queued, there is no limit to the number of jobs a group can submit.

For more information, see “Controlling the mix of idle and running jobs” on page 444.

### **max\_node = number**

Where *number* specifies the maximum number of nodes a user can request for a parallel job in a job command file using the **node** keyword. The default is -1, which means there is no limit. The **max\_node** keyword will not affect the use of the **min\_processors** and **max\_processors** keywords in the job command file.

### **max\_processors = number**

Where *number* specifies the maximum number of processors a user can request for a parallel job in a job command file using the **min\_processors** and **max\_processors** keywords. The default is -1, which means there is no limit.

### **max\_total\_tasks = number**

Specifies the maximum total number of tasks that the scheduler will allocate at any given time to run the jobs of this group. The default value for this keyword is -1 which is unlimited.

**Note:** This keyword is used by Gang scheduling only.

### **priority = number**

Where *number* is an integer that specifies the job priority for jobs associated with this group. The higher priority numbers result in a better job dispatch order. If the group stanza does not specify a priority or if there is no priority at all, the priority is defined in the default group stanza. The default priority is 0. The number specified for priority is referenced as **GroupSysprio** in the configuration file. **GroupSysprio** can be used in the assignment of job priorities. If the variable **GroupSysprio** does not appear in the SYSPRIO expression in the configuration file, the priority numbers for group specified in the administration file have no effect. See “Step 6: Prioritize the queue maintained by the negotiator” on page 343 for more information about the **GroupSysprio** keyword.

### **total\_tasks = number**

Where *number* specifies the maximum number of tasks a user specifying this group can request for a parallel job in a job command file using the **total\_tasks** keyword. The default is -1, which means there is no limit.

## Examples of 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:

## Customizing the administration file

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

## Step 5: Specify adapter stanzas

An adapter stanza identifies network adapters that are available on the machines in the LoadLeveler cluster. Adapter stanzas are optional, but you need to specify them when you want LoadLeveler jobs to be able to request a specific adapter. You do not need to specify an adapter stanza when you want LoadLeveler jobs to access a shared, default adapter via TCP/IP.

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 an SP switch adapter, the **switch\_node\_number** keyword is also required.

For information on creating adapter stanzas for an SP system, see “llexSDR - Extract adapter information from the SDR” on page 155.

An adapter stanza has the following format:

You can specify the following keywords in an adapter stanza:

```
label: type = adapter
adapter_name = name
css_type = type
interface_address = IP_address
interface_name = name
multilink_address = ip_address
multilink_list = adapter_name <, adapter_name>*
network_type = type
switch_node_number = integer
```

Figure 34. Format of an adapter stanza

### **adapter\_name = string**

Where *string* is the name used to refer to a particular interface card installed on the node. Some examples are **en0**, **tk1**, and **css0**. Whenever a machine has one or more adapters with a name that starts with **css** (eg. **css0** or **css1**), a virtual adapter named **csss** is created for that machine. This adapter is used on the network statement when a job requires striped communication. This keyword defines the adapters a user can specify in a job command file using the **network** keyword. This keyword is required.

### **css\_type = type**

Where *type* is the designation for the type of switch adapter to be used. The allowable choices are: **SP\_Switch\_Adapter**, **SP\_Switch\_MX\_Adapter**, **SP\_Switch\_MX2\_Adapter**, **RS/6000\_SP\_System\_Attachment\_Adapter**, and **SP\_Switch2\_Adapter**. This keyword must be specified in combination with a switch adapter ("**css . . .**"), otherwise it will be ignored. The **css\_type** attribute for the available adapters are defined in the SDR. Execute the command **SDRGetObjects Adapter css\_type** to obtain a list of **css\_types**, or use **llexSDR** to obtain all of the adapter information from the SDR.

**Note:** `css_type` cannot be `RS/6000_SP_System_Attachment_Adapter` or `SP_Switch_Adapter` if you are using Gang scheduling. For more information see “Restrictions for Gang scheduling and preemption” on page 392.

**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 = *string***

Where *string* is the name by which the adapter is known by other nodes in the network. This keyword is required.

**multilink\_address = *ip\_address***

Where *ip\_address* indicates the IP address that includes the adapters that can be striped across.

**multilink\_list = *adapter\_name* <, *adapter\_name*>\***

Where *adapter\_name* indicates multilinked devices which stripes IP addresses across the adapters given in the list.

**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 (for example, `css0` and `css1` define two different communication paths). 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.

**switch\_node\_number = *integer***

Where *integer* specifies the node on which the SP switch adapter is installed. This keyword is required for SP switch adapters. Its value is defined in the `switch_node_number` field in the Node class in the SDR. This value must match the value in the `/spdata/sys1/st/switch_node_number` file of the Parallel System Support Programs (PSSP).

### Example of an adapter stanza

**Example 1: Specifying an SP Switch adapter:** In the following example, the adapter stanza called “sp01sw.ibm.com” specifies an SP switch adapter. Note that sp01sw.ibm.com is also specified on the **adapter\_stanzas** keyword of the machine stanza for the “yugo” machine.

```
yugo:  type=machine
      adapter_stanzas = sp01sw.ibm.com
      ...

sp01sw.ibm.com: type = adapter
               adapter_name = css0
               interface_address = 12.148.44.218
               interface_name = sp01sw.ibm.com
               network_type = switch
               switch_node_number = 7
               css_type = SP_Switch_MX2_Adapter
```

---

## Customizing the global and local configuration file

This section presents a step-by-step approach to configuring LoadLeveler. However, you do not have to perform the steps in the order they appear here.

### Step 1: Define LoadLeveler administrators

Specify the following keyword:

## Customizing the configuration file

### **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. 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 “Administrative uses for the Graphical User Interface” on page 27.

LoadLeveler administrators on this list also receive mail describing problems that are encountered by the master daemon. When DCE is enabled, the **LOADL\_ADMIN** list is used only as a mailing list. For more information, see “Step 16: Configuring LoadLeveler to use DCE security services” on page 365.

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 userid in every local configuration file that exists in the LoadLeveler cluster.

For example, to grant administrative authority to users bob and mary, enter the following in the configuration file:

```
LOADL_ADMIN = bob mary
```

## Step 2: Define LoadLeveler cluster characteristics

Use the following keywords to define the characteristics of the LoadLeveler cluster:

### **CUSTOM\_METRIC = *number***

Specifies a machine's relative priority to run jobs. This is an arbitrary number which you can use in the MACHPRIO expression. Negative values are not allowed. If you specify neither **CUSTOM\_METRIC** nor

**CUSTOM\_METRIC\_COMMAND**, **CUSTOM\_METRIC = 1** is assumed. For more information, see “Step 7: Prioritize the order of executing machines maintained by the negotiator” on page 345.

### **CUSTOM\_METRIC\_COMMAND = *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.

### **MACHINE\_AUTHENTICATE = *true* | *false***

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.

**Note:** **MACHINE\_AUTHENTICATE** must be set as “true” for Gang scheduling to work. For more information see “Restrictions for Gang scheduling and preemption” on page 392.

### SCHEDULER\_TYPE and SCHEDULER\_API

The last cluster characteristic that needs to be defined is the LoadLeveler scheduler. Two keywords are available for setting this configuration and each has multiple options. SCHEDULER\_TYPE is the preferred keyword but SCHEDULER\_API is still available for migration purposes. For more information, see “Choosing a scheduler”.

### Choosing a scheduler

This section discusses the types of schedulers available and the keywords (SCHEDULER\_TYPE and SCHEDULER\_API) used to define which scheduler LoadLeveler will use.

**Scheduler keyword definitions:** Use the following keywords to define your scheduler:

#### SCHEDULER\_TYPE = LL\_DEFAULT | BACKFILL | API | GANG

This keyword sets the scheduler used by LoadLeveler. When SCHEDULER\_TYPE is defined, the obsolete keyword SCHEDULER\_API is ignored.

##### 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**.

The SCHEDULER\_TYPE definitions are:

#### 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 “Step 8: Manage a job’s status using control expressions” on page 347.

**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 “Workload Management API” on page 270.

**GANG** Specifies that you will use the LoadLeveler Gang scheduling algorithm. For more information, see “Chapter 17. Using Gang scheduling” on page 381.

#### SCHEDULER\_API = YES | NO

The SCHEDULER\_API keyword sets an external scheduler but it is now obsolete and should only be used for migration purposes. Use SCHEDULER\_TYPE=API as a replacement for SCHEDULER\_API=YES. If SCHEDULER\_API has been set to YES and SCHEDULER\_TYPE has not been defined, then SCHEDULER\_API=YES is functionally equivalent to SCHEDULER\_TYPE=API; LoadLeveler will ignore all other instances of SCHEDULER\_API. For more information on setting an external scheduler, see “Workload Management API” on page 270.



## Customizing the configuration file

**Note:** If you change the scheduler from a specified `SCHEDULER_TYPE` to `SCHEDULER_API=YES`, you must stop and restart LoadLeveler using `llctl`.

### ***SCHEDULER\_TYPE option details:***

- **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 “Keyword considerations for parallel jobs” on page 49 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. Backfilling is the capability to schedule a job that is short in duration, or which requires a small number of nodes, before a higher priority job. Any idle resources available between the current time and the earliest projected start time of the highest priority job can be used to run other waiting jobs. Jobs will only be backfilled if they will not delay the start of the higher priority job. The scheduler makes this determination by comparing the projected start time of the highest priority job with the **wall\_clock\_limit** of the potential backfilled job. If the backfilled job will end before the higher priority job’s start time, then it is eligible to run.

For example: 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.

The Backfill scheduler supports:

- The scheduling of multiple tasks per node.
- The scheduling of multiple user space tasks per adapter.

The above functions are not supported by the default LoadLeveler 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.
- You should use only the default settings for the **START** expression and the other job control functions described in “Step 8: Manage a job’s status using control expressions” on page 347. 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 “Step 7: Prioritize the order of executing machines maintained by the negotiator” on page 345.

See “Keyword considerations for parallel jobs” on page 49 for information on which keywords associated with parallel jobs are supported by the Backfill scheduler.

- **GANG** This scheduling algorithm combines coordinated context switching with both space-sharing and time-sharing strategies to support parallel applications. It generally provides good overall system utilization and responsiveness to interactive workloads. Gang scheduler differs from backfill scheduling which supports time-sharing and space-sharing but not coordinated context switching. Gang scheduler also differs from EASY scheduling (a common instance of an external scheduler) which supports space-sharing only. The default setting for Gang scheduling will satisfy most job requirements.

For more information on setting up Gang scheduling, see “Chapter 17. Using Gang scheduling” on page 381.

- **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. 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 (for more information, see “Query API” on page 265). Also, some LoadLeveler functions are not available when you use API (for more information, see “Workload Management API” on page 270 and “Usage notes” on page 281).

### Setting up file system monitoring

You can use the file system keywords to monitor the file system space 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. This is done by setting the frequency that LoadLeveler checks the file system free space and by setting the upper and lower thresholds that initialize system responses to the free space available. By setting a realistic span between the lower and upper thresholds, you will avoid excessive system actions.

#### **FS\_INTERVAL** = *seconds*

Defines the interval (in seconds) used when checking the size of the file system. 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

## Customizing the configuration file

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

**Note:** If FS\_INTERVAL is not specified but any of the other three keywords (FS\_NOTIFY, FS\_SUSPEND, or FS\_TERMINATE) are specified, the FS\_INTERVAL value will default to 5 and the file system will be checked.

### **FS\_NOTIFY** = *lower threshold, upper threshold*

This configuration file keyword defines when LoadLeveler notifies the administrator that there is a file system problem or that a file system problem has been resolved.

If the free space associated with the LoadLeveler file system drops below the lower threshold, LoadLeveler sends a mail message to the administrator indicating that logging problems may occur. When file system free space rises above the upper threshold (after passing the lower threshold), LoadLeveler sends a mail message to the administrator indicating that problem has been resolved.

Default value (in blocks): 1000, -1

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.

### **FS\_SUSPEND** = *lower threshold, upper threshold*

This configuration file keyword defines when LoadLeveler drains and resumes the schedd and startd daemons running on a node.

If the free space associated with the LoadLeveler file system drops below lower threshold, 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 file system free space rises above the upper threshold (after passing the lower threshold), 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.

Default value (in blocks): -1, -1

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.

### **FS\_TERMINATE** = *lower threshold, upper threshold*

This keyword sends the SIGTERM signal to the Master daemon which then terminates all LoadLeveler daemons running on the node.

If the free space associated with the LoadLeveler file system drops below lower threshold, all LoadLeveler daemons are terminated.

**Note:** Although the upper threshold setting for FS\_TERMINATE is ignored when LoadLeveler is terminated, the upper threshold is still required on the statement.

Default value (in blocks): -1, -1

The valid range for the lower thresholds is -1 and all positive integers. If the value is set to -1, the transition across the threshold is not checked.

### Step 3: Define LoadLeveler machine characteristics

You can use the following keywords to define the characteristics of machines in the LoadLeveler cluster:

- ARCH
- CLASS
- Feature
- START\_DAEMONS
- SCHEDD\_RUNS\_HERE
- SCHEDD\_SUBMIT\_AFFINITY
- STARTD\_RUNS\_HERE
- X\_RUNS\_HERE

#### **ARCH = *string* (required)**

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.

For example, to define a machine as a RS/6000, the keyword would look like:

```
ARCH = R6000
```

#### **CLASS = { "*class\_name*" ... } | { "**No\_Class**" } | *class\_name* (*count*) ...**

This keyword 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, it will be set 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.

You can specify a **default\_class** in the default user stanza of the administration file to set a default class. If you don't, jobs will default to the class called **No\_Class**.

In order 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.

## Customizing the configuration file

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 446 and “Step 3: Specify class stanzas” on page 319 for more information on classes.

### Defining classes – examples

**Example 1:** This example defines the default class:

```
Class = No_Class(1)
```

or

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

**This is the default.** The machine will only run one LoadLeveler job at a time that has either defaulted to, or explicitly requested class **No\_Class**. A LoadLeveler job with class **CPU\_bound**, for example, would not be eligible to run here. Only one LoadLeveler job at a time will run on the machine.

**Example 2:** 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 “Step 5: Specify how many jobs a machine can run” on page 342 for more information on **MAX\_STARTERS**.

**Example 3:** 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 4:** 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.

**Feature = {"string" ...}**

Where *string* is the (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.

For 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")
```

**START\_DAEMONS = true | false**

Specifies whether to start the LoadLeveler daemons on the node. 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**. For more information, see “the submit-only keyword” on page 315.

**SCHEDD\_RUNS\_HERE = true | false**

Specifies whether the `schedd` daemon runs on the host. If you do not want to run the `schedd` daemon, specify **false**.

To define the machine as an executing machine only, set this keyword to **false**. For more information, see “the submit-only keyword” on page 315.

**SCHEDD\_SUBMIT\_AFFINITY = true | false**

Specifies that the `lsubmit` command submits a job to the machine where the command was invoked, provided that the `schedd` daemon is running on that machine (this is called `schedd` affinity). Installations with a large number of nodes should consider setting this keyword to **false**. For more information, see “Scaling considerations” on page 443.

**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**.

**X\_RUNS\_HERE = true | false**

Set **X\_RUNS\_HERE** to **true** if you want to start the keyboard daemon.

## Step 4: Define consumable resources

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

## Customizing the configuration file

### **SCHEDULE\_BY\_RESOURCES** = *name name ... name*

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**, or **ConsumableVirtualMemory**.

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.

### **FLOATING\_RESOURCES** = *name(count) name(count) ... name(count)*

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.

### **ENFORCE\_RESOURCE\_USAGE** = **ConsumableCpus ConsumableMemory** | **deactivate**

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.

### **ENFORCE\_RESOURCE\_SUBMISSION** = **true** | **false**

If the value specified is **true**, LoadLeveler will check all jobs at submission time for the **resources** keyword. The job command file **resources** keyword needs to have at least the resources specified as the **ENFORCE\_RESOURCE\_USAGE** keyword in order 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.

## Step 5: Specify 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, which is described in this section, and the **Class** statement, which is mentioned here and described in more detail in “Step 3: Define LoadLeveler machine characteristics” on page 339

The syntax for **MAX\_STARTERS** is:

### **MAX\_STARTERS** = *number*

Where *number* specifies the maximum number of tasks that can run simultaneously on a machine. In this case, a task can be a serial job step, a parallel task, or an instance of the PVM daemon (PVMd). If not specified, the default is the number of elements in the **Class** statement. **MAX\_STARTERS** defines the number of initiators on the machine (the number of tasks that can be initiated from a **startd**).

## Customizing the configuration file

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 in conjunction 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 above 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.

With Gang scheduling, the value of **MAX\_STARTERS** divided by the value for **max\_smp\_tasks**, sets the maximum size of the Gang Matrix.

## Step 6: Prioritize the queue maintained by the negotiator

Each job 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 to the queue of jobs eligible for dispatch. Once assigned, the system priority number for a job is never changed (unless jobs for a user swap their **SYSPRIO**, or **NEGOTIATOR\_RECALCULATE\_SYSPRIO\_INTERVAL** is not zero). Jobs assigned higher **SYSPRIO** numbers are considered for dispatch before jobs with lower numbers. See “How does a job’s priority affect dispatching order?” on page 45 for more information on job priorities.

You can use the following LoadLeveler variables to define the **SYSPRIO** expression:

### **ClassSysprio**

The priority for the class of the job step, defined in the class stanza in the administration file. The default is 0.

### **GroupQueuedJobs**

The number of job steps associated with a LoadLeveler group which are



## Customizing the configuration file

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.)

### 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.

### GroupSysprio

The priority for the group of the job step, defined in the group stanza in the administration file. The default is 0.

### GroupTotalJobs

The total number of job steps associated with this LoadLeveler group. Total job steps are all job steps reported by the **llq** command.

**QDate** The difference in the UNIX date when the job step enters the queue and the UNIX date when the negotiator starts up.

### UserPrio

The user-defined priority of the job step, specified in the job command file with the **user\_priority** keyword. The default is 50.

### UserQueuedJobs

The number of job steps either running or queued for the user. (That is, job steps which are in one of these states: Checkpointing, Preempted, Preempt Pending, Resume Pending, Running, Starting, Pending, or Idle.)

### 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.

### UserSysprio

The priority of the user who submitted the job step, defined in the user stanza in the administration file. The default is 0.

### UserTotalJobs

The total number of job steps associated with this user. Total job steps are all job steps reported by the **llq** command.

## Usage notes for the SYSPRIO keyword

- 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.
- It is recommended that you do not use **UserPrio** in the **SYSPRIO** expression, since user jobs are already ordered by **UserPrio**.
- You can use the **UserRunningJobs**, **GroupRunningJobs**, **UserQueuedJobs**, **GroupQueuedJobs**, **UserQueuedJobs**, **GroupQueuedJobs** **UserTotalJobs**, and **GroupTotalJobs** parameters 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.

## Using the SYSPRIO keyword – 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:



$\text{SYSPRIO} : (\text{ClassSysprio} * 100) + (\text{UserSysprio} * 10) + (\text{GroupSysprio} * 1) - (\text{QDate})$

**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.

$\text{SYSPRIO} : 0 - \text{UserRunningJobs}$

## Step 7: 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 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 371.

You can use the following LoadLeveler variables in the **MACHPRIO** expression:

### LoadAvg

The Berkeley one-minute load average of the machine, reported by startd.

**Cpus** The number of processors of the machine, reported by startd.

**Speed** The relative speed of the machine, defined in a machine stanza in the administration file. The default is 1.

### Memory

The size of real memory in megabytes of the machine, reported by startd.

### VirtualMemory

The size of available swap space (free paging space) on the machine (in kilobytes), reported by startd.

**Disk** The size of free disk space in kilobytes on the file system where the executables reside.

### CustomMetric

Allows you to set a relative priority number for one or more machines, based on the value of the **CUSTOM\_METRIC** keyword. (See "Example 4" 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

## Customizing the configuration file

are not master nodes. Assigning a high priority to master nodes may help job scheduling performance for parallel jobs which require master node features.

### ConsumableCpus

If **ConsumableCpus** is specified in the **SCHEDULE\_BY\_RESOURCES** keyword, then this is the number of **ConsumableCpus** available on the machine. If **ConsumableCpus** is not specified in the **SCHEDULE\_BY\_RESOURCES** keyword, then this is the same as **Cpus**.

### ConsumableMemory

This is the number of megabytes of **ConsumableMemory** available on the machine, provided that **ConsumableMemory** is specified in the **SCHEDULE\_BY\_RESOURCES** keyword. If **ConsumableMemory** is not specified in the **SCHEDULE\_BY\_RESOURCES** keyword, then this is the same as **Memory**.

### ConsumableVirtualMemory

This is the number of megabytes of **ConsumableVirtualMemory** available on the machine, provided that **ConsumableVirtualMemory** is specified in the **SCHEDULE\_BY\_RESOURCES** keyword. If **ConsumableVirtualMemory** is not specified in the **SCHEDULE\_BY\_RESOURCES** keyword, then this is the same as **VirtualMemory**.

### PagesFreed

The number of pages freed per second by the page replacement algorithm of the virtual memory manager.

### PagesScanned

The number of pages scanned per second by the page replacement algorithm of the virtual memory manager.

### 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 whose units are 4K blocks.

## Using the MACHPRIO keyword – 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)
```

## Step 8: Manage a job's status using 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" }**

## Customizing the configuration file

(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.

**START:** *expression that evaluates to T or F (true or false)*

Determines whether a machine can run a LoadLeveler job. 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 371.

**SUSPEND:** *expression that evaluates to T or F (true or false)*

Determines whether running jobs should be suspended. 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.

**CONTINUE:** *expression that evaluates to T or F (true or false)*

Determines whether suspended jobs should continue execution. When **T**, suspended LoadLeveler jobs resume execution on the machine.

**VACATE:** *expression that evaluates to T or F (true or false)*

Determines whether suspended jobs should be vacated. 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.

**KILL:** *expression that evaluates to T or F (true or false)*

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. When **T**, vacated LoadLeveler jobs are removed from the machine with no attempt to take checkpoints.

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

### How control expressions affect jobs

After LoadLeveler selects a job for execution, the job can be in any of several states. Figure 35 on page 349 shows how the control expressions can affect the state a job is in. The rectangles represent job or daemon states, and the diamonds represent the control expressions.

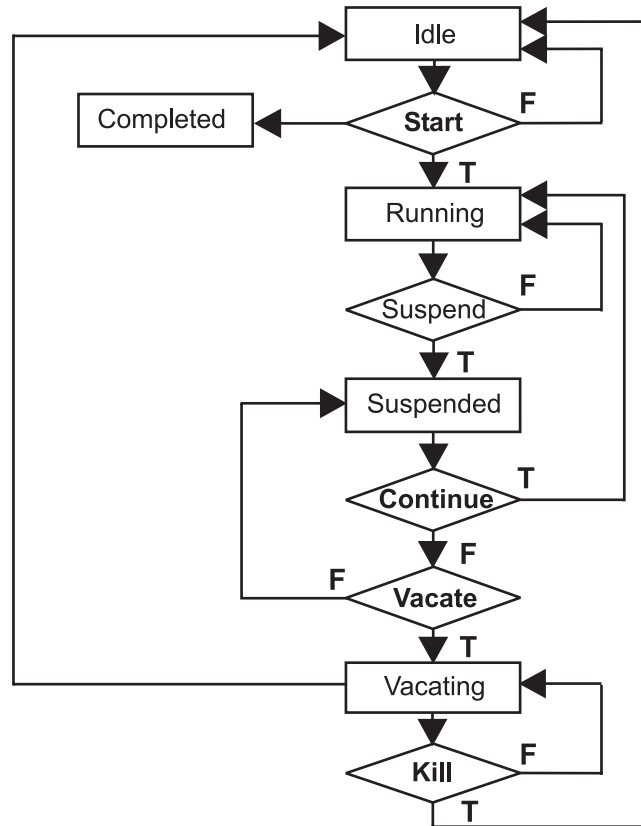


Figure 35. How control expressions affect jobs

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 may be different for each machine in the cluster. They may be modified to meet local requirements.

## Step 9: Define job accounting

LoadLeveler provides accounting information on completed LoadLeveler jobs. For detailed information on this function, refer to “Chapter 9. Gathering job accounting data” on page 75.

The following keywords allow you to control accounting functions:

**ACCT = flag**

The available flags are:

**A\_ON** Turns accounting data recording on. If specified without the **A\_DETAIL** flag, the following is recorded:

- The total amount of CPU time consumed by the entire job
- The maximum memory consumption of all tasks (or nodes).

**A\_OFF**

Turns accounting data recording off. This is the default.

**A\_VALIDATE**

Turns account validation on.

**A\_DETAIL**

Enables extended accounting. Using this flag causes LoadLeveler to record detail resource consumption by machine and by events for each

## Customizing the configuration file

job step. This flag also enables the **-x** flag of the **llq** command, permitting users to view resource consumption for active jobs.

For example:

```
ACCT = A_ON A_DETAIL
```

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\_VALIDATION = \$(BIN)/llacctval (optional)**

Keyword used to identify the executable that is called to perform account validation. You can replace the **llacctval** executable with your own validation program by specifying your program in this keyword.

### **GLOBAL\_HISTORY = \$(SPOOL) (optional)**

Keyword used to identify the directory that will contain the global history files produced by **llacctlmrg** command when no directory is specified as a command argument.

### **HISTORY\_PERMISSION = permissions | rw-rw----**

*Permissions* value of this keyword specifies the owner, group, and world permissions of the history file associated with a LoadL\_schedd daemon. It must be a string with a length of nine characters and consisting of the characters, **r**, **w**, **x**, or **-**. The default is **rw-rw----**. LoadL\_schedd will use the default setting if the specified permission are less than **rw-----**.

For example, the following section of the configuration file specifies that the accounting function is turned on. It also identifies the 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)
```

## Step 10: Specify 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.

## Customizing the configuration file

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

### **CENTRAL\_MANAGER\_HEARTBEAT\_INTERVAL = *number***

Where *number* is the amount of time in seconds that defines how frequently primary and alternate central managers communicate with each other. The default is 300 seconds or 5 minutes.

### **CENTRAL\_MANAGER\_TIMEOUT = *number***

Where *number* is the number of heartbeat intervals that an alternate central manager will wait without hearing from the primary central manager before declaring that the primary central manager is not operating. The default is 6.

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 440.

## Step 11: Specify where files and directories are located

The configuration file provided with LoadLeveler specifies default locations for all of the files and directories. You can modify their locations using the following keywords. 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.

To specify the location of the:	Specify these keywords:
Administration File	<b>ADMIN_FILE = <i>pathname</i> (required)</b> Points to the administration file containing user, class, group, machine, and adapter stanzas. For example, <code>ADMIN_FILE = \$(tilde)/admin_file</code>
Local Configuration File	<b>LOCAL_CONFIG = <i>pathname</i></b> Defines the pathname of the optional local configuration file containing information specific to a node in the LoadLeveler network. If you are using a distributed file system like NFS, some examples are: <code>LOCAL_CONFIG = \$(tilde)/\$(host).LoadL_config.local</code> <code>LOCAL_CONFIG = \$(tilde)/LoadL_config.\$(host).\$(domain)</code> <code>LOCAL_CONFIG = \$(tilde)/LoadL_config.local.\$(hostname)</code>  If you are using a local file system, an example is: <code>LOCAL_CONFIG = /var/LoadL/LoadL_config.local</code>  See “LoadLeveler variables” on page 66 for information about the <b>tilde</b> , <b>host</b> , and <b>domain</b> variables.



## Customizing the configuration file

To specify the location of the:	Specify these keywords:
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> <p><b>EXECUTE = <i>local directory/execute</i> (required)</b>            Defines the local directory to store the executables of jobs submitted by other machines.</p> <p><b>LOG = <i>local directory/log</i> (required)</b>            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.</p> <p><b>SPOOL = <i>local directory/spool</i> (required)</b>            Defines the local directory where LoadLeveler keeps the local job queue and checkpoint files, as well as:</p> <p><b>HISTORY = <i>\$(SPOOL)/history</i> (required)</b>            Defines the pathname where a file containing the history of local LoadLeveler jobs is kept.</p>
Release Directory	<p><b>RELEASEDIR = <i>release directory</i> (required)</b>            Defines the directory where all the LoadLeveler software resides. The following subdirectories are created during installation and they reside in the release directory. You can change their locations.</p> <p><b>BIN = <i>\$(RELEASEDIR)/bin</i> (required)</b>            Defines the directory where LoadLeveler binaries are kept.</p> <p><b>LIB = <i>\$(RELEASEDIR)/lib</i> (required)</b>            Defines the directory where LoadLeveler libraries are kept.</p> <p><b>NQS_DIR = <i>NQS directory</i> (optional)</b>            Defines the directory where NQS commands <b>qsub</b>, <b>qstat</b>, and <b>qdel</b> reside. The default is <b>/usr/bin</b>.</p>

## Step 12: Record and control 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 20, 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 353 describes the events that can be reported through logging controls.

“Setting up file system monitoring” on page 337 describes keywords that monitor the available free space on the file system and also enable system responses to increase system fault tolerance.

Table 20. Log control statements

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



Table 20. Log control statements (continued)

Daemon/ Process	Log File (required) (See note 1)	Max Length (required) (See note 2)	Debug Control (required) (See note 4)
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 355 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\_AFS**

Logs information related to AFS credentials.

**D\_CKPT**

Logs information related to checkpoint and restart

## Customizing the configuration file

### **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\_DCE**

Logs information related to DCE credentials.

### **D\_EXPR**

Logs steps in parsing and evaluating control expressions.

### **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.

**Note:** 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\_MACHINE**

Logs machine control functions and variables when making decisions regarding starting, suspending, resuming, and aborting remote jobs.

### **D\_NEGOTIATE**

Displays the process of looking for a job to run in the negotiator. It only pertains to this daemon.

### **D\_NQS**

Provides more information regarding the processing of NQS files.

### **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\_STANZAS**

Displays internal information about the parsing of the administration file.

### **D\_SCHEDD**

Displays how the schedd works internally.

### **D\_STARTD**

Displays how the startd works internally.

### **D\_STARTER**

Displays how the starter works internally.

### **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.

**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 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. The syntax is:

```
SAVELOGS = <directory>
```

Where *<directory>* is the directory in which log files will be archived.

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:

```
NegotiatorLogNov02.16:10:39c163n10.ppd.pok.ibm.com
NegotiatorLogNov02.16:10:42c163n10.ppd.pok.ibm.com
NegotiatorLogNov02.16:10:46c163n10.ppd.pok.ibm.com
NegotiatorLogNov02.16:10:48c163n10.ppd.pok.ibm.com
NegotiatorLogNov02.16:10:51c163n10.ppd.pok.ibm.com
NegotiatorLogNov02.16:10:53c163n10.ppd.pok.ibm.com
StarterLogNov02.16:09:19c163n10.ppd.pok.ibm.com
StarterLogNov02.16:09:51c163n10.ppd.pok.ibm.com
StarterLogNov02.16:10:30c163n10.ppd.pok.ibm.com
SchedLogNov02.16:09:05c163n10.ppd.pok.ibm.com
SchedLogNov02.16:09:26c163n10.ppd.pok.ibm.com
SchedLogNov02.16:09:47c163n10.ppd.pok.ibm.com
SchedLogNov02.16:10:12c163n10.ppd.pok.ibm.com
SchedLogNov02.16:10:37c163n10.ppd.pok.ibm.com
StartLogNov02.16:09:05c163n10.ppd.pok.ibm.com
StartLogNov02.16:09:26c163n10.ppd.pok.ibm.com
StartLogNov02.16:09:47c163n10.ppd.pok.ibm.com
StartLogNov02.16:10:12c163n10.ppd.pok.ibm.com
StartLogNov02.16:10:37c163n10.ppd.pok.ibm.com
```

**Step 13: Define network characteristics**

A **port number** is an integer that specifies the port number 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 the port number is in the configuration file and is valid, this value is used. If it is an invalid value, the default value is used.

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.

## Customizing the configuration file

The configuration file keywords associated with port numbers are the following:

### **CLIENT\_TIMEOUT = *number***

Where *number* specifies the maximum time, in seconds, that a LoadLeveler 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. The default is 30 seconds. In general, you should use this default setting unless you are experiencing delays due to an excessively loaded network. If so, you should try increasing this value. **CLIENT\_TIMEOUT** is used by all LoadLeveler daemons.

### **CM\_COLLECTOR\_PORT = *port number***

The default is 9612.

### **MASTER\_STREAM\_PORT = *port number***

The default is 9616.

### **NEGOTIATOR\_STREAM\_PORT = *port number***

The default is 9614.

### **SCHEDD\_STATUS\_PORT = *port number***

The default is 9606.

### **SCHEDD\_STREAM\_PORT = *port number***

The default is 9605.

### **STARTD\_STREAM\_PORT = *port number***

The default is 9611.

### **STARTD\_DGRAM\_PORT = *port number***

The default is 9615.

### **MASTER\_DGRAM\_PORT = *port number***

The default is 9617.

As stated earlier, 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. The first field on each line in the example that follows represents the name of a "service". In most cases, these services are also the names of daemons because few daemons need more than one udp and one tcp connection. There are two 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.

LoadL_master	9616/tcp	# Master port number for stream port
LoadL_negotiator	9614/tcp	# Negotiator port number
LoadL_negotiator_collector	9612/tcp	# Second negotiator stream port
LoadL_schedd	9605/tcp	# Schedd port number for stream port
LoadL_schedd_status	9606/tcp	# Schedd stream port for job status data
LoadL_startd	9611/tcp	# Startd port number for stream port
LoadL_master	9617/udp	# Master port number for dgram port
LoadL_startd	9615/udp	# Startd port number for dgram port

## Step 14: Enable checkpointing

This section tells you how to set up checkpointing for 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.

## Customizing the configuration file

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.

### Checkpoint keywords

The following is a summary of keywords associated with the checkpoint and restart function.

#### Administration file keyword summary:

Table 21. Administration file keyword summary

Keyword	Stanza	Default Value	Description
ckpt_dir	Class	Initial working directory	The location to be used for checkpoint files
ckpt_time_limit	Class	Unlimited	Amount of time a job step can take to checkpoint
<b>Note:</b> For more information on these keywords see “Administration file keywords” on page 111.			

#### Configuration file keyword summary:

Table 22. Configuration file keyword summary

Keyword	Default Value	Description
CKPT_CLEANUP_INTERVAL	-1	How frequently, in seconds, the <b>CKPT_CLEANUP_PROGRAM</b> should be run
CKPT_CLEANUP_PROGRAM	No default	Identify an administrator provided program to be run at the interval specified by <b>CKPT_CLEANUP_INTERVAL</b>
MAX_CKPT_INTERVAL	7200 (2 hours)	Maximum interval, in seconds, LoadLeveler will use for checkpointing running job steps
MIN_CKPT_INTERVAL	900 (15 minutes)	Initial (and minimum) interval, in seconds, LoadLeveler will use for checkpointing running job steps
<b>Note:</b> For more information on these keywords see “Configuration file keywords and LoadLeveler variables” on page 115.		

#### Job command file keyword summary:

Table 23. Job command file keyword summary

Keyword	Default Value	Description
checkpoint	No	Indicates if a job step should be enabled for checkpoint

## Customizing the configuration file

Table 23. Job command file keyword summary (continued)

Keyword	Default Value	Description
ckpt_dir	The value of the <b>ckpt_dir</b> keyword in the class stanza of the administration file	The location to be used for checkpoint files
ckpt_file	[jobname.]job_step_id.ckpt	The base name to be used for checkpoint file
ckpt_time_limit	The value of the <b>ckpt_time_limit</b> keyword in the class stanza of the administration file	Amount of time a job step can take to checkpoint
restart_from_ckpt	No	Indicates if a job step is to be restarted from an existing checkpoint file
<b>Note:</b> For more information on these keywords see “Chapter 11. Job command file keywords” on page 85.		

### 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.

See “How to checkpoint a job” on page 362 for more information.

**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

## Customizing the configuration file

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.

## Planning considerations for checkpointing jobs

**Note:** Before you consider using the Checkpoint/Restart function refer to the LoadL.README file in /usr/lpp/LoadL/READMEs for information on availability and support of this function.

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 allows 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.



## Customizing the configuration file

### 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 358.

### 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.
  - 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:
  - 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 (SP Switch or SP Switch2) 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.

- 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.



## Customizing the configuration file

- 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.

## Customizing the configuration file

### How to checkpoint a job

Checkpoints can be taken either under the control of the user application or external to the application.

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 point(s) in the application it would be appropriate 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  
For more information see “llckpt - Checkpoint a running job step” on page 142
- Checkpoint from a program which invokes the **ll\_ckpt** API to initiate checkpoint of a specific job step  
For more information see “ll\_ckpt” on page 219.
- Have LoadLeveler automatically checkpoint all running jobs which have been enabled for checkpoint
- 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*.

Table 24. Checkpoint configurations

To specify that:	Do this:
Your job is checkpointable	<ul style="list-style-type: none"><li>• Add either one of the following two options to your job command file:<ol style="list-style-type: none"><li>1. <b>checkpoint = yes</b> This enables your job to checkpoint in any of the following ways:<ul style="list-style-type: none"><li>– The application can initiate the checkpoint</li><li>– Checkpoint from a program which invokes the <b>ll_ckpt</b> API</li><li>– Checkpoint using the <b>llckpt</b> command</li><li>– As the result of a flush command</li></ul></li><li>OR</li><li>2. <b>checkpoint = interval</b> This enables your job to checkpoint in any of the following ways:<ul style="list-style-type: none"><li>– The application can initiate the checkpoint</li><li>– Checkpoint from a program which invokes the <b>ll_ckpt</b> API</li><li>– Checkpoint using the <b>llckpt</b> command</li><li>– Checkpoint automatically taken by LoadLeveler</li><li>– As the result of a flush command</li></ul></li></ol></li><li>• If you would like your job to checkpoint itself, use the API <b>ll_init_ckpt</b> in your serial application, or <b>mpc_init_ckpt</b> for parallel jobs to cause the checkpoint to occur.</li></ul>

Table 24. Checkpoint configurations (continued)

To specify that:	Do this:
LoadLeveler automatically checkpoints your job at preset intervals	<ol style="list-style-type: none"> <li>1. Add the following option to your job command file:  <b>checkpoint = interval</b>  This enables your job to checkpoint in any of the following ways: <ul style="list-style-type: none"> <li>• Checkpoint automatically at preset intervals</li> <li>• Checkpoint initiated from user application</li> <li>• Checkpoint from a program which invokes the <b>ll_ckpt</b> API</li> <li>• Checkpoint using the <b>llckpt</b> command</li> <li>• As the result of a flush command</li> </ul> </li> <li>2. 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:  <b>MIN_CKPT_INTERVAL = number</b>  Where <i>number</i> specifies the initial period, in seconds, between checkpoints taken for running jobs.  <b>MAX_CKPT_INTERVAL = number</b>  Where <i>number</i> specifies the maximum period, in seconds, between checkpoints taken for running jobs.  The time between checkpoints will be increased after each checkpoint within these limits as follows: <ul style="list-style-type: none"> <li>• The first checkpoint is taken after a period of time equal to the <b>MIN_CKPT_INTERVAL</b> has passed.</li> <li>• The second checkpoint is taken after LoadLeveler waits <i>twice as long</i> (<b>MIN_CKPT_INTERVAL</b> X 2)</li> <li>• The third checkpoint is taken after LoadLeveler waits twice as long again (<b>MIN_CKPT_INTERVAL</b> X 4) before taking the third checkpoint.</li> </ul> LoadLeveler continues to double this period until the value of <b>MAX_CKPT_INTERVAL</b> has been reached, where it stays for the remainder of the job.  A minimum value of 900 (15 minutes) and a maximum value of 7200 (2 hours) are the defaults.  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.</li> </ol>
Your job will not be checkpointed	Add the following option to your job command file: <ul style="list-style-type: none"> <li>• <b>checkpoint = no</b></li> </ul> This will disable checkpoint.

## Customizing the configuration file

Table 24. 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 would like LoadLeveler to restart your executable from an existing checkpoint file.	<ol style="list-style-type: none"><li>1. Add the following option to your job command file:<ul style="list-style-type: none"><li>• <b>restart_from_ckpt = yes</b></li></ul></li><li>2. 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"><li>• <b>ckpt_dir</b></li><li>• <b>ckpt_file</b></li></ul></li></ol> <p>When the job command file is submitted, a new job will be started which 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 <b>restart_from_ckpt = yes</b> and ensuring <b>ckpt_dir</b> and <b>ckpt_file</b> 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>

## Remove 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** = *name of program to be run*

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. The name of the program to be run should be fully qualified and must be accessible and executable by LoadLeveler.

A sample program to remove checkpoint files is provided in the `/usr/lpp/LoadL/full/samples/lckpt/rmckptfiles.c` file.

**ckpt\_cleanup\_interval** = *number*

Specifies the interval, in seconds, at which the **schedd** daemon will run the **ckpt\_cleanup\_program**. This number must be a positive integer.

**Note:** Both **ckpt\_cleanup\_program** and **ckpt\_cleanup\_interval** must contain valid values to automate this process.

## Step 15: Specify process tracking

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. Using process tracking is optional. 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**.

**Note:** **PROCESS\_TRACKING** must be set as "true" for preemption to work. For more information see "Restrictions for Gang scheduling and preemption" on page 392.

### PROCESS\_TRACKING\_EXTENSION

This keyword is used to specify 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**.

## Step 16: Configuring LoadLeveler to use DCE security services

When LoadLeveler is configured to exploit DCE security, it uses PSSP and DCE security services to:

- Authenticate the identity of users and programs interacting with LoadLeveler.
- Authorize users and programs to use LoadLeveler services. It will prevent unauthorized users and programs from misusing resources or disrupting services.
- Delegate the DCE credentials of the user submitting a job to all processes making up that job.

You can skip this section if you do not plan to use these security features or if you plan to continue to use only the limited support for DCE available in LoadLeveler 2.1. Please consult "Usage notes" on page 370 for additional information.

When LoadLeveler is configured to exploit DCE security, most of its interactions with DCE are through the PSSP security services API. For this reason, it is important that you configure PSSP security services before you configure LoadLeveler for DCE. For more information on PSSP security services, please refer to: *RS/6000 SP Planning Volume 2, Control Workstation and Software Environment* (GA22-7281-05), *Parallel System Support Programs for AIX Installation and Migration Guide Version 3 Release 2* (GA22-7347-02), and *Parallel System Support Programs for AIX Administration Guide Version 3 Release 2* (SA22-7348-02).

DCE maintains a registry of all DCE principals which have been authorized to login to the DCE cell. In order for LoadLeveler daemons to login to DCE, DCE accounts must be set up, and DCE key files must be created for these daemons. Each LoadLeveler daemon on each node is associated with a different DCE principal. The DCE principal of the Schedd daemon running on node A is distinct from the DCE principal of the Schedd daemon running on node B. Since it is possible for up to seven LoadLeveler daemons to run on any particular node (Master, Negotiator, Schedd, Startd, Kbdd, Starter, and GSmonitor), the number of DCE principal accounts and key files that must be created could reach as high as 7x(number of nodes). Since it is not always possible to know in advance on which node a particular daemon will run, a conservative approach would be to create accounts and key files for all seven daemons on all nodes in a given LoadLeveler cluster. However, it is only necessary to create accounts and key files for DCE principals which will actually be instantiated and run in the cluster.

These are the steps used for configuring LoadLeveler for DCE. We recommend that you use SMIT and the `lldcegrpmaint` command to perform this task. The manual steps are also described in "Manual configuration" on page 367, and may be useful should you need to create a highly customized LoadLeveler environment. Some of the names used in this section are the default names as defined in the file `/usr/lpp/ssp/config/spsec_defaults` and can be overridden with appropriate specifications in the file `/spdata/sys1/spsec/spsec_overrides`. Also, the term "LoadLeveler node" is used to refer to a node on an SP system that will be part of a LoadLeveler cluster.

## Customizing the configuration file

### Using SMIT and the `lldcegrpmain` command

1. Login to the SP control workstation as **root**, then login to DCE as **cell\_admin**.
2. Start the SMIT program. From SMIT's main menu, select the **RS/6000 SP System Management** option, then select the **RS/6000 SP Security** option in the next menu.
3. Perform the appropriate steps associated with this menu to configure the security features of this SP system. From LoadLeveler's perspective, the important actions are:

- **Create dcehostnames**
- **Configure SP Trusted Services to use DCE Authentication**

Before continuing to step 4, ensure that:

- DCE hostnames for LoadLeveler nodes are defined.
  - A DCE group named **spsec-services** and a DCE organization named **spsec-services** are created.
  - The DCE principals of the LoadLeveler daemons on LoadLeveler nodes are created.
  - The DCE principals of the LoadLeveler daemons on LoadLeveler nodes are added to the **spsec-services** group and the **spsec-services** organization.
  - A DCE account is created for each DCE principal associated with the LoadLeveler daemons on the SP system.
  - A DCE key file is created for each LoadLeveler daemon on the LoadLeveler nodes.
4. If the LoadLeveler cluster consists of nodes spanning several SP systems, then you should repeat step 1 through step 3 for each SP system.
  5. PSSP security services use certain fields in the SDR (System Data Repository) to determine the current software configuration. Use the command "**splstdata -p**" to verify that the field **ts\_auth\_methods** is set to either **dce** or **dce:compat**. If **ts\_auth\_methods** is set to **dce:compat** then either DCE or non-DCE authentication is allowed. For some PSSP applications, this setting also implies that if DCE authentication is activated but, DCE authentication cannot be performed, then non-DCE authentication will be used. However, LoadLeveler can not change authentication methods dynamically, and the **dce:compat** setting simply indicates that LoadLeveler can be brought up in either DCE or non-DCE authentication modes using the **DCE\_ENABLEMENT** keyword.
  6. Add these statements to the LoadLeveler global configuration file:

```
DCE_ENABLEMENT = TRUE
DCE_ADMIN_GROUP = LoadL-admin
DCE_SERVICES_GROUP = LoadL-services
```

**DCE\_ENABLEMENT** must be set to **TRUE** to activate the DCE security features of LoadLeveler. The *LoadL-admin* group should be populated with DCE principals of users who are to be given LoadLeveler administrative privileges. For more information on populating the *LoadL-admin* group, see 9 on page 367. The *LoadL-services* group should be populated with the DCE principals of all the LoadLeveler daemons that will be running in the current cluster. You can use the **lldcegrpmain** command to automate this process. For more information on populating the *LoadL-services* group, see step 8 on page 367. Note that these daemons are already members of the **spsec-services** group. If there is more than one DCE-enabled LoadLeveler cluster within the same DCE cell, then it is important that the name assigned to **DCE\_SERVICES\_GROUP** for each cluster be distinct; this will avoid any potential operational conflict.



## Customizing the configuration file

7. Add DCE hostnames to the machine stanzas of the LoadLeveler administration file. The machine stanza of each node defined in the LoadLeveler administration file must contain a statement with this format:

```
dce_host_name = DCE hostname
```

Execute either "**SDRGetObjects Node dcehostname**," or "**llexstSDR**" to obtain a listing of DCE hostnames of nodes on an SP system.

8. Execute the command:

```
lldcegrpmain config_pathname admin_pathname
```

Where *config\_pathname* is the pathname of the LoadLeveler global configuration file and *admin\_pathname* is the pathname of the LoadLeveler administration file. The **lldcegrpmain** command will:

- Create the *LoadL-services* and *LoadL-admin* DCE groups (if they do not already exist).
- Add the DCE principals of all the LoadLeveler daemons in the LoadLeveler cluster defined by the *admin\_pathname* file to the *LoadL-services* group.

For more information about the **lldcegrpmain** command, see "lldcegrpmain - LoadLeveler DCE group maintenance utility" on page 153.

9. Add the DCE principals of users who will have LoadLeveler administrative authority for the cluster to the *LoadL-admin* group. For example, this command adds **loadl** to the **LoadL-admin** group:

```
dcecp -c group add LoadL-admin -member loadl
```

## Manual configuration

Here is an example of the steps you must take to configure LoadLeveler for DCE.

In this example, the LoadLeveler cluster consists of 3 nodes of an SP system which belong to the same DCE cell. Their hostnames and DCE hostnames are the same: c163n01.pok.ibm.com, c163n02.pok.ibm.com, and c163n03.pok.ibm.com. Assume that the basic PSSP security setup steps have been performed, and that the DCE group **spsec-services** and the DCE organization **spsec-services** have been created.

1. Login to any node in the DCE cell as **root** and login to DCE as **cell\_admin**.
2. Create LoadLeveler's product directory if it does not already exist. First, see if the directory has already been created:

```
dcecp -c cdsli ././subsys
```

This command lists the contents of the *././subsys* directory in DCE. LoadLeveler's product name within DCE is **LoadL**, so its product directory is *././subsys/LoadL*. If this directory already exists, then continue to the next step. If it does not exist, issue the following command to create it:

```
dcecp -c directory create ././subsys/LoadL
```

3. Create the DCE principal names for all of the LoadLeveler daemons in the LoadLeveler cluster. PSSP security services expect the DCE principal name of a LoadLeveler daemon to have the format:

```
product_name/dce_host_name/dce_daemon_name
```

Where:

*product\_name*

Is the product name and should always be set to **LoadL**.

## Customizing the configuration file

*dce\_host\_name*

Is the DCE hostname of the node on which the daemon will run.

*dce\_daemon\_name*

Is the DCE name of the daemon and is defined in the file **/usr/lpp/ssp/config/spsec\_defaults**. Go to the LoadLeveler section of this file. You will find a **SERVICE** record similar to this for all the seven daemons:

```
SERVICE:LoadL/Master:kw:root:system
```

The relevant portion of this record is **Master**; this is the DCE daemon name of **LoadL\_master**. The DCE daemon names of other daemons can be identified in a similar manner.

For the c163n01.pok.ibm.com node, the following commands will create the desired principal names:

```
dcecp -c principal create LoadL/c163n01.pok.ibm.com/Master
dcecp -c principal create LoadL/c163n01.pok.ibm.com/Negotiator
dcecp -c principal create LoadL/c163n01.pok.ibm.com/Schedd
dcecp -c principal create LoadL/c163n01.pok.ibm.com/Kbdd
dcecp -c principal create LoadL/c163n01.pok.ibm.com/Startd
dcecp -c principal create LoadL/c163n01.pok.ibm.com/Starter
dcecp -c principal create LoadL/c163n01.pok.ibm.com/GSmonitor
```

These commands must then be repeated for each node in the LoadLeveler cluster, replacing the *dce\_host\_name* with the DCE hostname of each respective node.

4. Add the principals defined in step 3 on page 367 to the PSSP security services' services group. This group is named **spsec-services**. PSSP security services require that any daemon using their APIs be members of this group. This command will add the DCE principal of the Master daemon on node c163n01 to the spsec-services group.

```
dcecp -c group add spsec-services -member LoadL/c163n01.pok.ibm.com/Master
```

This operation must be repeated for all of the other LoadLeveler daemons on c163n01, and the complete set of operations must be repeated for all of the nodes in the LoadLeveler cluster.

5. Add the principals defined in step 3 on page 367 to the **spsec-services** organization. The following command will add the DCE principal of the Master daemon on node c163n01 to the **spsec-services** organization.

```
dcecp -c organization add spsec-services -member LoadL/c163n01.pok.ibm.com/Master
```

This operation must be repeated for all of the other LoadLeveler daemons on c163n01, and the complete set of operations must be repeated for all of the nodes in the LoadLeveler cluster.

6. Create a DCE account for each of the principals defined in step 3 on page 367. This series of commands will create a DCE account for the Master daemon on node c163n01:

```
dcecp <Enter>
dcecp> account create LoadL/c163n01.pok.ibm.com/Master \
        -group spsec-services -organization spsec-services \
        -password service-password -mypwd cell_admin's-password
dcecp> quit
```

The *service-password* passed to DCE in this command can be any valid DCE password. Please take note of it since you will need it when you create the key file for this daemon in step 8 on page 369. The continuation character "\" is not



## Customizing the configuration file

supported by **dcecp**, but appears in the example merely for clarity. This operation must be repeated for the other LoadLeveler daemons on c163n01, and the complete set of operations must be repeated for all of the nodes in the LoadLeveler cluster.

7. Create directories to contain the key files for the principals defined in step 3 on page 367.

```
mkdir -p /spdata/sys1/keyfiles/LoadL/dce_host_name
```

You must login to the appropriate node to perform this operation. This operation must be repeated for every node in the LoadLeveler cluster.

NOTE: The directory **/spdata/sys1/keyfiles** should already exist on each node in the cluster which has been installed with a level of PSSP software that supports DCE Security exploitation. If this directory does not exist, then the node cannot support DCE Security and LoadLeveler 2.2 in DCE mode will not run on it. If this configuration seems to be in error, contact your system administrator to determine which nodes in the cluster should support DCE Security.

8. Create a key file for each LoadLeveler daemon on the node on which it will run. The key file contains security-related information specific to each daemon. Use this series of commands:

```
dcecp <Enter>
dcecp> keytab create LoadL/c163n01.pok.ibm.com/Master \
        -storage /spdata/sys1/keyfiles/LoadL/c163n01.pok.ibm.com/Master \
        -data { LoadL/c163n01.pok.ibm.com/Master plain 1 service-password }
dcecp> quit
```

You must login to node c163n01 to perform this operation. DCE must be able to locate the key file locally, otherwise the daemon's login to DCE on startup will fail. The principal name passed to DCE in the preceding example is the same principal name defined in step 3 on page 367. The AIX path passed with the "-storage" flag should point to the same directory created in step 7. The principal name passed with the "-data" flag should match the principal name used at the beginning of the command. The password used in the *service-password* field must be the same as the service password defined when this principal's account was created in step 6 on page 368.

This operation must be repeated for all of the other LoadLeveler daemons on node c163n01, and the complete set of operations must be repeated for all of the nodes in the LoadLeveler cluster.

9. Perform steps 5 on page 366, 6 on page 366, and 7 on page 367 of "Using SMIT and the lldcegrpmaint command" on page 366.
  10. Create the DCE groups *LoadL-admin*, and *LoadL-services*. This command creates the DCE group **LoadL-admin**:
- ```
dcecp -c group create LoadL-admin
```
11. Add the DCE principals of users who will have LoadLeveler administrative authority for the cluster to the *LoadL-admin* group. This command adds **loadl** to the **LoadL-admin** group:
- ```
dcecp -c group add LoadL-admin -member loadl
```
12. Add the principals defined in step 3 on page 367 to the *LoadL-services* group. This command will add the DCE principal of the Master daemon on node c163n01.pok.ibm.com to **LoadL-services**:

```
dcecp -c group add LoadL-services -member LoadL/c163n01.pok.ibm.com/Master
```

## Customizing the configuration file

This operation must be repeated for all of the other LoadLeveler daemons on node c163n01, and the complete set of operations must be repeated for all of the nodes in the LoadLeveler cluster.

### Usage notes

1. If the **DCE\_ENABLEMENT** keyword is set to **TRUE**, LoadLeveler uses the PSSP security service API to perform mutual authentication of all appropriate transactions in addition to using the pair of programs specified by **DCE\_AUTHENTICATION\_PAIR** to obtain the opaque credentials object and to authenticate to DCE before starting a job. The default pair of programs used by LoadLeveler, **ldelegate** and **l impersonate** support credentials forwarding. See pages 371, and 283 for more information on the **DCE\_AUTHENTICATION\_PAIR** keyword.

If the **DCE\_ENABLEMENT** keyword is not defined or set to **FALSE**, the limited form of DCE authentication introduced in LoadLeveler 2.1 can still be activated through the use of the **DCE\_AUTHENTICATION\_PAIR** keyword in conjunction with the **lgetdce** and **lsetdce** programs or an installation defined functionally equivalent pair of programs. If this level of DCE support meets your requirements, then you can ignore the setup steps in this section.

2. When **DCE\_ENABLEMENT** is set to **TRUE**, LoadLeveler uses a different set of criteria to determine who owns job steps, and who has administrator privileges.
  - LoadLeveler considers you to be the owner of a job step if your DCE principal matches the DCE principal associated with that job step.
  - LoadLeveler administrators are usually defined to LoadLeveler through a list of names associated with the **LOADL\_ADMIN** keyword. However, when **DCE\_ENABLEMENT** is **TRUE**, this list is no longer used for this purpose. Instead, users and processes whose DCE principals are members of the *LoadL-admin* DCE group are given LoadLeveler administrative privileges.

Note: The **LOADL\_ADMIN** keyword is also used to provide LoadLeveler with a list of users who are to receive mail notification of problems encountered by the **LoadL\_master** daemon. This function is not affected by the **DCE\_ENABLEMENT** keyword.

3. If **DCE\_ENABLEMENT** is set to **TRUE**, you must login to DCE with the **dce\_login** command before attempting to execute any LoadLeveler command. Also, if an AIX user's user name is different from the user's DCE principal name, then the AIX user must have a **.k5login** file in the home directory specifying which DCE principal may execute using the AIX account. For example, if your DCE principal in the cell **local\_dce\_cell** is **user1\_dce**, and your AIX user name is **user1**, then you will have to add an entry such as "user1\_dce@local\_dce\_cell" to the **.k5login** file in your home directory.

## Step 17: Specify additional configuration file keywords

This section describes 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.

**Note:** For the keywords listed below which have a *number* as the value on the right side of the equal sign, that *number* must be a numerical value and cannot be an arithmetic expression.

### **ACTION\_ON\_MAX\_REJECT = HOLD | SYSHOLD | CANCEL**

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

## Customizing the configuration file

System Hold status; CANCEL specifies that jobs are canceled. The default is HOLD. When a job is rejected, LoadLeveler sends a mail message stating why the job was rejected.

### **ACTION\_ON\_SWITCH\_TABLE\_ERROR = *program***

Where *program* is 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. The default is to not run a program.

### **AFS\_GETNEWTOKEN = *myprog***

Where *myprog* is an administrator supplied program that, for example, can be used to refresh an AFS token. The default is to not run a program.

For more information, see “Handling an AFS token” on page 284.

### **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* is the path name of a LoadLeveler or installation supplied program that uses the handle obtained by *program1* to authenticate to DCE before starting the job on the executing machine(s).

For more information on DCE security credentials, see “Handling DCE security credentials” on page 283.

### **DRAIN\_ON\_SWITCH\_TABLE\_ERROR = true | false**

When **DRAIN\_ON\_SWITCH\_TABLE\_ERROR** is set to true, the **startd** will be drained when the switch table fails to unload. This will flag the administrator that intervention may be required to unload the switch table. The default is **false**.

### **HISTORY\_PERMISSION = *permissions* | **rw-rw----****

*Permissions* value of this keyword specifies the owner, group, and world permissions of the history file associated with a LoadL\_schedd daemon. It must be a string with a length of nine characters and consisting of the characters, **r**, **w**, **x**, or **-**. The default is **rw-rw----**. LoadL\_schedd will use the default setting if the specified permission are less than **rw-----**.

### **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*. The default is 300 seconds.

### **MAX\_JOB\_REJECT = *number***

Where *number* specifies the number of times a job can be rejected before it is removed (canceled) or put in User Hold or System Hold status. That is, a rejected job is redispached until the **MAX\_JOB\_REJECT** value is reached. The default is -1, meaning a job is redispached an unlimited number of times. A job that cannot run for various reasons (such as a **uid** mismatch, unavailable resources, or wrong permissions) on one machine will be rejected on that machine, and LoadLeveler will attempt to run the job on another machine. A value of 0 means that if the job is rejected, it is immediately removed. (For related information, see the **NEGOTIATOR\_REJECT\_DEFER** keyword in this section.)

### **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. The default is 30 seconds.

## Customizing the configuration file

### **NEGOTIATOR\_CYCLE\_DELAY = *number***

Where *number* specifies the 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, etc. Delaying the scheduling of jobs might improve the overall performance of the negotiator by preventing it from spending excessive time attempting to schedule jobs. The

**NEGOTIATOR\_CYCLE\_DELAY** must be less than the **NEGOTIATOR\_INTERVAL**. The default is 0 seconds.

### **NEGOTIATOR\_LOADAVG\_INCREMENT = *number***

Where *number* 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. The default value is .5.

### **NEGOTIATOR\_PARALLEL\_DEFER = *number***

Where *number* 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. The default, set internally by LoadLeveler, is **NEGOTIATOR\_INTERVAL** multiplied by 5.

### **NEGOTIATOR\_PARALLEL\_HOLD = *number***

Where *number* 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. The default, set internally by LoadLeveler, is **NEGOTIATOR\_INTERVAL** multiplied by 5.

### **NEGOTIATOR\_RECALCULATE\_SYSPRIO\_INTERVAL = *number***

Where *number* specifies the amount of time in seconds between calculation of the **SYSPRIO** values for waiting jobs. The default is 120 seconds. Recalculating the priority can be CPU-intensive; specifying low values for the **NEGOTIATOR\_RECALCULATE\_SYSPRIO\_INTERVAL** keyword may 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. For more information, see "Step 6: Prioritize the queue maintained by the negotiator" on page 343.

### **NEGOTIATOR\_REJECT\_DEFER = *number***

Where *number* specifies the amount of time in seconds the negotiator waits before it considers scheduling a job to a machine that recently rejected the job. The default is 120 seconds. (For related information, see the **MAX\_JOB\_REJECT** keyword in this section.)

### **NEGOTIATOR\_REMOVE\_COMPLETED = *number***

Where *number* is 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. The default is 0 seconds.

### **NEGOTIATOR\_RESCAN\_QUEUE = *number***

Where *number* 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

## Customizing the configuration file

keyword must be greater than the **NEGOTIATOR\_INTERVAL** value; if it is not, the default is used. The default is 900 seconds.

### **OBITUARY\_LOG\_LENGTH = *number***

Where *number* 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. The default is 25.

### **POLLING\_FREQUENCY = *number***

Where *number* 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 is also the minimum interval at which the kbdd daemon reports keyboard or mouse activity to the startd daemon. A value of 5 is the default.

### **POLLS\_PER\_UPDATE = *number***

Where *number* 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**. The default is 24.

### **PUBLISH\_OBITUARIES = true | false**

Where **true** specifies that the master daemon sends mail to the administrator(s), identified by **LOADL\_ADMIN** keyword, when any of the daemons it manages dies abnormally.

### **RESTARTS\_PER\_HOUR = *number***

Where *number* 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 administrator(s) identified by the **LOADL\_ADMIN** keyword and exits. The default is 12.

### **SCHEDD\_INTERVAL = *number***

Where *number* specifies the interval, in seconds, at which the schedd daemon checks the local job queue and updates the negotiator daemon. The default is 60 seconds.

### **VM\_IMAGE\_ALGORITHM = FREE\_PAGING\_SPACE | FREE\_PAGING\_SPACE\_PLUS\_FREE\_REAL\_MEMORY**

Specifies which algorithm the Central Manager uses to decide whether a machine has enough virtual memory to meet the *image\_size* requirement of a job step. The default for this keyword is **FREE\_PAGING\_SPACE**. The LoadLeveler computed values of free real memory and free paging space should track very closely with the values reported by the **svmon -G** command.

If **VM\_IMAGE\_ALGORITHM = FREE\_PAGING\_SPACE\_PLUS\_FREE\_REAL\_MEMORY** and all other requirements are met, then a machine having 1.2 GB of free real memory and 0.9 GB of free paging space (2.1 GB free) should be able to start a job step that has an image size requirement of 2 GB (*image\_size* = 2000000).

### **WALLCLOCK\_ENFORCE = true | false**

Where **true** specifies that the **wall\_clock\_limit** on the job will be enforced. The **WALLCLOCK\_ENFORCE** keyword is only valid when the External Scheduler is enabled.

### Setting up job accounting files

The following procedure 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.

#### Task 1: Update the configuration file

Edit the configuration file according to the following table:

Edit this keyword:	To:
GLOBAL_HISTORY	Specify a directory in which to place the global history files.
ACCT	Turn accounting and account validation on and off and specify detailed accounting.
ACCT_VALIDATION	Specify the account validation routine.
<b>Note:</b> See “Step 9: Define job accounting” on page 349 for more information on these keywords.	

#### Task 2: Merge multiple files collected from each machine into one file

You can accomplish this step using either the **llacctmrg** command or the graphical user interface:

- Using **llacctmrg**: See “llacctmrg - Collect machine history files” on page 138 for the syntax of this command.
- Using the graphical user Interface:

**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.

#### Task 3: Report job information on all the jobs in the history file

You can accomplish this step using either the **llsummary** command or the graphical user interface:

- Using **llsummary**: see “llsummary - Return job resource information for accounting” on page 202 for the syntax of this command.
- Using the graphical user interface:

**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.

## Task 4: Using account numbers and setting up account validation

1. Specify the following keyword in the user stanza in the administration file:

**account = *list***

Where *list* is a blank delimited list of account numbers a user may use when submitting jobs.

2. Instruct users to associate an account number with their job:

- Using the job command file: add the **account\_no** keyword to the job command file. See “Chapter 11. Job command file keywords” on page 85 for details.
- Using the graphical user interface:

**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.

3. 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.

## Task 5: Specifying machines and their weights

To specify weights to associate with machines, specify the following keyword in a machine’s machine stanza in the administration file:

**speed = *number***

Where *number* defines the weight associated with a particular machine. The higher numbers correspond with a greater weight. The default weight is 1.0.

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.

---

## Routing jobs to NQS machines

The following procedure details how to set up your system for routing jobs to machines running NQS.

## Routing jobs to NQS machines

Assume Figure 36 depicts your environment. You have three machines in the cluster named A, B, and C. Outside of the cluster, you have machine D running NQS.

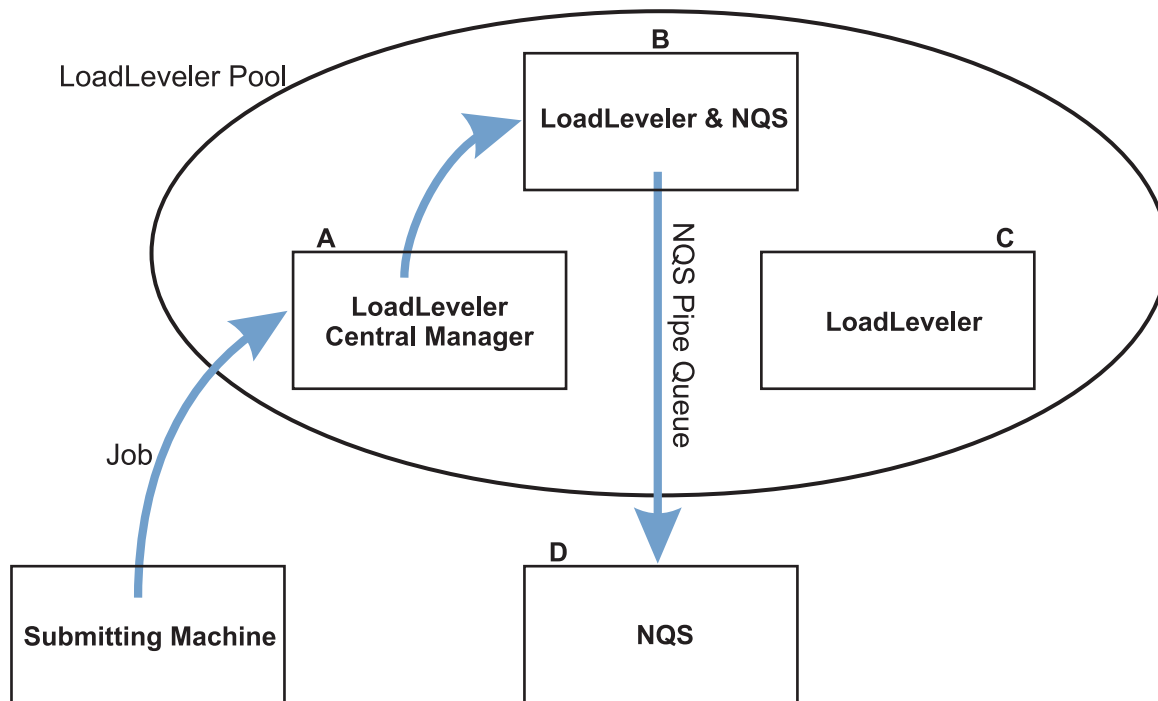


Figure 36. Environment illustrating jobs being routed to NQS machines.

### Task 1: Modify the administration file

After setting up your NQS environment, modify the **LoadL\_admin** file by defining the class **NQS** including the following stanzas:

```
NQS:
type = class
NQS_class = true
NQS_submit = pipe_a
NQS_query = queue@chevy.kgn.ibm.com
```

### Task 2: Modify the configuration file

Modify the **LoadL\_config.local** on the machine(s) that you want to accept this class of jobs. In this example, you would modify machine B's **LoadL\_config.local** file. To do this, add a class statement similar to:

```
CLASS = {"NQS" "a" "b" ....}
```

Where NQS is the name of the class of jobs that will be routed to the machines that run NQS, and a and b are names of additional classes.

### Task 3: Submit the jobs

After you perform the previous tasks, users can route their jobs to machines running NQS using the **llsubmit** command. The job command file must specify the **class** keyword. For example:

```
class = NQS
```



## Routing jobs to NQS machines

The job command file must also contain the shell script to be submitted to the NQS node. NQS accepts only shell scripts, binaries are not allowed. All options in the command file pertaining to scheduling the job will be used by LoadLeveler to schedule the job. When the job is dispatched to the node running the specified NQS class, the LoadLeveler options pertaining to the runtime environment are converted to NQS options and the job is submitted to the specified NQS queue.

LoadLeveler command file options are used as follows:

### **arguments**

Error message generated and job not submitted

### **checkpoint**

Error message generated and job not submitted

**class** Used only for LoadLeveler scheduling

### **core\_limit**

Converted to **-lc** option

### **cpu\_limit**

Converted to **-lt** option

### **data\_limit**

Converted to **-ld** option

### **environment**

If COPY\_ALL is specified, the option is converted to **-x**, otherwise error message generated and job not submitted

**error** Converted to **-e**

### **executable**

Error message generated and job not submitted

### **file\_limit**

Converted to **-lf** option

**hold** Used only for LoadLeveler scheduling

### **image\_size**

Error message generated and job not submitted

### **initialdir**

Error message generated and job not submitted

**input** Error message generated and job not submitted

### **notification**

If the option specified is

#### **always**

Converted to **-mb** and **-me** options

**error** Converted to **-me** option

**start** Converted to **-mb** option

**never** Ignored

#### **complete**

Converted to **-me** option

### **notify\_user**

Converted to **-mu** option

## Routing jobs to NQS machines

### **output**

Converted to **-o** option

### **preferences**

Used only for LoadLeveler scheduling

**queue** Places one copy of job in the LoadLeveler queue

### **requirements**

Used only for LoadLeveler scheduling

### **restart**

If the option specified is

**yes** Ignored

**no** Converted to **-nr** option

### **rss\_limit**

Converted to **-lw** option

**shell** Converted to **-s** option

### **stack\_limit**

Converted to **-ls** option

### **start\_date**

Used only for LoadLeveler scheduling

### **user\_priority**

Used only for LoadLeveler scheduling

Users can also submit an NQS script. In this case, any NQS options in the script are used to schedule the job and once dispatched by LoadLeveler, the file is sent to NQS unmodified.

LoadLeveler schedules these jobs the same as it schedules other jobs. When the job is dispatched, LoadLeveler determines whether or not it is running in an NQS class. If it is, an NQS command **qsub** is issued.

LoadLeveler monitors the job by periodically invoking a **qstat** command. A **qstat** command is first issued for the pipe queue on the local host. If the request id is not found, a **qstat** is issued for each queue listed in the NQS\_query class keyword. If the request id is still not found, starter marks the job as complete.

When a job is sent to an NQS class, **lsubmit** saves the following environment variables:

- HOME
- LOGNAME
- MAIL
- PATH
- SHELL
- TZ
- USER

When LoadLeveler dispatches the job, these environment variables are installed so that they are available to **qsub**. **lsubmit** also saves the name of the current directory (pwd) and the current value of the user file create mask (umask).

## Task 4: Obtain status of NQS jobs

Users can obtain status of NQS jobs in the same way as they obtain status of LoadLeveler jobs - either by using the **llq** command or by viewing the Jobs window on the graphical user interface. The users can identify the NQS jobs by the class field on the Jobs window.

LoadLeveler monitors the job until **qstat** shows the job is no longer in any specified queue.

NQS does not provide job accounting. Therefore, the only accounting information LoadLeveler will have is the total time for the job.

LoadLeveler will not send mail when the job completes. The LoadLeveler notification option is translated to the appropriate NQS flag (me or mb) and NQS will send the mail.

## Task 5: Cancel NQS jobs

Users can cancel NQS jobs using the LoadLeveler **llcancel** command. All they need to know is the LoadLeveler job id for the NQS job. Once they submit their request to cancel the job, LoadLeveler forwards their request to the appropriate node and a **qdel** will be issued for the job for the queue listed in the **NQS\_submit** and **NQS\_query** keywords.

## Routing jobs to NQS machines

---

## Chapter 17. Using Gang scheduling

---

### Overview

Gang scheduling is designed for the parallel environment and features:

- Support for time-sharing for parallel jobs
- Resource sharing that maximizes the resources available to individual jobs
- Support for preempting jobs

Without time-sharing, when a job starts depends on the jobs that are already running and the jobs that are ahead of it in the queue. By allocating time to jobs in small increments on an equitable basis, more jobs have the opportunity to start sooner. Otherwise a long running job could considerably delay the start of other jobs. Time-sharing has the effect of smoothing out job throughput by giving more jobs a chance to start.

Resource sharing allows two or more jobs to use the same resource and relies on controlling which jobs are running simultaneously on a node. It is possible to intentionally over commit the resources of a node if the actual number of concurrent users of those resources is limited. This allows, for example, two jobs which each require all of the adapter windows to run together on a node because at any given point in time only one of those jobs will actually be executing.

The concept of time-sharing under the Gang Scheduler is similar to context switching in the AIX kernel in that job steps take turns running. There are three important differences. First, all of the tasks that belong to the same job step must be executing at the same time on each of the nodes. This implies coordination among nodes and a common knowledge of the order in which to run job steps. Second, a synchronization mechanism among the nodes running a parallel job step is required so the suspension and resumptions of the tasks of a job step occur relatively simultaneously on each node. Because of the inter-task communication inherent in a parallel job step, any task that is not active effectively stops any other tasks attempting to communicate with it. Third, resource management must occur at each context switch. Because of these differences, the time-slice must be considerably longer than the time-slice AIX allocates to the process.

### Gang scheduling concepts

The mechanism that the Gang Scheduler uses to perform both the coordination and the synchronization is the gang scheduling matrix where columns represent the scheduling slots that run jobs and the rows represent the blocks of time available to run jobs, called time-slices. The intersection of a row and a column specifies the single job step that is to be running on that node's processor at a given time.

**Note:** Even though the Gang Scheduler can schedule to the individual processors on a node, you cannot bind a job process to a specific CPU.

Also present in the matrix is synchronization information and the size of the time-slice, set by the **GANG\_MATRIX\_TIME\_SLICE** keyword in the configuration file. As it considers nodes to run a job, the Negotiator requires that each node can run the job step at the same time. That is, all scheduling slots that the Negotiator assigns to the job step will have nothing running in the same row in the Gang scheduling matrix.

## Using Gang scheduling

As the number of time-slices for a node under Gang Scheduling increases, the percentage of total time that any one job receives decreases, so it is desirable to prevent the number of rows in the gang scheduling matrix from increasing indefinitely. The total number of jobs that can be concurrently dispatched to a node is still constrained by the **MAX\_STARTERS** keyword which, along with the number of jobs to be allowed to run in the same time-slice (presumably on separate CPUs on the node), effectively limits the number of rows in the gang scheduling matrix. There is an internal LoadLeveler limit of eight unique rows in the Gang Scheduling Matrix. The number of jobs that can run on a node (and the number of rows in the matrix) is also limited by several other factors:

- Job requirements
- Real memory available
- Memory needed by applications
- Limits set by system administration
- Attributes set by system administration

For an illustration of how keywords can affect the Gang matrix, see Figure 37 on page 383.

As the scheduler builds the matrix, each matrix element is considered a scheduling slot occupied by a job task. The entire job occupies some number of scheduling slots across a time-slice. This allows parallel tasks to context switch in and out of a set of processors simultaneously. The number of scheduling slots occupied by a job depends on two factors:

- Number of parallel task required
- Proportion of execution time relative to other jobs

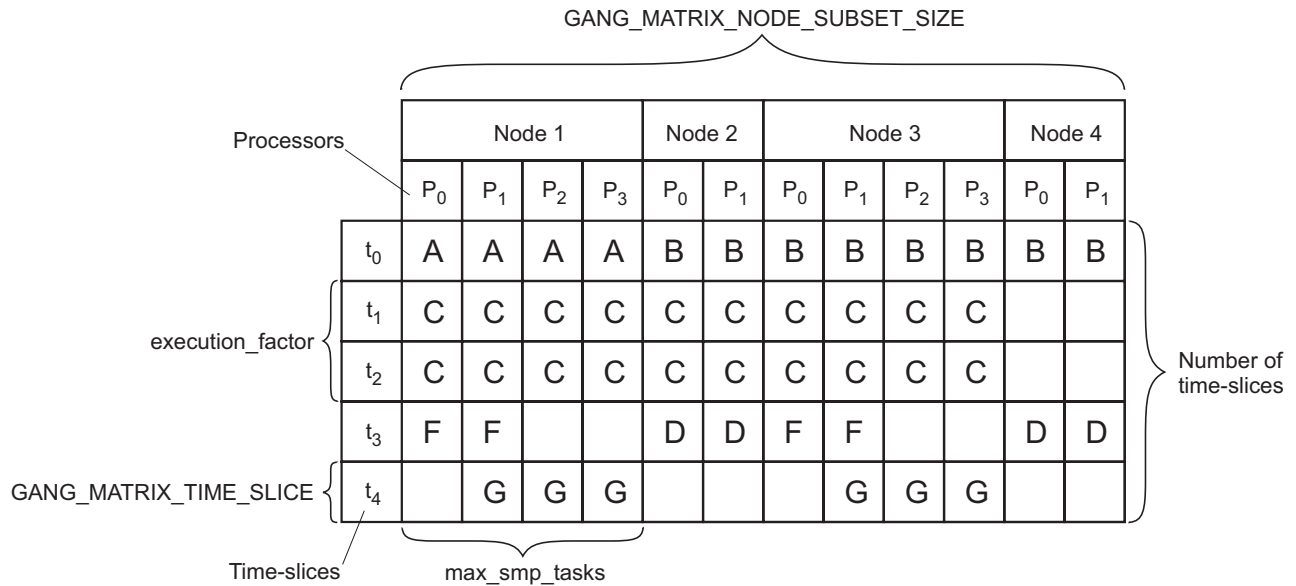


Figure 37. Effect of keywords on a Gang matrix subset.

1. This illustration represents one Gang matrix node subset
2. **GANG\_MATRIX\_NODE\_SUBSET\_SIZE**: Sets the minimum number of nodes in a Gang matrix subset
3.  $P_x$ : Represents an individual node processor
4.  $t_x$ : Represents an individual time-slice
5. **execution\_factor**: Defines the number of time-slices a job step receives
6. **GANG\_MATRIX\_TIME\_SLICE**: Defines the time-slice duration (all time-slices must be the same duration)
7. **max\_smp\_tasks**: Defines the maximum number of simultaneous tasks possible in a time-slice for a single node.
  - The default value for this keyword is the number of CPUs and the maximum value is 128
  - For nodes 1 and 3, **max\_smp\_tasks** = 4
  - For nodes 2 and 4, **max\_smp\_tasks** = 2
  - The number of tasks possible may differ from the number of tasks actually running
8. The number of time-slices (rows) a node can support is limited to the value of **MAX\_STARTERS** divided by the **max\_smp\_tasks** value for that node
  - A Gang matrix supports the largest number of time-slices required up to a maximum of eight unique time-slices
  - **MAX\_STARTERS** defines the maximum number of tasks that can run on a node
    - For nodes 1 and 3, **MAX\_STARTERS** = 20
    - For nodes 2 and 4, **MAX\_STARTERS** = 10

## Hierarchical communication

Once the gang scheduling matrix for the currently dispatched jobs has been constructed, the columns for each node in the matrix must be distributed to the Startd daemons on the nodes. In order to distribute the matrix as rapidly as possible without imposing additional burden on the Negotiator daemon, a new hierarchical communication path among Master daemons has been introduced. The Negotiator daemon sends the gang scheduling matrix to the Master daemon on the first node in the matrix. When the matrix is received, the Master daemon extracts the matrix column for the node and sends it to the local Startd daemon. It then divides the remaining columns into a number of groups specified by the **HIERARCHICAL\_FANOUT** keyword in the config file which defaults to 2. Each group is then sent to the node of its first column and the process repeats until the matrix has been delivered to every node in the hierarchy. To follow the matrix transmission in the LoadLeveler log files, **D\_HIERARCHICAL** should be set in the Master, Startd and Negotiator debug flags. The Negotiator log will indicate when a

## Using Gang scheduling

matrix is being sent and to which node. The Master log of that first node will indicate receipt of the matrix, the transmission of the matrix to the local Startd and the nodes to which the subsets of the matrix are sent. Finally, the Startd log will indicate receipt of its columns of the matrix.

There are two reasons why a matrix may be undeliverable. First, a node in the hierarchy may be unable to forward one of its subsets to a child in the hierarchy. Second, transmission of the matrix may be unacceptably slow due to network problems. Since it is crucial that all Startd daemons use the same matrix, both of these situations trigger a message to the Negotiator daemon. The Negotiator will attempt to resend the matrix two more times but if it is ultimately unsuccessful, it must cancel that matrix. In this case it will attempt to avoid the original problem either by extending the required delivery time (in the case of slow transmission) or rebuilding the matrix without the unreachable node. Failure to deliver a matrix will be recorded in the log of the Master that declared the failure and the Negotiator log when the Negotiator is notified.

## Task switching

Once a Startd daemon receives its columns of a gang scheduling matrix, it determines which time-slice should be executing at the current time by using the synchronization information in the matrix. For this reason and also for the purposes of detecting slow matrix delivery, it is important that all of the nodes where Startd is running have synchronized clocks. If the job in the time-slice that should be running (called the current time-slice), has not been started, it is started. If the job in the current time-slice was started but suspended, it is resumed. It may also be that the job in the current time-slice is already running because it is in two consecutive time-slices. In this case it continues running. Finally, if the job information for the job in the current time-slice has not been received by the Startd daemon, the time-slice is left idle. At the end of the time-slice the currently running jobs are suspended unless they also run in the next time-slice.

## Supported hardware

Gang scheduling can be used with all SP switch adapters except:

- SP Switch adapter (Micro Channel Architecture (MCA))
- RS/6000 SP System Attachment adapter

## Application support

User applications do not have to be modified to take advantage of Gang enhancements. However, user applications using the communications libraries need to be linked with the multi-threaded versions. Application environments such as POE function without modification.

## Preemption

Gang scheduling enables preemption. A running job step can be preempted to become inactive and be held in the virtual memory. A preempted job step can be resumed to continue running again. Using preemption, resources are released from preempted jobs. These resources can then be used to run other jobs which might otherwise not be able to run due to lack of resources.

There are two types of preemption:

1. System-initiated preemption
  - Automatically enforced by LoadLeveler
  - Governed by the PREEMPT\_CLASS rules defined in the global configuration file



- When resources required by an incoming job are not available, all or some job steps in certain classes may be preempted according to the `PREEMPT_CLASS` rules
- An automatically preempted job step may be resumed by LoadLeveler when resources become available and conditions such as `START_CLASS` rules are satisfied
- An automatically preempted job step can not be resumed using `llpreempt` command or `ll_preempt` subroutine
- A special kind of system-initiated preemption is related to the `llmodify` command. When `llmodify -x 99` makes a job step non-preemptable, neither user-initiated preemption nor system-initiated preemption will be able to preempt the job step. All other job steps sharing the same node will be preempted and stay in preempted state until the non-preemptable job step finishes or becomes preemptable by `llmodify -x 1,2 or 3`.

For more information see “`llmodify` - Change attributes of a submitted job step” on page 168.

2. User-initiated preemption
  - Manually initiated by LoadLeveler administrators using `llpreempt` command or `ll_preempt` subroutine
  - A manually preempted job step can be resumed using `llpreempt` command or `ll_preempt` subroutine
  - A manually preempted job step can not be resumed automatically by LoadLeveler

Once a job has been preempted, the following resources used by the job step will be released:

- Processors
- Communication switches
- Scheduling slots
- Real memory
- ConsumableCpus and ConsumableMemory

---

## Keywords specific to Gang scheduling

### Configuration file keywords for Gang scheduling

If needed, the LoadLeveler administrator can set the values associated with Gang specific configuration file keywords. If this is done, you must be aware of the restrictions imposed by the settings. For more information, see “Gang scheduling interactions and restrictions” on page 392.

**Note:** Although they are not specific to Gang scheduling, the following configuration file keywords **must** be set equal to **true** when you are using Gang scheduling:

- `MACHINE_AUTHENTICATE`
- `PROCESS_TRACKING`

### Configuration file keyword summary

Table 25. Configuration file keywords for Gang scheduling

Keyword	Default value	Range
<code>HIERARCHICAL_FANOUT</code>	2	1 or greater
<code>GANG_MATRIX_TIME_SLICE</code>	60	60 to 3600 (seconds)

## Using Gang scheduling

Table 25. Configuration file keywords for Gang scheduling (continued)

Keyword	Default value	Range
GANG_MATRIX_NODE_SUBSET_SIZE	512	1 or greater
GANG_MATRIX_REORG_CYCLE	16	1 or greater
GANG_MATRIX_BROADCAST_CYCLE	300	1 or greater (seconds)
PREEMPT_CLASS [classname]	No preemption	NA
START_CLASS [classname]	No restriction	NA

### Configuration file keyword details

#### HIERARCHICAL\_FANOUT

Specifies how many children a hierarchical message is distributed to at each level of the hierarchy. The value "1" is valid for HIERARCHICAL\_FANOUT but if specified, it means the message is delivered sequentially with each node sending the message on to the next. If the value is greater than or equal to the number of nodes in the cluster, the first node sends the message to all the other nodes. The default value for this keyword is 2.

#### GANG\_MATRIX\_TIME\_SLICE

This is the basic unit for time-sharing in Gang scheduling and it specifies the length of time (in seconds) that each matrix row (time-slice) will use to run jobs. The minimum (and default) value for this keyword is 60 (1 minute) while the maximum is 3600 (1 hour). If you specify a value lower than 60, the default value will be used. If you specify a value larger than 3600, the maximum value of one hour will be used.

#### GANG\_MATRIX\_NODE\_SUBSET\_SIZE

Specifies the ideal number of nodes for processing user jobs. When Gang scheduling initializes, it creates subsets of the nodes no smaller than the value assigned to this keyword. Gang scheduler divides the matrix into corresponding subsets for better CPU utilization and scheduling efficiency by allowing matrix subsets to have different numbers of time-slices. This reduces the number of idle time-slices. When a subset is not large enough for a user job, multiple subsets may be temporarily combined to allow the job to run. The default value for this keyword is 512 and the range is any positive integer value.

- A node can only be in one subset at a time and is separated from nodes outside the set
- The "ideal" keyword value equals the number of nodes required for the largest, frequently run job
- If you set the value too high, you may have idle cycles on some nodes during some time-slices
- If you set the value too low, the scheduling algorithm may need to merge subsets to schedule larger jobs

#### GANG\_MATRIX\_REORG\_CYCLE

Specifies the number of "negotiation loops" that the scheduler waits to reorganize the Gang matrix into subsets whose sizes are as close as possible to the ideal GANG\_MATRIX\_SUBSET\_SIZE. The default value for this keyword is 16 and the range is any positive integer value.

#### GANG\_MATRIX\_BROADCAST\_CYCLE

Specifies how often (in seconds) that the scheduler broadcasts the complete matrix to all startd daemons. This is in addition to the matrix sent

whenever resources are allocated or released for a job. The default value for this keyword is 300 seconds and the range is any positive integer value.

### **PREEMPT\_CLASS** [*incoming\_class*]

Defines the preemption rule for the job class *incoming\_class*. Uses the form:

**PREEMPT\_CLASS**[*incoming\_class*] = **ALL** { *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.

**PREEMPT\_CLASS**[*incoming\_class*] = **ENOUGH** { *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. For example:

**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.

### **Notes:**

1. Using the "ALL" value in the **PREEMPT\_CLASS** keyword places implied restrictions on when a job can start. See "Implied **START\_CLASS** values" on page 393 for more information.
2. The incoming class is designated inside [ ] brackets.
3. Outgoing classes are designated inside { } curly braces.
4. 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.
5. 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**}

6. 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**}

7. More than one **ALL** statement and more than one **ENOUGH** statement may appear at the right hand side. Multiple statements have a cumulative effect.

## Using Gang scheduling

8. Each ALL or ENOUGH statement can have multiple class names inside the curly braces. However, Gang requires a blank space delimiter between each class name.
9. ALL and ENOUGH may be in mixed cases.
10. Spaces are allowed around the brackets and curly braces.
11. PREEMPT\_CLASS [allclasses] will be ignored.

### START\_CLASS[incoming\_class]

Specifies the rule for starting a job of the *incoming\_class*. The START\_CLASS rule is applied whenever the scheduler decides whether a job step of the *incoming\_class* should start or not. Uses the form:

**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 can not 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*.

#### **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.

## Sample configuration file

The following sample illustrates the configuration file (LoadL\_config) for Gang scheduling:

```
ARCH                = R6000
LOADL_ADMIN         = loadl
MACHINE_AUTHENTICATE = True
SCHEDULER_TYPE      = GANG
```

```

PROCESS_TRACKING           = True

MAX_STARTERS               = 20
Class                      = No_Class(1) small(3) medium(4) large(4) secure(6)

PREMPT_CLASS[secure]       = ALL {allclasses}
PREMPT_CLASS[small]        = ENOUGH {medium}
PREMPT_CLASS[medium]       = ENOUGH {large}

START_CLASS[secure]        = (secure < 2)
START_CLASS[No_Class]      = (secure < 1) && (allclasses < 3)
START_CLASS[small]         = (secure < 1) && (allclasses < 3)
START_CLASS[medium]        = (secure < 1) && (allclasses < 4) && (small < 3)
START_CLASS[large]         = (secure < 1) && (allclasses < 4) && (small < 3)

GANG_MATRIX_TIME_SLICE = 120

RELEASEDIR                 = /usr/lpp/LoadL/full
ADMIN_FILE                 = $(tilde)/LoadL_admin
LOG                        = /tmp/log
SPOOL                     = /tmp/spool
EXECUTE                   = /tmp/execute
HISTORY                   = /tmp/history
BIN                       = $(RELEASEDIR)/bin
LIB                       = $(RELEASEDIR)/lib
KBDD                     = $(BIN)/LoadL_kbdd
KBDD_LOG                  = $(LOG)/KbdLog
STARTD                   = $(BIN)/LoadL_startd
STARTD_LOG                = $(LOG)/StartLog
SCHEDD                   = $(BIN)/LoadL_schedd
SCHEDD_LOG                = $(LOG)/SchedLog
NEGOTIATOR                = $(BIN)/LoadL_negotiator
NEGOTIATOR_LOG            = $(LOG)/NegotiatorLog
GSMONITOR                 = $(BIN)/LoadL_GSmonitor
GSMONITOR_LOG             = $(LOG)/GSmonitorLog
STARTER                   = $(BIN)/LoadL_starter
STARTER_LOG               = $(LOG)/StarterLog
MASTER                   = $(BIN)/LoadL_master
MASTER_LOG                = $(LOG)/MasterLog
PROCESS_TRACKING_EXTENSION = $(BIN)

START                     : T
SUSPEND                   : F
CONTINUE                  : T
VACATE                    : F
KILL                      : F

```

## Administration file keywords for Gang

If needed, the administrator can set the values associated with Gang specific administration file keywords. If this is done, you must be aware of the restrictions imposed by the settings. For more information, see “Gang scheduling interactions and restrictions” on page 392.

### Notes:

1. To use Gang 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.
2. The **css\_type** administration file keyword **must not** have the following values when you are using Gang scheduling:
  - RS/6000\_SP\_System\_Attachment\_Adapter
  - SP\_Switch\_Adapter

## Using Gang scheduling

See “Supported hardware” on page 384 for a list of switch adapters which can be used with Gang scheduling.

3. The **master\_node\_requirement** keyword in the class stanza and the **master\_node\_exclusive** keyword in the machine stanza are ignored when using Gang scheduler.

## Administration file keyword summary

Table 26. Administration file keywords for Gang scheduling

Keyword	Default value	Range
execution_factor	1	1 to 3
max_smp_tasks	The number of processors in the node	1 to 128
max_total_tasks	-1	-1, 0, or any positive integer

## Administration keyword details

### execution\_factor

This keyword appears in the Class stanza. It specifies how much processing time the job class you are defining will receive relative to other classes running on the same set of nodes.

For example, in the class stanza you set `execution_factor = 2` for job class A and `execution_factor = 1` for job class B. With that information, Gang scheduler allocates job class A two times more processing time than it does to job class B (therefore twice the number of rows in the Gang matrix ).

This keyword uses the form:

**execution\_factor = *number***

For this keyword, *number* allows the values of 1, 2, or 3. The default value is 1.

### max\_smp\_tasks

This keyword appears in the Machine stanza. It specifies the maximum number of tasks that can run at the same time on a node. This keyword uses the form:

**max\_smp\_tasks = *number***

To avoid committing a processor to more than one task at a time, set *number* so that it is less than or equal to the number of processors in the node. If you do not specify this keyword or you use a negative integer, then Gang will use the default (the number of processors in the node).

### max\_total\_tasks

This keyword appears in the User, Group and Class stanzas. Specifies the maximum number of tasks that the Gang scheduler allows a user, group, or class to run at any given time.

**max\_total\_tasks = *number***

For this keyword, the default value is -1 which allows an unlimited number of tasks. The range is: -1, 0, or any positive integer.

## Sample administration file

The following sample illustrates the administration file (LoadL\_admin) for Gang scheduling with preemption:

```

default:  type = machine
          pool_list = 1

default:  type = class          # default class stanza
          wall_clock_limit = 30:00 # default wall clock limit

default:  type = user          # default user stanza
          default_class = No_Class # default class = No_Class
          default_group = No_Group # default group = No_Group
          default_interactive_class = medium
          max_total_tasks = 50

default:  type = group          # default group stanza

secure:   type = class          # class for secure jobs
          wall_clock_limit = 120:30:00,120:00:00

small:    type = class          # class for small jobs
          wall_clock_limit = 35:00,30:00
          maxjobs = 2
          max_total_tasks = 20
          execution_factor = 2

medium:   type = class          # class for medium jobs
          wall_clock_limit = 04:30:00,04:00:00

large:    type = class          # class for large jobs
          wall_clock_limit = 120:30:00,120:00:00

c163n02.ppd.pok.ibm.com: type = machine
                        adapter_stanzas = c163sn02.ppd.pok.ibm.com c163n02.ppd.pok.ibm.com
                        alias = c163sn02.ppd.pok.ibm.com
                        max_smp_tasks = 2
                        central_manager = true

c163n03.ppd.pok.ibm.com: type = machine
                        adapter_stanzas = c163sn03.ppd.pok.ibm.com c163n03.ppd.pok.ibm.com
                        alias = c163sn03.ppd.pok.ibm.com
                        max_smp_tasks = 2

c163sn03.ppd.pok.ibm.com: type = adapter
                        adapter_name = css0
                        network_type = switch
                        interface_address = 9.114.52.131
                        interface_name = c163sn03.ppd.pok.ibm.com
                        switch_node_number = 2
                        css_type = SP_Switch_MX_Adapter

c163n03.ppd.pok.ibm.com: type = adapter
                        adapter_name = en0
                        network_type = ethernet
                        interface_address = 9.114.52.67
                        interface_name = c163n03.ppd.pok.ibm.com

c163sn02.ppd.pok.ibm.com: type = adapter
                        adapter_name = css0
                        network_type = switch
                        interface_address = 9.114.52.130
                        interface_name = c163sn02.ppd.pok.ibm.com
                        switch_node_number = 1
                        css_type = SP_Switch_MX_Adapter

c163n02.ppd.pok.ibm.com: type = adapter
                        adapter_name = en0
                        network_type = ethernet
                        interface_address = 9.114.52.66
                        interface_name = c163n02.ppd.pok.ibm.com

```



# Gang scheduling interactions and restrictions

## Network Time Protocol (NTP)

Use of NTP (such as `xntpd`) is required to run the Gang scheduler. Because Gang scheduling offers coordinated context switching for parallel jobs, it is crucial that the clocks on all nodes in the cluster be synchronized to within a few seconds. Failure to do so can cause unpredictable results and is very likely to cause jobs to fail due to the inability of one task to communicate with another, since one task could be active while another is suspended.

## Consumable resource enforcement

When a job gets suspended under Gang scheduling, the WLM class for that job gets deleted. When the job restarts (resumes), a new WLM class is created and the process is reassigned. Because of that, Gang scheduling has a limit of 27 job steps per time-slice when consumable resources are being enforced. The statistics returned by `llq -w` representing total CPU and real memory high water mark will only represent the current time slice. Also, when a suspended job is queried, the system will not report any data.

## Reconfiguration

When reconfiguring (`llctl reconfig`) LoadLeveler the following restrictions apply:

- 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
- Changes to `max_smp_tasks` will not take effect at the reconfiguration

## Circular preemption

Gang scheduling enables job preemption using rules specified with the `PREEMPT_CLASS` keyword. When you are setting up the preemption rules, make certain 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`:

```
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.

## Restrictions for Gang scheduling and preemption

The following conditions are not supported with Gang scheduling or preemption:

- `css_type = RS/6000_SP_System_Attachment_Adapter` in the administration file
- `css_type = SP_Switch_Adapter` in the administration file
- `MACHINE_AUTHENTICATE = false` in the configuration file
- `PROCESS_TRACKING = false` in the configuration file



- Circular preemption rules specified in the configuration file

If any of the conditions listed above exist, the following will occur:

- LoadLeveler will not start when SCHEDULER\_TYPE = GANG
- Reconfiguration will not take place when SCHEDULER\_TYPE = GANG

## 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.

## Last one wins rule

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:

```
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).

## Using Gang scheduling

### Job command file and Gang scheduling

The only job command file keyword that has been modified for use with Gang scheduling is the **node\_usage** keyword. For Gang scheduling **node\_usage** has a new value of **slice\_not\_shared**. **slice\_not\_shared** specifies that nodes are not shared during the time-slice the job is running. If you are using a scheduling algorithm other than Gang scheduling, **slice\_not\_shared** performs the same function as **not\_shared**. This allows use of the same job command files with different values of **SCHEDULER\_TYPE**.

Although **slice\_not\_shared** has the same function as **not\_shared** for other scheduler types, these two values have different meanings when used with Gang scheduling. With Gang scheduling, **not\_shared** means that the job will not share nodes with other jobs but it may be preempted. In contrast, **slice\_not\_shared** allows the job to time-share with other jobs on the same set of nodes in different time-slices.

The **preferences** keyword is ignored when using Gang scheduling.

---

### LoadLeveler commands for Gang

*Table 27. Commands created or modified for Gang scheduling*

Command	Detailed on page:
llclass	144
llmatrix	165
llmodify	168
llpreempt	170
llq	173
llsubmit	200

---

### APIs used with Gang scheduling

The following APIs have been created or modified for Gang scheduling:

- “Data Access API” on page 223
- “Error Handling API” on page 259
- “ll\_preempt subroutine” on page 277
- “ll\_modify subroutine” on page 275

---

## Chapter 18. Support for 64-bit applications

With 64-bit support for applications running under LoadLeveler:

- Users and Administrators can assign 64-bit integer values to selected keywords in the Job Command, Configuration, and Administration files. System resource limits, with the exception of CPU limits, are treated by LoadLeveler daemons and commands as 64-bit limits.
- The LoadLeveler commands and the GUI (xloadl) accept and display 64-bit information where appropriate.
- Both sets of 32-bit and 64-bit LoadLeveler APIs and libraries are available for application development. LoadLeveler and MPI checkpointing libraries support both 64-bit and 32-bit applications.
- Accounting statistics of completed jobs that have 64-bit integer values are preserved in the LoadLeveler history files as 64-bit data. The LoadLeveler interfaces that access the history files correctly process these 64-bit statistics.

---

### 64-bit support for Job Command, Configuration, and Administration keywords

#### 64-bit support for Job Command file keywords

**data\_limit, file\_limit, core\_limit, stack\_limit, 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. Unit specifications are accepted. In LoadLeveler 2.2, the units may be one of the following: b, w, kb, kw, mb, mw, gb, gw. In LoadLeveler 3.1, the unit specifications may also include tb (terabyte: 2\*\*40 bytes), tw (teraword: 2\*\*42 bytes), pb (petabyte: 2\*\*50 bytes), pw (petaword: 2\*\*52 bytes), eb (exabyte: 2\*\*60 bytes), ew (exaword: 2\*\*62 bytes).

#### Examples:

```
data_limit = 25.5tb,8.25gb
file_limit = 1.2eb,8pb
```

**Note:** In LoadLeveler 3.1, the hard and soft time limits associated with the keywords **cpu\_limit, job\_cpu\_limit, wall\_clock\_limit, ckpt\_time\_limit** remain 32-bit integers. If a value beyond the range of an `int32_t` data type is assigned to one of these limits, it will be truncated to either `INT32_MAX` (2147483647) or `INT32_MIN` (-2147483648).

**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)
```

**requirements, 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.

#### Examples:

## 64-bit applications

```
requirements = (Arch == "R6000") && (Disk > 5000000000) && (Memory > 60000000000)
preferences = (Disk > 60000000000) && (Memory > 90000000000)
```

**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
```

**Note:** LoadLeveler and NQS job command keywords: If your LoadLeveler cluster interacts with an NQS system, some mappings of keywords and values may occur as information is transferred between the two systems. 64-bit integer values in a LoadLeveler job command file will be mapped to 64-bit integer values in an NQS script. 64-bit integer values in an NQS script may be truncated when mapped to a LoadLeveler Job Command file. Truncation occurs when the corresponding LoadLeveler keywords support only 32-bit integers.

## 64-bit support for Administration keywords

**data\_limit, file\_limit, core\_limit, stack\_limit, rss\_limit** keywords of the Class stanza: 64-bit integer values may be assigned to these limits. Fractional specifications are allowed and will be converted to 64-bit integer values. Unit specifications are accepted and may be one of the following: b, w, kb, kw, mb, mw, gb, gw, tb, tw, pb, pw, eb, ew.

### Examples:

```
core_limit = 8gb,4.25gb
rss_limit = 1.25eb,3.33pw
```

**default\_resources** keyword of the Class stanza: Consumable resources associated with the **default\_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.

### Example:

```
default_resources = ConsumableVirtualMemory(12 gb) db2_license(112)
```

**resources** keyword of the Machine stanza: 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)
```

## 64-bit support for Configuration keywords and expressions

**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. In LoadLeveler 3.1 they are 64-bit integers and 64-bit arithmetic is used to evaluate this expression.

**Example:**

```
MACHPRIO: (Memory + FreeRealMemory) - (LoadAvg*1000 + PagesScanned)
```

---

## 64-bit support for Command line interfaces and the GUI

### 64-bit support for Command line interfaces

The LoadLeveler commands have been modified to display 64-bit information where appropriate. Shown below are fragments of the output listings of the **llclass -l**, **llstatus -l**, and **llq -l** commands on a 64-bit LoadLeveler cluster.

**llclass -l**

Sample output from **llclass -l** command is illustrated in Figure 38.

```
Name: No_Class
Priority: 30
...
Resource_requirement: ConsumableMemory(1.000 gb) ConsumableCpus(1)
...
Ckpt_limit: undefined, undefined
Wall_clock_limit: 3+08:01:01, 23:59:59 (288061 seconds, 86399 seconds)
Job_cpu_limit: 3+08:00:00, 23:59:59 (288000 seconds, 86399 seconds)
Cpu_limit: 00:30:00, 00:25:00 (1800 seconds, 1500 seconds)
Data_limit: 4.250 pb, 1.500 tb (4785074604081152 bytes, 1649267441664 bytes)
Core_limit: 2.250 tb, 1.250 tb (2473901162496 bytes, 1374389534720 bytes)
File_limit: 1.200 eb, 1.100 eb (1383505805528216384 bytes, 1268213655067531680 bytes)
Stack_limit: 40.000 mb, 30.000 mb (41943040 bytes, 31457280 bytes)
Rss_limit: 1.200 eb, 5.500 pb (1383505805528216384 bytes, 6192449487634432 bytes)
...
```

Figure 38. Sample output from **llclass -l** command

**llstatus -l**

Sample output from **llstatus -l** command is illustrated in Figure 39 on page 398.

## 64-bit applications

```
Machine           = c209f1n05.ppd.pok.ibm.com
...
SYSPRIO           = (0 - QDate)
MACHPRIO          = ((Memory + FreeRealMemory) - ((LoadAvg*1000) + PagesScanned))
VirtualMemory     = 26841210 kb
Disk              = 58243620 kb
KeyboardIdle      = 344
Tmp               = 1324854 kb
LoadAvg           = 0.290
...
Memory            = 8192 mb
FreeRealMemory    = 2390 mb
...
ConsumableResources = ConsumableCpus(3,4) ConsumableMemory(7.000 gb,8.000 gb)
...
```

Figure 39. Sample output from `llstatus -l` command

## llq -l

Sample output from `llstatus -l` command is illustrated in Figure 40.

```
Job Step Id: c209f1n05.ppd.pok.ibm.com.2.0
...
Owner: loadl
Queue Date: Mon Jul  9 21:14:17 EDT 2001
Status: Running
...
Resources: ConsumableMemory(1.000 gb) ConsumableCpus(1)
Requirements: (Arch == "R6000") && (OpSys == "AIX51")
...
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: 4.250 pb (4785074604081152 bytes)
Data Soft Limit: 1.500 tb (1649267441664 bytes)
Core Hard Limit: 2.250 tb (2473901162496 bytes)
Core Soft Limit: 1.250 tb (1374389534720 bytes)
File Hard Limit: 1.200 eb (1383505805528216384 bytes)
File Soft Limit: 1.100 eb (1268213655067531680 bytes)
Stack Hard Limit: 40.000 mb (41943040 bytes)
Stack Soft Limit: 30.000 mb (31457280 bytes)
Rss Hard Limit: 1.200 eb (1383505805528216384 bytes)
Rss Soft Limit: 5.500 pb (6192449487634432 bytes)
Step Cpu Hard Limit: 3+08:00:00 (288000 seconds)
Step Cpu Soft Limit: 23:59:59 (86399 seconds)
Wall Clk Hard Limit: 00:11:40 (700 seconds)
Wall Clk Soft Limit: 00:11:40 (700 seconds)
...
```

Figure 40. Sample output from `llq -l` command

## 64-bit support for the GUI

The LoadLeveler Graphical User Interface (`xloadl` or `xloadl_so`) accepts and displays 64-bit information where appropriate.

## 64-bit support for the LoadLeveler APIs

In LoadLeveler 3.1, 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 can not 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 the following issues:

- If DCE is not enabled, 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.
- If DCE is enabled (DCE\_ENABLEMENT = TRUE), only the 32-bit interfaces of the LoadLeveler API library are available. Subroutine calls using the 64-bit interfaces of the libllapi.a library that require DCE authentication will fail with appropriate error codes and messages.
- In LoadLeveler 3.1 the data type of a number of members of LoadLeveler objects are changed to `int64_t` from `int32_t`. For example, the data item associated with the name `LL_StepDataLimitHard` is now a 64-bit integer. In order to maintain compatibility between LoadLeveler 3.1 and older versions, an `int32_t` value is still returned when the `ll_get_data()` subroutine is called using the `LL_StepDataLimitHard` specification. However, the value of the returned data may be truncated. To obtain the `int64_t` value, `ll_get_data()` should be called using the new specification `LL_StepDataLimitHard64`. In the LoadLeveler header file `llapi.h`, the specifications ending with the "64" character string correspond to 64-bit integer data items.

## 64-bit support for Accounting functions

If the flags associated with the configuration keyword `ACCT` are turned on (`A_ON`, `A_DETAIL`), accounting statistics of jobs managed by a `LoadL_schedd` daemon are saved in a "history" file. This file resides in the "spool" directory of the `LoadL_schedd` daemon.

In LoadLeveler 3.1, a number of changes have been made to the format of the history file and the methods used to access this file. These changes include:

- 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`.
- In LoadLeveler 2.2, data contained in a history file is accessible to users through the `llsummary` command and the `GetHistory()` interface of the LoadLeveler API library. In LoadLeveler 3.1 the `GetHistory()` interface has been enhanced. The `LL_job_step` structure defined in `llapi.h` has been modified so that users of the `GetHistory()` interface can gain 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 has been modified to display 64-bit information where appropriate.
- The `LLAPI_Specification` enum defined in `llapi.h` has been significantly expanded and the `ll_get_data()` interface modified so that 32-bit and 64-bit accounting and usage information of a history file can also be accessed through this interface. Please refer to the code fragment on page 256 for an example of how to use the `ll_get_data()` subroutine to access information stored in a LoadLeveler history file.





---

## Part 6. Appendixes



---

## Appendix contents

Appendix A. Examples	405
User tasks: building job command files	405
User tasks: building parallel job command files	411
Appendix B. Customer case studies	417
Appendix C. Troubleshooting	435



---

## Appendix A. Examples

---

### User tasks: building job command files

#### Using commands

The section presents a series of simple tasks which a user might perform using commands. This section is meant for new users of LoadLeveler. More experienced users may want to continue on to “Additional examples of building job command files” on page 407.

##### Step 1: Build a job

Since you are not using the GUI, you have to 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 41:

```
# 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 41. Building a job command file*

##### Step 2: Edit a job

You can optionally edit the job command file you created in step 1.

##### Step 3: Submit a job

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).

For more information on **llsubmit**, see “llsubmit - Submit a job” on page 200

##### Step 4: Display the status of a job

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. For more information, see “llq - Query job status” on page 173.

## Using commands

### Step 5: Change the priorities of jobs in the queue

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 "How does a job's priority affect dispatching order?" on page 45.

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
```

For more information, see "llprio - Change the user priority of submitted job steps" on page 171.

### Step 6: Hold a job

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
```

For more information, see "llhold - Hold or release a submitted job" on page 161.

### Step 7: Release a hold on a job

To release the hold you placed in step 6, use the **llhold** command:

```
llhold -r wizard.22.0
```

For more information, see "llhold - Hold or release a submitted job" on page 161.

### Step 8: Display the status of a machine

To display the status of the machine to which you submitted a job, use the **llstatus** command:

```
llstatus -l wizard
```

For more information, see "llstatus - Query machine status" on page 191.

### Step 9: Cancel a job

To cancel *wizard.22.0*, use the **llcancel** command:

```
llcancel wizard.22.0
```

For more information, see "llcancel - Cancel a submitted job" on page 140.

### Step 10: Find the location of the central manager

Enter the **llstatus** command with the appropriate options to display the machine on which the central manager is running. For more information, see "llstatus - Query machine status" on page 191.

### Step 11: Find the location of the public scheduling machines

Public scheduling machines are those machines that participate in the scheduling of LoadLeveler jobs. The **llstatus** command can also be used to display the public scheduling machines.

## Additional examples of building job command files

“Serial job command file” on page 40 gives you an example of a simple job command file. This section contains examples of building and submitting more complex 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 43 on page 409. 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 42 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 42 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
# @ queue
# @ queue
# @ queue
# @ queue
```

Figure 42. Job command file with varying input statements

To submit the job, type the command:

```
llsubmit longjob.cmd
```

LoadLeveler responds by issuing the following:

```
submit: The job "116.23" with 5 job steps has been submitted.
```

The following table shows you the standard input files, standard output files, and standard error 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

## Additional examples

Job Step	Standard Input	Standard Output	Standard Error
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 43 on page 409 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/lltest1.122.0.out
#   error : /u/rhclark/OSL/oslsslv_err/lltest1_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 == "AIX43")
# @ 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/lltest1.122.1.out
#   error : /u/rhclark/OSL/oslsslv_err/lltest1_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)_$(process)_err
# @ queue
#
# Job step 2 =====
# The names of the output and error files created by this job step are:
#
#   output: /u/rhclark/OSL/oslsslv_out/lltest1.122.2.out
#   error : /u/rhclark/OSL/oslsslv_err/lltest1_122_2_err
#
# @ step_name = OSL
# @ dependency = (step_0 == 0) && (step_1 == 0)
# @ comment = oslsslv
# @ initialdir = /u/rhclark/$(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 43. 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 44 on page 411 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.)

## Additional examples

- 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 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/loadl/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 == "AIX43") && (Memory > 64))
# @ queue
# @ queue
# @ preferences = ((Memory > 128) && (Feature == "ESSL"))
# @ queue
# @ class = large
# @ arguments = spice3f5_input_benchmark1 spice3f5_output_benchmark1
# @ requirements = (Machine == "l15.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 44. Job command file used as the executable

---

## User tasks: building parallel job command files

This section contains sample job command files for the following parallel environments:

- IBM AIX Parallel Operating Environment (POE)
- Parallel Virtual Machine (PVM) 3.3 (RS6K architecture)
- Parallel Virtual Machine (PVM) 3.3.11+ (SP2MPI architecture)

### POE

Figure 45 on page 412 is a sample job command file for POE.

## Additional examples

```
#
# @ job_type = parallel
# @ environment = COPY_ALL
# @ output = poe.out
# @ error = poe.error
# @ node = 8,10
# @ tasks_per_node = 2
# @ network.LAPI = csss,shared,US
# @ network.MPI = csss,shared,US
# @ wall_clock_limit = 60
# @ executable = /usr/bin/poe
# @ arguments = /u/richc/My_POE_program -euilib "us"
# @ class = POE
# @ queue
```

Figure 45. POE job command file – multiple tasks per node

Figure 45 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 can run using the LAPI protocol in US mode with an SP switch adapter (**network.LAPI=csss,shared,US**), and/or using the MPI protocol in US mode with an HPS adapter (**network.MPI=csss,shared,US**). Note that **csss** is an installation-defined network type which is used for css0 adapters in these examples.
- The maximum run time allowed for the job is 60 seconds (**wall\_clock\_limit=60**).

Figure 46 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 = csss,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 46. POE sample job command file – invoking POE twice

Figure 46 shows the following:

- POE is invoked twice, via **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 an SP switch adapter in IP mode (**network.MPI=csss,shared,IP**).
- The maximum run time allowed for the job is 60 seconds (**wall\_clock\_limit=60**).

## PVM 3.3 (non-SP)

Figure 47 shows a sample job command file for PVM 3.3 (RS6K architecture). Before using PVM, users should contact their administrator to determine which PVM architecture has been installed.

```
# @ executable      = my_PVM_program
# @ job_type        = pvm3
# @ parallel_path    = /home/LL_userid/cmds/pvm3/$PVM_ARCH:$PVM_ROOT/lib/$PVM_ARCH
# @ class           = PVM3
# @ requirements     = (Pool == 4)
# @ output           = my_PVM_program.$(cluster).$(process).out
# @ error           = my_PVM_program.$(cluster).$(process).err
# @ min_processors   = 8
# @ max_processors   = 10
# @ queue
```

*Figure 47. Sample PVM 3.3 job command file*

Note the following requirements for PVM 3.3 (RS6K architecture) jobs:

- The job must have **job\_type = pvm3**.
- You must specify the parallel executable as the executable.

## PVM 3.3.11+ (SP2MPI architecture)

Figure 48 on page 414 shows a sample job command file for PVM 3.3.11+ (SP2MPI architecture). Before using PVM, users should contact their administrator to determine which PVM architecture has been installed. The SP2MPI architecture version should be used when users require that their jobs run in user space.

## Additional examples

```
#!/bin/ksh
# @ job_type      = parallel
# @ class         = PVM3
# @ requirements  = (Adapter == "hps_user")
# @ output = my_PVM_program.${cluster}.${process}.out
# @ error  = my_PVM_program.${cluster}.${process}.err
# @ node = 3,3
# @ queue

# Set PVM daemon and starter path dictated by LoadLeveler administrator
starter_path=/home/userid/loadl/pvm3/bin/SP2MPI
daemon_path=/home/userid/loadl/pvm3/lib/SP2MPI

# Export "MP_EUILIB" before starting PVM3 (default is "ip")
export MP_EUILIB=us
echo MP_EUILIB=$MP_EUILIB

# Clean up old PVM log and daemon files belonging to user
filelog=/tmp/pvml.id | awk -F=' ' '{print $2}' | awk -F(' ' '{print $1}'
filedaemon=/tmp/pvmd.id | awk -F=' ' '{print $2}' | awk -F(' ' '{print $1}'
rm -f $filelog > /dev/null
rm -f $filedaemon > /dev/null

# Start PVM daemon in background
$daemon_path/pvmd3 &
echo "pvm background pid=$!"
echo "Sleep 2 seconds"
sleep 2
echo "PVM daemon started"

# Start parallel executable
llnode_cnt='echo "$LOADL_PROCESSOR_LIST" | awk '{print NF}''
actual_cnt=expr "$llnode_cnt" - 1
$starter_path/starter -n $actual_cnt /home/userid/my_PVM_program
echo "Parallel executable starting"

# Check processes running and halt PVM daemon
echo "ps -a" | /home/userid/loadl/pvm3/lib/SP2MPI/pvm
echo "Halt PVM daemon"
echo "halt" | /home/userid/loadl/pvm3/lib/SP2MPI/pvm
wait
echo "PVM daemon completed"
```

Figure 48. Sample PVM 3.3.11+ (SP2MPI Architecture) job command file

Note the following requirements for PVM 3.3.11+ (SP2MPI architecture) jobs:

- The job must have **job\_type = parallel**.
- You must specify one more processor than you actually need to run the parallel job. PVM spawns an additional task to relay messages to and from the PVM daemon. Parallel tasks cannot communicate with PVM daemon directly. The additional task will be spawned on the last processor in the `LOADL_PROCESSOR_LIST`. For more information on this environment variable set by LoadLeveler see "Obtaining allocated host names" on page 55.
- You must use the PVM daemon and starter path dictated by the LoadLeveler administrator. The **parallel\_path** keyword is ignored.
- You must export `MP_EUILIB` as **us** when running in user space over the switch. `MP_PROCS`, `MP_RMPOOL` and `MP_HOSTFILE` are ignored when running under LoadLeveler.
- You should clean up any temporary PVM log or daemon files before starting the PVM daemon.

- You must start the PVM daemon in the job script, and you must start it in the background (**`$daemon_path/pvmd3 &`**).
- You must compile your parallel program following the PVM guidelines for PVM 3.3.11+ (SP2MPI architecture).
- You must start the parallel executable through the PVM starter program. The PVM starter program has no relationship to the LoadLeveler starter daemon.
- You must specify the parallel executable as an argument to the PVM starter program.
- You must specify the actual number of parallel tasks to the PVM starter program. This number must be one less than the number of processors allocated through LoadLeveler.
- You must halt the PVM daemon when the PVM starter program completes.
- You can invoke the PVM starter program only once.

### Sequence of events in a PVM 3.3.11+ job

This example demonstrates the sequence of events that occur when you submit the sample job command file shown in Figure 48 on page 414.

Figure 49 on page 416 illustrates the following:

- From the job command file, **(1)** the PVM daemon, `pvmd3`, and **(2)** the PVM starter are started under the LoadLeveler starter. The PVM starter tells the PVM daemon to start two tasks (**`my_PVM_program`**).
- **(3)** The PVM daemon starts the POE Partition Manager, which in turn **(4)** starts the POE daemons, (represented as `pvmd2`) on all three nodes.
- **(5)** The POE daemons (`pvmd2`) start the parallel tasks, **`my_PVM_program`**, on all nodes under the LoadLeveler starter. The last parallel task, **`my_PVM_program`** on Node 3, is the additional task which relays messages between the PVM daemon and the parallel tasks.

## Additional examples

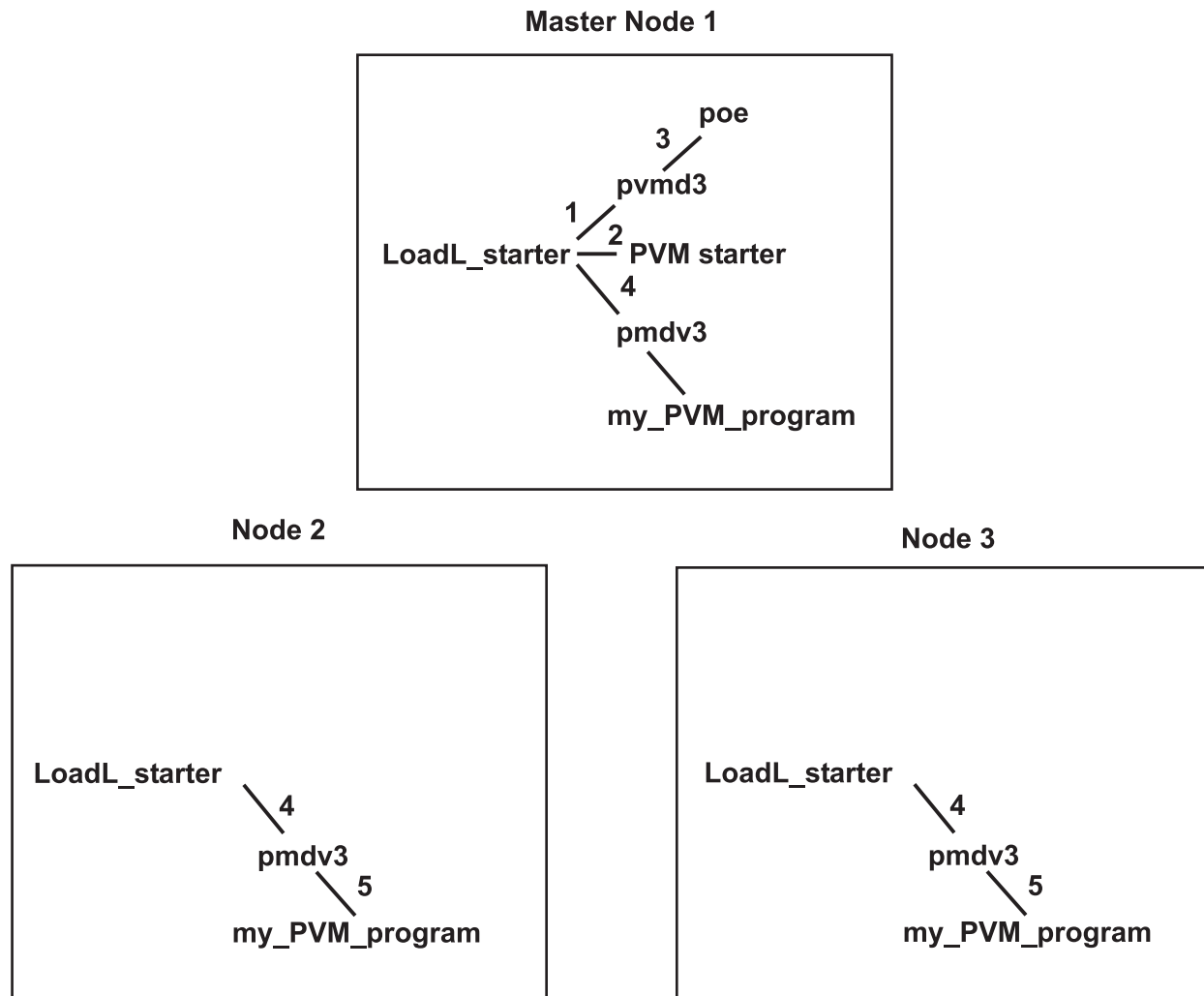


Figure 49. Sequence of events in a PVM 3.3.11+ job



---

## Appendix B. Customer case studies

This chapter gives you an overview, including configuration information, of some LoadLeveler customers. These profiles are meant to highlight how customers in different industries use LoadLeveler.

Note that all of these configurations apply to Version 1 Release 3 of the default LoadLeveler scheduler unless otherwise noted.

---

### Customer 1: technical computing at the Cornell Theory Center

The Cornell Theory Center (CTC) of Cornell University provides a high-performance computing environment to advance and facilitate research and education.

#### System configuration

The CTC runs a 160-node SP with 16 wide nodes and 144 thin nodes. The SP nodes include two interactive nodes and two submit-only nodes. The majority of the other SP nodes run batch jobs. The LoadLeveler central manager runs on a workstation outside of the SP. Also, two other non-SP workstations act as schedd hosts.

#### LoadLeveler configuration

The CTC runs parallel jobs by disabling the default LoadLeveler scheduler (**SCHEDULER\_API=YES**) and running an external scheduler. The CTC has developed this scheduler to meet the needs of its users.

The following figures represent sections of the CTC's **LoadL\_admin** file. Note that not all nodes are shown here.

```
#####
# DEFAULTS FOR MACHINE, CLASS, USER, AND GROUP STANZAS:
# Remove initial # (comment), and edit to suit.
#####
default:      type = machine
               central_manager = false # default not central manager
               schedd_host = false    # default not a public scheduler
               submit_only = false     # default not a submit-only machine
               pvm_root = /usr/local/app/pvm3 # default pvm3 directory
               rm_host = true          # default is parallel SP2 node
#            speed = 1                # default machine speed
#            cpu_speed_scale = false  # scale cpu limits by speed

default:      type = class              # default class stanza
#            priority = 0               # default ClassSysprio
#            max_processors = -1        # default max processors for class (no

default:      type = user               # default user stanza
#            priority = 0               # default UserSysprio
#            default_class = DSI        # default class
#            default_group = No_Group   # default group = No_Group (not
#                                       # optional)
#            maxjobs = -1               # default maximum jobs user is allowed
#                                       # to run simultaneously (no limit)
#            maxqueued = -1            # default maximum jobs user is allowed
#                                       # on system queue (no limit). does not
#                                       # limit jobs submitted.

default:      type = group              # default group stanza
#            priority = 0               # default GroupSysprio
#            maxjobs = -1               # default maximum jobs group is allowed
```

## Case studies

```
#                                     # to run simultaneously (no limit)
#                                     # default maximum jobs group is allowed
#                                     # on system queue (no limit). does not
#                                     # limit jobs submitted.
#####
# MACHINE STANZAS:
# These are the machine stanzas; the first machine is defined as
# the central manager. mach1:, mach2:, etc. are machine name labels -
# revise these placeholder labels with the names of the machines in the
# pool, and specify any schedd_host and submit_only keywords and values
# (true or false), if required.
#####

# spscheduler is a 43P running EASY-LL and the Central Manager
spscheduler.tc.cornell.edu:  type = machine
                             central_manager = true
                             rm_host = false

# ctc1 and ctc2 are two 43P's running as dedicated SchedDs
ctc1.tc.cornell.edu: type = machine
                    schedd_host = true

ctc2.tc.cornell.edu: type = machine
                    schedd_host = true

# Submit only node for Sweb server
arms.tc.cornell.edu:  type = machine
                    submit_only = true

#
#   Nodes of the SP2
#
# Rack 1
#
# PIOFS name server, HiPPi router, Switch & JMD primary
#r01n01.tc.cornell.edu:  type = machine
#                       alias = r01n01-css
# r01n02 & r01n05 are interactive nodes
r01n03.tc.cornell.edu:  type = machine
                       alias = r01n03-css
                       submit_only = true
r01n05.tc.cornell.edu:  type = machine
                       alias = r01n05-css
                       submit_only = true
r01n07.tc.cornell.edu:  type = machine
                       alias = r01n07-css
r01n09.tc.cornell.edu:  type = machine
                       alias = r01n09-css
r01n11.tc.cornell.edu:  type = machine
                       alias = r01n11-css
r01n13.tc.cornell.edu:  type = machine
                       alias = r01n13-css
r01n15.tc.cornell.edu:  type = machine
                       alias = r01n15-css
#
# Rack 2
#
# HPSS/PIOFS backup
#r02n01.tc.cornell.edu:  type = machine
#                       alias = r02n01-css
# r02n03, r02n05, r02n07, r02n09 are splong nodes
r02n03.tc.cornell.edu:  type = machine
                       alias = r02n03-css
                       submit_only = true
r02n05.tc.cornell.edu:  type = machine
                       alias = r02n05-css
                       submit_only = true
r02n07.tc.cornell.edu:  type = machine
```

```

alias = r02n07-css
submit_only = true
r02n09.tc.cornell.edu: type = machine
alias = r02n09-css
submit_only = true

# VIS node
#r02n11.tc.cornell.edu: type = machine
# alias = r02n11-css
r02n13.tc.cornell.edu: type = machine
alias = r02n13-css
r02n15.tc.cornell.edu: type = machine
alias = r02n15-css

#
# Rack 3
#
r03n01.tc.cornell.edu: type = machine
alias = r03n01-css
r03n02.tc.cornell.edu: type = machine
alias = r03n02-css
r03n03.tc.cornell.edu: type = machine
alias = r03n03-css
r03n04.tc.cornell.edu: type = machine
alias = r03n04-css
r03n05.tc.cornell.edu: type = machine
alias = r03n05-css
r03n06.tc.cornell.edu: type = machine
alias = r03n06-css
r03n07.tc.cornell.edu: type = machine
alias = r03n07-css
r03n08.tc.cornell.edu: type = machine
alias = r03n08-css
r03n09.tc.cornell.edu: type = machine
alias = r03n09-css
r03n10.tc.cornell.edu: type = machine
alias = r03n10-css
r03n11.tc.cornell.edu: type = machine
alias = r03n11-css
r03n12.tc.cornell.edu: type = machine
alias = r03n12-css
r03n13.tc.cornell.edu: type = machine
alias = r03n13-css
r03n14.tc.cornell.edu: type = machine
alias = r03n14-css
r03n15.tc.cornell.edu: type = machine
alias = r03n15-css

# ATM/FDDI routing node
#r03n16.tc.cornell.edu: type = machine
# alias = r03n16-css

#
# Rack 4
#
r04n01.tc.cornell.edu: type = machine
alias = r04n01-css
r04n02.tc.cornell.edu: type = machine
alias = r04n02-css
r04n03.tc.cornell.edu: type = machine
alias = r04n03-css
r04n04.tc.cornell.edu: type = machine
alias = r04n04-css
r04n05.tc.cornell.edu: type = machine
alias = r04n05-css
r04n06.tc.cornell.edu: type = machine
alias = r04n06-css
r04n07.tc.cornell.edu: type = machine

```

## Case studies

```
alias = r04n07-css
r04n08.tc.cornell.edu: type = machine
alias = r04n08-css
r04n09.tc.cornell.edu: type = machine
alias = r04n09-css
r04n10.tc.cornell.edu: type = machine
alias = r04n10-css
r04n11.tc.cornell.edu: type = machine
alias = r04n11-css
# r04n12 - r14n16 HPSS nodes
#r04n12.tc.cornell.edu: type = machine
# alias = r04n12-css
#r04n13.tc.cornell.edu: type = machine
# alias = r04n13-css
#r04n14.tc.cornell.edu: type = machine
# alias = r04n14-css
#r04n15.tc.cornell.edu: type = machine
# alias = r04n15-css
#r04n16.tc.cornell.edu: type = machine
# alias = r04n16-css
#
#####
# CLASS STANZAS: (optional)
# These are sample class stanzas; small, medium, large, and nqs are sample
# labels for job classes - revise these labels and specify attributes
# to each class.
#####
DSI:      type = class

piofs:    type = class
#####
```

The following represents the CTC's **LoadL\_config** file:

```
#
# Machine Description
#
ARCH = R6000

#
# Specify LoadLeveler Administrators here:
#
LOADL_ADMIN = loadl admin1 admin2 admin3 admin4

#
# Default to starting LoadLeveler daemons when requested
#
START_DAEMONS = TRUE

#
# Machine authentication
#
# If TRUE, only connections from machines in the ADMIN_LIST are accepted.
# If FALSE, connections from any machine are accepted. Default if not
# specified is FALSE.
#
MACHINE_AUTHENTICATE = FALSE

#
# Specify which daemons run on each node
#
SCHEDD_RUNS_HERE = False
STARTD_RUNS_HERE = True
```

```

#
# Specify information for backup central manager
#
# CENTRAL_MANAGER_HEARTBEAT_INTERVAL = 300
# CENTRAL_MANAGER_TIMEOUT = 6
#
# Specify pathnames
#
RELEASEDIR = /usr/lpp/LoadL/nfs
LOCAL_CONFIG = $(tilde)/local/configs/LoadL_config. $(host)
ADMIN_FILE = $(tilde)/LoadL_admin
LOG = /var/loadl/log
SPOOL = /var/loadl/spool
EXECUTE = /var/loadl/execute
HISTORY = $(SPOOL)/history
BIN = $(RELEASEDIR)/bin
LIB = $(RELEASEDIR)/lib
ETC = $(RELEASEDIR)/etc
#
# Specify port numbers
#
COLLECTOR_STREAM_PORT = 9612
MASTER_STREAM_PORT = 9616
NEGOTIATOR_STREAM_PORT = 9614
SCHEDD_STREAM_PORT = 9605
STARTD_STREAM_PORT = 9611
COLLECTOR_DGRAM_PORT = 9613
STARTD_DGRAM_PORT = 9615
MASTER_DGRAM_PORT = 9617
SCHEDULER_API = YES
SCHEDULER_PORT = 9624

#
# Specify accounting controls
#
ACCT = A_ON
ACCT_VALIDATION = $(BIN)/llacctval
GLOBAL_HISTORY = $(SPOOL)

#
# Specify prolog and epilog path names
#
JOB_PROLOG = $(ETC)/llprolog
JOB_EPILOG = $(ETC)/llepilog
JOB_USER_PROLOG = $(ETC)/ll_user_prolog
JOB_USER_EPILOG = $(ETC)/ll_user_epilog
#
#
# Refresh AFS token program.
#
AFS_GETNEWTOKEN = $(ETC)/tokenreviveclient
#
# Customized mail delivery program.
#
# MAIL =

#
# Customized submit (job command file) filter program.
#
# SUBMIT_FILTER =

#
# Specify checkpointing intervals
#
MIN_CKPT_INTERVAL = 900
MAX_CKPT_INTERVAL = 7200

```

## Case studies

```
# LoadL_KeyboardD Macros
#
KBDD          = $(BIN)/LoadL_kbdd
KBDD_LOG      = $(LOG)/KbdLog
MAX_KBDD_LOG  = 64000
KBDD_DEBUG    =

#
# Specify whether to start the keyboard daemon
#

X_RUNS_HERE   = False

#
# Specify whether to use X server XGetIdleTime() protocol extension
#

USE_X_IDLE_EXTENSION = False

#
# LoadL_StartD Macros
#
STARTD        = $(BIN)/LoadL_startd
STARTD_LOG    = $(LOG)/StartLog
MAX_STARTD_LOG = 5000000
#STARTD_DEBUG = D_STARTD D_FULLDEBUG D_THREAD
STARTD_DEBUG  = D_FULLDEBUG
POLLING_FREQUENCY = 10
POLLS_PER_UPDATE = 24
JOB_LIMIT_POLICY = 240
JOB_ACCT_Q_POLICY = 3600

#
# LoadL_SchedD Macros
#
SCHEDD        = $(BIN)/LoadL_schedd
SCHEDD_LOG    = $(LOG)/SchedLog
MAX_SCHEDD_LOG = 5000000
SCHEDD_DEBUG  = D_SCHEDD
SCHEDD_INTERVAL = 180

CLIENT_TIMEOUT = 300

#
# Negotiator Macros
#
NEGOTIATOR    = $(BIN)/LoadL_negotiator
NEGOTIATOR_DEBUG = D_FULLDEBUG D_ALWAYS D_NEGOTIATE
NEGOTIATOR_LOG = $(LOG)/NegotiatorLog
MAX_NEGOTIATOR_LOG = 5000000
NEGOTIATOR_INTERVAL = 60
MACHINE_UPDATE_INTERVAL = 600
NEGOTIATOR_PARALLEL_DEFER = 1800
NEGOTIATOR_PARALLEL_HOLD = 300
NEGOTIATOR_REDRIPE_PENDING = 1800
NEGOTIATOR_RESCAN_QUEUE = 180
NEGOTIATOR_REMOVE_COMPLETED = 0

#
# Sets the interval between recalculation of the SYSPRIO values
# for all the jobs in the queue
#
NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL = 0

#
# Starter Macros
#
```

```

STARTER = $(BIN)/LoadL_starter
STARTER_DEBUG = D_FULLDEBUG
STARTER_LOG = $(LOG)/StarterLog
MAX_STARTER_LOG = 500000

#
# LoadL_Master Macros
#
MASTER = $(BIN)/LoadL_master
MASTER_LOG = $(LOG)/MasterLog
MASTER_DEBUG = D_FULLDEBUG
MAX_MASTER_LOG = 64000
RESTARTS_PER_HOUR = 12
PUBLISH_OBITUARIES = TRUE
OBITUARY_LOG_LENGTH = 25

#
# Specify whether log files are truncated when opened
#
TRUNC_MASTER_LOG_ON_OPEN      = False
TRUNC_STARTD_LOG_ON_OPEN      = False
TRUNC_SCHEDD_LOG_ON_OPEN      = False
TRUNC_KBDD_LOG_ON_OPEN        = False
TRUNC_STARTER_LOG_ON_OPEN     = False
TRUNC_COLLECTOR_LOG_ON_OPEN   = False
TRUNC_NEGOTIATOR_LOG_ON_OPEN  = False

#      NQS Directory
#
#
# For users of NQS resources:
# Specify the directory containing qsub, qstat, qdel
#
# NQS_DIR = /usr/bin

#
# Specify Custom metric keywords
#
# CUSTOM_METRIC =
# CUSTOM_METRIC_COMMAND = $(ETC)/sw_chip_number
#
# Machine control expressions and macros
#

OpSys : $(OPSYS)
Arch : $(ARCH)
Machine : $(HOST).$(DOMAIN)

#
# Expressions used to control starting and stopping of foreign jobs
#
MINUTE = 60
HOUR = (60 * $(MINUTE))
StateTimer = (CurrentTime - EnteredCurrentState)

BackgroundLoad = 0.7
HighLoad = 1.5
StartIdleTime = 15 * $(MINUTE)
ContinueIdleTime = 5 * $(MINUTE)
MaxSuspendTime = 10 * $(MINUTE)
MaxVacateTime = 10 * $(MINUTE)

KeyboardBusy= KeyboardIdle < $(POLLING_FREQUENCY)
CPU_Idle = LoadAvg <= $(BackgroundLoad)
CPU_Busy = LoadAvg >= $(HighLoad)
# START : $(CPU_Idle) && KeyboardIdle > $(StartIdleTime)
# SUSPEND : $(CPU_Busy) || $(KeyboardBusy)
# CONTINUE : $(CPU_Idle) && KeyboardIdle > $(ContinueIdleTime)

```

## Case studies

```
# VACATE : $(StateTimer) > $(MaxSuspendTime)
# KILL   : $(StateTimer) > $(MaxVacateTime)

START   : T
SUSPEND : F
CONTINUE : T
VACATE  : F
KILL    : F

#
# Expressions used to prioritize job queue
#
# Values which can be part of the SYSPRIO expression are:
#
# QDate      Job submission time
# UserPrio    User priority
# UserSysprio System priority value based on userid (from the user
#             list file with default of 0)
# ClassSysprio System priority value based on job class (from the class
#             list file with default of 0)
# UserRunningProcs Number of jobs running for the user
# GroupRunningProcs Number of jobs running for the group
#
# The following expression is an example.
#
#SYSPRIO: (ClassSysprio * 100) + (UserSysprio * 10) + (GroupSysprio * 1) - (QDate
)
#
# The following (default) expression for SYSPRIO creates a FIFO job queue.
#
SYSPRIO: (ClassSysprio * 100) - (QDate)
#
# Expressions used to prioritize machines
#
# The following example orders machines by the load average
# normalized for machine speed:
#
#MACHPRIO: 0 - (1000 * (LoadAvg / (Cpus * Speed)))
#
# The following (default) expression for MACHPRIO orders
# machines by load average.
#
#MACHPRIO: 0 - (LoadAvg) + (MasterMachPriority * 10000)
#
# The following expression for MACHPRIO orders
# machines by increasing ammount of memory and
# decreasing node number.
#
MACHPRIO: 0 - (100 * Memory) + CustomMetric + (MasterMachPriority * 10000)

#
# The MAX_JOB_REJECT value determines how many times a job can be
# rejected before it is canceled or put on hold. The default value
# is -1, which indicates no limit to the number of times a job can be
# rejected.

#
MAX_JOB_REJECT = 0
#
# When ACTION_ON_MAX_REJECT is HOLD, jobs will be put on user hold
# when the number of rejects reaches the MAX_JOB_REJECT value. When
# ACTION_ON_MAX_REJECT is CANCEL, jobs will be canceled when the
# number of rejects reaches the MAX_JOB_REJECT value. The default
# value is HOLD.
#
ACTION_ON_MAX_REJECT = CANCEL
```



## Customer 2: circuit simulation

This customer performs CPU-intensive work in the area of circuit simulation using Electronic Design Automation (EDA).

### System configuration

The customer has 752 batch servers; 209 are dedicated to run LoadLeveler jobs 24 hours a day (the central manager is excluded). The rest are used by LoadLeveler when they are not in use by their respective owners.

The LoadLeveler administrators control all the 173 dedicated machines. That means that users cannot get onto these systems without submitting a LoadLeveler job. 117 of the dedicated machines are public schedulers. The user machines are submit-only machines, and users do not have access to their root password. If a user needs root access to his or her machine, he or she is allowed alternate root access only; he or she cannot get global root access to all the machines on site. (Site administrators use a common global root password.)

This site runs over 31,000 jobs per week and about 2,800 CPU days of resource utilization. The central manager is a RISC/System 6000 model 370 with 128MB of RAM. The batch machines are generally 80 percent busy. The central manager is about 35 percent to 70 percent busy. The central manager does not run any jobs, it just manages. All of the LoadLeveler machines run one job at a time. (That is, **MAX\_STARTERS=1**.)

This customer sees some machines in a down state occasionally. The administrator feels the CPU on these machines are too busy to get a time-slice to report its state to the central manager. However, this down state does not cause any problem for this customer.

117 public schedulers are subset of our 173 dedicated machines and are listed in the admin file.

### LoadLeveler configuration

The following figures represent sections of this customer's **LoadL\_admin** file for dedicated machines. Notice the default stanza. Also, every machine in the LoadLeveler cluster is listed in this file.

```
=====#
# type = machine default stanza
=====#

default: type = machine          # defaults for machine stanzas
central_manager = false        # no central manager on machine
schedd_host = true             # public schedd on machine
=====#
# Central Manager
=====#

mips1:  type = machine          # PRIMARY server - MANAGER  370 128M 3.2.5
central_manager = true         # runs negotiator
=====#
#                               Primary Servers
=====#

beast100: type = machine
# PRIMARY C=a/b/o/s2/t2      . . 550    128M 3.2.5
beast101: type = machine
# PRIMARY C=a/b/b1/b4/c/o/r/s/t  F . 550    128M 3.2.5
```

## Case studies

```
beast102: type = machine
# PRIMARY C=a                F . 550    128M 3.2.5
beast103: type = machine
# PRIMARY C=a                . . 550    128M 3.2.5
```

Later in the **Loadl\_admin** file, user machines are defined. Notice the default stanza.

```
#=====#
default:  type = machine          # defaults for machine stanzas
central_manager = false          # no central manager on machine
schedd_host = false              # no public schedd on machine
#=====#

agni:     type = machine
# SECONDARY server - rmkohn      550    64M 3.2.5
akama:     type = machine
# SECONDARY server - poultier   365    64M 3.2.5
alaska:    type = machine
# SECONDARY server - jcahill    340    64M 3.2.5
alcor:     type = machine
# SECONDARY server - drolson    340    64M 3.2.5
```

The following represents a local configuration file for a dedicated, public scheduler machine:

```
#                                PRIMARY LoadL SERVER ==> mips27
#
# this loadl.config.local is tuned for a machine that is part of a compute
# farm.  Interactive users are discouraged.
#
# Run up to one jobs at a time.
#
# Always start a job if there is a class available.
#
# Never suspend a job.
#
# Since jobs never get suspended they never get vacated or killed.
#

SCHEDD_RUNS_HERE    = True
STARTD_RUNS_HERE    = True

Class = { "a" "b" "b1" "b4" "c" "k" "r" "s" "t" }
Feature = { "PRI" }

MAX_STARTERS = 1

POLLING_FREQUENCY    = 30
POLLS_PER_UPDATE     = 15

START                : T
SUSPEND              : F

START_DAEMONS = True
X_RUNS_HERE   = False
```

The following represents a local configuration file for a user's machine.

```
#                                SECONDARY SERVER ==> common
#
# This loadl_config.local is tuned to be "nice" to a workstation owner
# who permits loadl jobs on his system but wants good response whenever
# he is doing his own work.
#
```

```

# Run only one LoadLeveler job at a time.
#
# Check the keyboard for activity every five seconds.
#
#
# Suspend a job if the load average exceeds 1.4
#
# Continue a job when keyboard again goes idle for 10 minutes and the load
# average is <.5

SCHEDD_RUNS_HERE = False
STARTD_RUNS_HERE = True

Class = { "a" "b" "b1" "b4" "c" "o" "r" "s" "t" }
MAX_STARTERS = 1

START          : $(FirstShift_KB9999) && $(StartS1) || $(Off_Shift) ||
$(Week_End)) && $(Mach_Idle_S)
SUSPEND        : $(CPU_Busy) || $(KeyboardBusy)
CONTINUE       : $(Mach_Idle_C)
VACATE         : ((Class == "a") && $(Vacate_A)) || ($(Vacate_ClassesB)
&& $(Vacate_B)) || $(Vacate_X)
KILL           : $(Kill_Job)

START_DAEMONS = True
X_RUNS_HERE   = True

```

---

## Customer 3: high-energy physics

This scientific customer provides experimental facilities for physicists from its 17 member states and for visiting scientists from throughout the world. The computing requirements of these users vary from mail and text processing to heavy batch and parallel processing.

### System configuration

Their processor is an SP2<sup>®</sup> using RISC System/6000 nodes linked by an internal high-speed network with a centrally managed software environment. The nodes are functionally divided into four groups of 16 each for different types of work: interactive logins, sequential job batch processing, parallel job batch processing and data, and tape and network services.

This customer uses AFS heavily. It provides the single system image for users' home directories and the files common to their experiments. Many software products are served directly out of AFS using symbolic links.

LoadLeveler provides this customer with the following facilities:

- Interactive load balancing of users across nodes on the SP2 and other UNIX services on site
- Batch services for serial compute jobs
- Scheduling for parallel applications

### LoadLeveler batch configuration

The batch configuration is designed to maximize short job turnaround while allowing the heavy CPU jobs to get good usage of the resources available.

The basic configuration uses a range of classes – short, medium, long and verylong – with a range of maximum job CPU times of from five minutes to six days. An

## Case studies

additional class, night, provides off-peak and weekend computing time on the interactive areas of the SP2 during periods of low demand. Access to this class is limited to specific users.

Users in different experiments are defined in LoadLeveler groups which provide associated queue priorities. This allows groups with a large computing budget to be given higher priorities. An automated procedure calculates each group's resource utilization over the last month and adjusts their priorities accordingly. This ensures a fair allocation of CPU time among the groups.

## LoadLeveler interactive configuration

This customer uses the Interactive Session Support facility to provide a name server which returns the least loaded node according to a site defined metric. This allows a user to be given the least loaded operational node when he or she logs in.

This metric is based on the number of logged in users, with some weight given to those using Xstations. Every few minutes, the system is scanned to evaluate the following:

*Xterminals\*3 + Telnet\*2 + Process*

Where:

- *Xterminals* is the number of users logged in from an Xstation
- *Telnet* is the number logged in via **telnet** or **rlogin**
- *Process* is the number of users who have processes running.

This metric tries to balance users across the system while providing some factor for their likely future utilization. A metric based on the CPU load average is too dependent on the current load to provide good balancing.

The metric can also be set to return a low priority if the file **/etc/iss.nologin** exists. This allows the administrator to drain the interactive use of a node if there is scheduled system maintenance. When the maintenance is completed, the file can be removed and the metric will return the correct value for the node. Users will therefore see an improved availability, since they will not be given a node that is about to shutdown.

## Processor configuration

The processors are configured as follows:

- **parallel** nodes support a mixture of short, medium, long, and verylong classes.
- **batch** nodes support the same class mix as parallel. Additional paging space is available on these nodes to provide multiple jobs running per node.
- **interactive** nodes support the night class only. The night class only allows jobs to start after 6 PM and before midnight during the week and anytime on weekends. A maximum CPU time of 8 hours ensures that the jobs are finished when the prime shift starts. This is configured using LoadLeveler's START expression:

```
Is_Weekend          = (tm_wday==0 || tm_wday==6)
Is_Start_Night_Time = (tm_hour>18)
```

```
START: $(Is_Start_Night_Time) || $(Is_Weekend)
```

---

## Customer 4: computer chip design

This customer uses EDA to perform work in the area of computer chip design.

## System configuration

The customer has seven clusters of RISC/System 6000 machines. The largest cluster has 530 machines; the smallest cluster has 87 machines. The total number of machines at this installation is over 1200.

## Interactive configuration

This customer has defined two configuration files for interactive work: one for standard workstations and one for large interactive servers. These files are meant to be tailored to machines of differing processing power.

### Standard workstation configuration

```
#####
# Description: LoadL_config.local for Standard Workstations (<370 Class)
#####
# Need 2x Paging Space to Real Memory ( minimum ) For Worst Case Of One
# Suspended and One Foreground Running Job.
#   *) All Jobs (btv,lp) Suspend on LoadAvg or Keyboard/Mouse Movement.
#####
# Class defines the permissible classes, MAX_STARTERS defines the max
# total jobs to be permitted.
#####
Class      = { "btv" "lp" }
MAX_STARTERS = 1
#####
# The next definitions are used in the expressions below to regulate the
# conditions under which jobs get started, suspended, and evicted.
#   All times are specified in units of seconds.
#####
BackgroundLoad  = 0.8
HighLoad        = 1.6
StartIdleTime   = 900
ContinueIdleTime = 900

#####
# LoadAvg is an internal variable whose value is the (Berkeley) load average
# of the machine.
#
#   CPU_Idle - No LoadL job running, or One job just finishing.
#   CPU_Busy  - One LoadL job running, second job ( Foreground or Batch )
#               starting up.
#   CPU_Max   - Two LoadL jobs running.
#####
CPU_Idle = (LoadAvg <= $(BackgroundLoad))
CPU_Busy = (LoadAvg >= $(HighLoad))

#####
# This defines a boolean "KeyboardBusy" whose value is TRUE if the keyboard
# or mouse has been used since loadl last checked. Thus if POLLING_FREQUENCY
# is 5 seconds, KeyboardBusy is TRUE if anybody has used the kbd or mouse in
# the last 5 seconds.
#####
KeyboardBusy = KeyboardIdle < $(POLLING_FREQUENCY)

#####
# This statement indicates when a job should be started on this machine
#####
Weekend = ( (tm_wday >= 6) || (tm_wday < 1) )
Day      = ( (tm_hour >= 7) && (tm_hour < 18) )
Night    = ( (tm_hour >= 18) || (tm_hour < 4) )
Inactive = ( (KeyboardIdle > $(StartIdleTime)) && $(CPU_Idle) )

HP       = ( (Class == "btv") )
LP       = ( ($(Weekend) || $(Night)) )
```

## Case studies

```
START      : ( $(HP) || $(LP)) && $(Inactive) )

#####
# The SUSPEND statement here says that a job should be suspended but not
# killed if:
#           LoadAvg >= 1.6 Or KeyboardIdle < 5
#####
SUSPEND    : ( $(CPU_Busy) || $(KeyboardBusy) )

#####
# This CONTINUE statement indicates that a suspended job should be continued
# if the cpu goes idle and the keyboard/mouse has not been used for the last
# 15 minutes.
#####
CONTINUE   : $(CPU_Idle) && KeyboardIdle > $(ContinueIdleTime)

#####
# Jobs in the SUSPEND state are never killed, after 60 minutes they are
# relocated to a different machine if possible.
#####
MaxSuspendTime = 60 * $(MINUTE)
VACATE        : $(StateTimer) > $(MaxSuspendTime)
KILL          : F

#####
# If you set START_DAEMONS to False loadl can never start on this machine.
# For example you may want to stop loadl for a couple days for maintenance
# and make sure no procedure automatically restarts it.
#####
START_DAEMONS = True

#####
# Set the maximum size each of the logs can reach before wrapping.
#####
MAX_SCHEDD_LOG   = 128000
MAX_COLLECTOR_LOG = 128000
MAX_STARTD_LOG   = 128000
MAX_SHADOW_LOG   = 128000
MAX_KBDD_LOG     = 128000
```

### Large interactive server configuration

```
#####
# Description: LoadL_config.local for Interactive Large Servers (580-590 Class)

#####
# Need 3x Real Memory To Paging Space ( minimum ) For Worst Case Of Two
# Suspended and One Foreground Running Job.
#   *) All Jobs (btv,lp) Suspend on LoadAvg or Keyboard/Mouse Movement.
#   *) Real Memory >= 192meg.
#####

#####
# Class defines the permissible classes, MAX_STARTERS defines the max
# total jobs to be permitted.
#####
Class      = { "btv" "lp" }
MAX_STARTERS = 2

#####
# The next definitions are used in the expressions below to regulate the
# conditions under which jobs get started, suspended, and evicted.
#
#   All times are specified in units of seconds.
#####
BackgroundLoad = 0.8
LowLoad        = 1.0
```

```

HighLoad      = 1.6
MaxLoad       = 2.0
StartIdleTime = 900
ContinueIdleTime = 900

#####
# LoadAvg is an internal variable whose value is the (Berkeley) load average
# of the machine.
#
#   CPU_Idle - No LoadL job running, or One job just finishing.
#   CPU_Busy - One LoadL job running, second job ( Foreground or Batch )
#               starting up.
#   CPU_Max  - Two LoadL jobs running.
#####
CPU_Idle = (LoadAvg <= $(BackgroundLoad))
CPU_Run  = (LoadAvg <= $(LowLoad))
CPU_Busy = (LoadAvg >= $(HighLoad))
CPU_Max  = (LoadAvg >= $(MaxLoad))

#####
# This defines a boolean "KeyboardBusy" whose value is TRUE if the keyboard
# or mouse has been used since loadl last checked. Thus if POLLING_FREQUENCY
# is 5 seconds, KeyboardBusy is TRUE if anybody has used the kbd or mouse in
# the last 5 seconds.
#####
KeyboardBusy = KeyboardIdle < $(POLLING_FREQUENCY)
#####
# This statement indicates when a job should be started on this machine
#####
Weekend = ( (tm_wday >= 6) || (tm_wday < 1) )
Day      = ( (tm_hour >= 7) && (tm_hour < 18) )
Night    = ( (tm_hour >= 18) || (tm_hour < 4) )
Inactive1 = ( (KeyboardIdle > $(StartIdleTime)) )
Inactive2 = ( (KeyboardIdle > $(ContinueIdleTime)) )

HP       = ( (Class == "btv") )
LP       = ( (Class == "lp") && $(CPU_Idle) )

START    : ( ($(HP) || $(LP)) && $(Inactive1) )

#####
# The SUSPEND statement here says that a job should be suspended but not
# killed if:
#           KeyboardIdle < 5           Or
#           lp Class And LoadAvg >= 1.6 Or
#           btv Class And LoadAvg >= 2.0
#####
SUSPEND  : ( ( (Class == "lp") && $(CPU_Busy) ) || \
( (Class == "btv") && $(CPU_Max) ) || \
( $(KeyboardBusy) ) ) )

#####
# This CONTINUE statement indicates that a suspended job should be continued
# if:
#           lp Class And LoadAvg <= 0.8 And KeyboardIdle > 15 min Or
#           btv Class And LoadAvg <= 1.0 And KeyboardIdle > 15 min
#####
CONTINUE : ( ( (Class == "lp") && $(CPU_Idle) && $(Inactive2) ) || \
( (Class == "btv") && $(CPU_Run) && $(Inactive2) ) ) )

#####
# Jobs in the SUSPEND state are never killed, after 60 minutes they are
# relocated to a different box if possible.
#####
MaxSuspendTime = 60 * $(MINUTE)
VACATE         : $(StateTimer) > $(MaxSuspendTime)
KILL           : F

```

## Case studies

```
#####  
# If you set START_DAEMONS to False loadl can never start on this machine.  
# For example you may want to stop loadl for a couple days for maintenance  
# and make sure no procedure automatically restarts it.  
#####  
START_DAEMONS = True  
  
#####  
# Set the maximum size each of the logs can reach before wrapping.  
#####  
MAX_SCHEDD_LOG    = 128000  
MAX_COLLECTOR_LOG = 128000  
MAX_STARTD_LOG    = 128000  
MAX_SHADOW_LOG    = 128000  
MAX_KBDD_LOG      = 128000
```

## Batch configuration

The following configuration file defines dedicated batch machines. Notice, however, that jobs in the lp class will suspend when a machine becomes too busy. So in this sense, the machines are not fully dedicated.

```
#####  
# Description: LoadL_config.local for Large Batch Servers ( 580 - 590 Class )  
#####  
# Need 3x Real Memory To Paging Space ( minimum ) For Worst Case Of One  
# Suspended and Two Foreground Running Job.  
# *) High Priority Jobs (btv) Never Suspend.  
# *) Job Suspension (lp) Based on LoadAvg Only.  
# *) Real Memory >= 192meg.  
#####  
  
#####  
# Class defines the permissible classes, MAX_STARTERS defines the max  
# total jobs to be permitted.  
#####  
Class      = { "btv" "lp" }  
MAX_STARTERS = 2  
  
#####  
# The next definitions are used in the expressions below to regulate the  
# conditions under which jobs get started, suspended, and evicted.  
#  
# All times are specified in units of seconds.  
#####  
BackgroundLoad = 0.5  
HighLoad       = 1.6  
StartIdleTime  = 900  
ContinueIdleTime = 900  
  
#####  
# LoadAvg is an internal variable whose value is the (Berkeley) load average  
# of the machine.  
#  
# CPU_Idle - No LoadL job running, or One job just finishing.  
# CPU_Busy - One LoadL job running, second job ( Foreground or Batch )  
#           starting up.  
# CPU_Max  - Two LoadL jobs running.  
#####  
CPU_Idle = (LoadAvg <= $(BackgroundLoad))  
CPU_Busy = (LoadAvg >= $(HighLoad))  
  
#####  
# This defines a boolean "KeyboardBusy" whose value is TRUE if the keyboard  
# or mouse has been used since loadl last checked. Thus if POLLING_FREQUENCY  
# is 5 seconds, KeyboardBusy is TRUE if anybody has used the kbd or mouse in  
# the last 5 seconds.  
#####
```



```

KeyboardBusy = KeyboardIdle < $(POLLING_FREQUENCY)

#####
# This statement indicates when a job should be started on this machine
#####
HP      = ( (Class == "btv") )
LP      = ( (Class == "lp") && $(CPU_Idle) )

START   : ( $(HP) || $(LP) )

#####
# The SUSPEND statement here says that a "lp" job should be suspended but not
# killed if a high priority job starts up or a foreground job causes the
# Loadavg to be greater than CPU_Busy ( 1.6 ).
#####
SUSPEND : (Class == "lp") && $(CPU_Busy)

#####
# This CONTINUE statement indicates that a suspended job should be continued
# if the cpu goes idle and the keyboard/mouse has not been used for the last
# 15 minutes.
#####
CONTINUE : $(CPU_Idle) && KeyboardIdle > $(ContinueIdleTime)

#####
# Jobs in the SUSPEND state are never killed, after 60 minutes they are
# relocated to a different box if possible.
#####
MaxSuspendTime = 60 * $(MINUTE)
VACATE         : $(StateTimer) > $(MaxSuspendTime)
KILL           : F

#####
# If you set START_DAEMONS to False loadl can never start on this machine.
# For example you may want to stop loadl for a couple days for maintenance
# and make sure no procedure automatically restarts it.
#####
START_DAEMONS = True

#####
# Set the maximum size each of the logs can reach before wrapping.
#####
MAX_SCHEDD_LOG   = 128000
MAX_COLLECTOR_LOG = 128000
MAX_STARTD_LOG   = 128000
MAX_SHADOW_LOG   = 128000
MAX_KBDD_LOG     = 128000

```

## Configuration for a machine that schedules (but doesn't run) jobs

The following statements define a machine that schedules jobs but does not run jobs. Notice that the schedd daemon is never forced to *not* run.

```

#
# This loadl local configuration file is set up to make a machine a
# submitter only.
#
# No jobs are allowed to run on this system.
#
MAX_STARTERS          = 0

START                 : F
#
# If you set START_DAEMONS to False loadl can never start on this machine.

```

## Case studies

```
# For example you may want to stop loadl for a couple days for maintenance
# and make sure no procedure automatically restarts it.
#
START_DAEMONS          = True
```

---

## Appendix C. Troubleshooting

---

### Troubleshooting LoadLeveler

This chapter is divided into the following sections:

- “Frequently Asked Questions”, which contains answers to questions frequently asked by LoadLeveler customers. This section focuses on answers that may help you get out of problem situations. The questions and answers are organized into the following categories:
  - **Jobs submitted to LoadLeveler do not run.** See “Why won’t my job run?” for more information.
  - **One or more of your machines goes down.** See “What happens to running jobs when a machine goes down?” on page 438 for more information.
  - **The central manager is not operating.** See “What happens if the central manager isn’t operating?” on page 440 for more information.
  - **Miscellaneous questions.** See “Other questions” on page 442 for more information.
- “Helpful hints” on page 443, which contains tips on running LoadLeveler, including some productivity aids.
- “Getting help from IBM” on page 447, which tells you how to contact IBM for assistance.

It is helpful to create error logs when you are diagnosing a problem. See to “Step 12: Record and control log files” on page 352 for information on setting up error logs.

### Frequently Asked Questions

This section contains answers to questions frequently asked by LoadLeveler customers.

#### Why won’t my job run?

If you submitted your job and it is in the LoadLeveler queue but has not run, issue **llq -s** first to help diagnose the problem. If you need more help diagnosing the problem, refer to the following table:

Why Your Job May Not Be Running:	Possible Solution
Job requires specific machine, operating system, or other resource.	<ul style="list-style-type: none"><li>• Does the resource exist in the LoadLeveler cluster? If yes, wait until it becomes available.</li></ul> <p>Check the GUI to compare the job requirements to the machine details, especially <b>Arch</b>, <b>OpSys</b>, and <b>Class</b>. Ensure that the spelling and capitalization matches.</p>
Job requires specific job class	<ul style="list-style-type: none"><li>• Is the class defined in the administration file? Use <b>llclass</b> to determine this. If yes,</li><li>• 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.</li></ul>
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.

## Troubleshooting

Why Your Job May Not Be Running:	Possible Solution
The start expression evaluates to false.	Examine the configuration files (both <b>LoadL_config</b> and <b>LoadL_config.local</b> ) to determine the <b>START</b> 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
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 <b>llprio</b> 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 <b>LoadL_config</b> file. The default polling frequency is five minutes.
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.

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 2. LoadLeveler command line interface” on page 19 for information on these commands.

### Why won't my parallel job run?

If you submitted your parallel job and it is in the LoadLeveler queue but has not run, issue **llq -s** first to help diagnose the problem. If issuing this command does not help, refer to the previous table and to the following table for more information:

Why Your Job May 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 <b>requirements</b> statement specifies a pool which is invalid or not available.	The specified pool must be valid and available.
The adapter specified in the <b>requirements</b> statement or the <b>network</b> statement identifies an adapter which is invalid or not available.	The specified adapter must be valid and available.
PVM3 is not installed	PVM3 must be installed on any machine you wish to use for pvm. The PVM3 system itself is not supplied with LoadLeveler.
You are already running a PVM3 job on one of the LoadLeveler machines.	PVM3 restrictions prevent a user from running more than one pvm daemon per user per machine. If you want to run pvm3 jobs on LoadLeveler, you must not run any pvm3 jobs outside of LoadLeveler control on any machine being managed by LoadLeveler.
The <b>parallel_path</b> keyword in your job command file is incorrect.	Use <b>parallel_path</b> to inform LoadLeveler where binaries that run your pvm tasks are for the pvm_spawn() command. If this is incorrect, the job may not run.
The <b>pvm_root</b> keyword in the administration file is incorrect.	This keyword corresponds to the pvm <b>ep</b> keyword and is required to tell LoadLeveler where the pvm system is installed.

Why Your Job May Not Be Running	Possible Solution
The file <code>/tmp/pvmd.userid</code> exists on some LoadLeveler machine but no PVM jobs are running.	If PVM3 exits unexpectedly, it will not properly clean up after itself. Although LoadLeveler attempts to clean up after pvm, some situations are ambiguous and you may have to remove this file yourself. Check all the systems specified as being capable of running PVM3, and remove this file if it exists.

**Gang scheduler checklist:** Before running the Gang Scheduler, verify that:

- **MACHINE\_AUTHENTICATE** is set to TRUE
- **PROCESS\_TRACKING** is set to TRUE
- **MAX\_SMP\_TASK** is set correctly (default value set to the number of processors)
- NTP daemons are running on all nodes
- Applications are compiled with the multi-threaded library

**Note:** Unlike the Default or Backfill schedulers, jobs running under Gang scheduler need an additional transaction to pass the matrix among the daemons. This transaction has to be successfully received by all nodes. If the matrix fails to propagate, jobs could be stuck in the starting state until the matrix is received by all the nodes in subsequent retries. Turning on the debug flag `D_HIERARCHICAL` generates messages to help determine why the matrix is not received properly.

**Common set up problems with parallel jobs:** This section 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 **-eulib 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*.
- For PVM:
  - Specify the parallel job as the executable. Do *not* specify PVM as the executable.
  - Compile only with the supported level of PVM.
  - Specify only **pvm3** as the *job\_type*.

**PVM problem determination:** If LoadLeveler is to manage PVM jobs on a machine for a user, that user should not attempt to run PVM jobs on that machine outside of LoadLeveler control. Because of PVM restrictions, only a single PVM daemon per user per machine is permitted. If a user tries to run PVM jobs without using LoadLeveler and LoadLeveler later attempts to start a job for that user on the same machine, LoadLeveler may not be able to start PVM for the job. This will cause the LoadLeveler job to be canceled.

If a PVM job submitted through LoadLeveler is rejected, it is probably because PVM was not correctly terminated the last time it ran on the rejecting machine.

## Troubleshooting

LoadLeveler attempts to handle this by making sure that it cleans up PVM jobs when they complete, but remember that you may need to clean up after the job yourself. If a machine refuses to start a PVM job, check the following:

- See if there is a process with the name **pvmd** running on the machine in question under the id of the user whose job will not start. Stop the process by issuing:

```
ps -ef | grep pvmd  
kill -TERM pid
```

Do not use either of the following variations to stop the daemon because this will prevent **pvmd** from cleaning up and jobs will still not start:

```
kill -9 pid  
kill -KILL pid
```

- If there is no **pvmd** process running, see if there is a file called **/tmp/pvmd.userid**, where *userid* is the ID of the user whose job will not start. If the file exists, remove it.

### 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
```

### 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 the default, -1, the job will be rejected an unlimited number of times. Try resetting this keyword to some finite number. Also, check the setting of the **ACTION\_ON\_MAX\_REJECT** keyword. These keywords are described in "Step 17: Specify additional configuration file keywords" on page 370.

### 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.

If	Then
The network goes down but the machines are still running	If the network goes down but the machines are still running, when LoadLeveler is restarted, it looks for all jobs that were marked running when it went down. On the machine where the job is running, the startd 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 schedd is running, schedd queues a transaction to the startd to re-establish the state of the job. This transaction stays queued until the state is established. Until that time, LoadLeveler 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 schedd and startd are recorded on disk. This ensures complete delivery of messages and prevents incorrect decisions based on incomplete state information.
The machine with startd goes down.	Because job state is maintained on disk in startd, when LoadLeveler is restarted it can forward correct status to the rest of LoadLeveler. 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 LoadLeveler failed, it is often possible to "find" the job if it is still running and resume management of it. In this case LoadLeveler sends JOB RUNNING to the schedd 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 schedd 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>
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 441.</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.

## Troubleshooting

**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. The following diagram illustrates how a machine can become the alternate central manager:

The diagram illustrates that Machine Z is the primary central manager but Machine

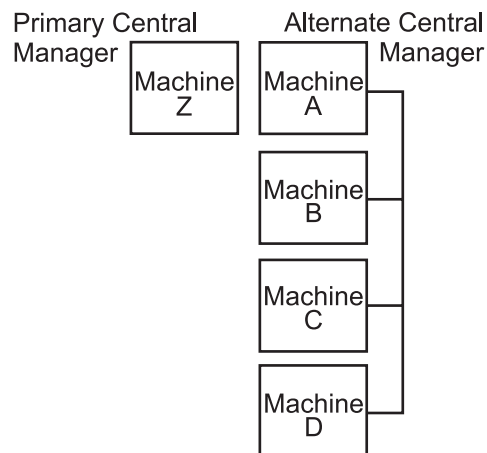


Figure 50. When the primary central manager is unavailable

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.

The following diagram illustrates how multiple central managers can function within the same LoadLeveler pool:



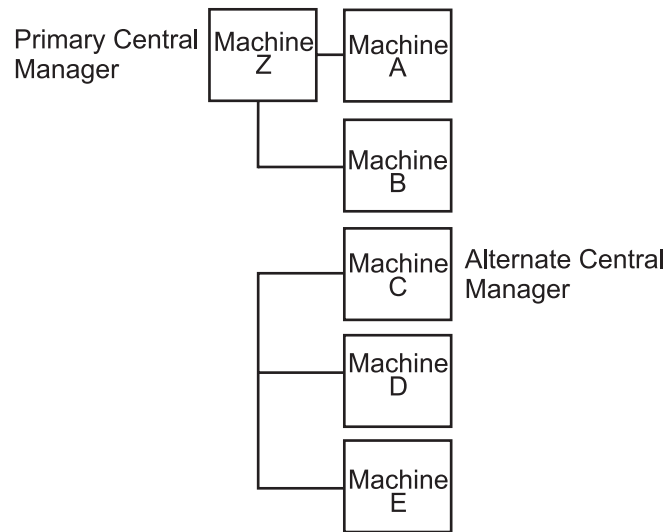


Figure 51. Multiple central managers

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 re-joined 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 “Step 1: Specify machine stanzas” on page 310.

### 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 (see below) is taking place.

## Troubleshooting

3. Reconfigure the central manager node so that it recognizes the “fenced” node. From the central manager machine issue **llctl reconfig**.
4. Issue **llctl -h host purgeschedd** to purge all jobs scheduled by the schedd on the failed node.
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.

## Other questions

**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.

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.

## Helpful hints

This section 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. You should activate as many schedd daemons as there are jobs likely to be running at any one time. When you do this, each schedd handles the dispatching of one parallel job.
- 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 PSSP **dsh** command; for example, **dsh llctl -g reconfig**. This command also allows you to limit your operation to a subset of machines by defining other working collectives.

### Hints 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: Sun Apr 30 11:40:41 1996
Started   at: Sun Apr 30 11:45:00 1996
Exited    at: Sun Apr 30 12:49:10 1996
```

## Troubleshooting

```
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 two fields above.

### 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 two fields above.

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_day >= 1700) || (tm_day <= 0800)
SUSPEND: (tm_day > 0800) && (tm_day < 1700)
CONTINUE: (tm_day >= 1700) || (tm_day <= 0800)
```

**Controlling the mix of idle and running jobs:** Three keywords determine the mix of idle and running jobs for a user. By a running job, we mean a job that is in one of the following states: Checkpointing, Preempted, Preempt Pending, Resume Pending, Running, Pending, or Starting. These keywords, which are described in detail in "Step 2: Specify user stanzas" on page 316, are:

### **maxqueued**

Controls the number of jobs in any of these states: Idle, Running, Pending, or Starting.

### **maxjobs**

Controls the number of jobs in any of these states: Running, Pending, or Starting; thus it controls a subset of what **maxqueued** controls. **maxjobs** effectively controls the number of jobs in the Running state, since Pending and Starting are usually temporary states.

### **maxidle**

Controls the number of jobs in any of these states: Idle, Pending, or Starting; thus it controls a subset of what **maxqueued** controls. **maxidle** effectively controls the number of jobs in the Idle state, since Pending and Starting are usually temporary states.

*What happens when you submit a job:* For a user's job to be allowed into the job queue, 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. Also, the total idle jobs (those in the Idle, Pending, and Starting states) must be less than the **maxidle** value for the user. If either of these constraints are at the maximum, the job 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. (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
# @
```

## Troubleshooting

```
# @ 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

**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).

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*).

## 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 (5765-E69).
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:

```
llctl 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).





---

## Bibliography

This bibliography helps you find product documentation related to the RS/6000 SP hardware and software products.

You can find most of the IBM product information for RS/6000 SP products on the World Wide Web. Formats for both viewing and downloading are available.

PSSP documentation is shipped with the PSSP product in a variety of formats and can be installed on your system. The man pages for public code that PSSP includes are also available online.

Finally, this bibliography contains a list of non-IBM publications that discuss parallel computing and other topics related to the RS/6000 SP.

---

## Information formats

Documentation supporting RS/6000 SP software licensed programs is no longer available from IBM in hardcopy format. However, you can view, search, and print documentation in the following ways:

- On the World Wide Web
- Online from the product media or the SP Resource Center

---

## Finding documentation on the World Wide Web

Most of the RS/6000 SP hardware and software books are available from the IBM Web site at:

<http://www.ibm.com/servers/eserver/pseries>

You can view a book or download a Portable Document Format (PDF) version of it. At the time this manual was published, the Web address of the "RS/6000 SP Hardware and Software Books" page was:

[http://www.rs6000.ibm.com/resource/aix\\_resource/sp\\_books](http://www.rs6000.ibm.com/resource/aix_resource/sp_books)

However, the structure of the RS/6000 Web site can change over time.

---

## Accessing PSSP documentation online

On the same medium as the PSSP product code, IBM ships PSSP man pages, HTML files, and PDF files. In order to use these publications, you must first install the **spp.docs** file set.

To view the PSSP HTML publications, you need access to an HTML document browser such as Netscape. The HTML files and an index that links to them are installed in the **/usr/lpp/spp/html** directory. Once installed, you can also view the HTML files from the RS/6000 SP Resource Center.

If you have installed the SP Resource Center on your SP system, you can access it by entering the **/usr/lpp/spp/bin/resource\_center** command. If you have the SP Resource Center on CD-ROM, see the **readme.txt** file for information about how to run it.

To view the PSSP PDF publications, you need access to the Adobe Acrobat Reader. The Acrobat Reader is shipped with the AIX Bonus Pack and is also freely available for downloading from the Adobe Web site at:

<http://www.adobe.com>

To successfully print a large PDF file (approximately 300 or more pages) from the Adobe Acrobat reader, you may need to select the "Download Fonts Once" button on the Print window.

---

## Manual pages for public code

The following manual pages for public code are available in this product:

**SUP**    `/usr/lpp/ssp/man/man1/sup.1`

**Perl (Version 4.036)**

`/usr/lpp/ssp/perl/man/perl.man`

`/usr/lpp/ssp/perl/man/h2ph.man`

`/usr/lpp/ssp/perl/man/s2p.man`

`/usr/lpp/ssp/perl/man/a2p.man`

Manual pages and other documentation for **Tcl**, **TclX**, **Tk**, and **expect** can be found in the compressed **tar** files located in the `/usr/lpp/ssp/public` directory.

---

## RS/6000 SP planning publications

This section lists the IBM product documentation for planning for the IBM RS/6000 SP hardware and software.

*IBM RS/6000 SP:*

- *Planning, Volume 1, Hardware and Physical Environment*, GA22-7280
- *Planning, Volume 2, Control Workstation and Software Environment*, GA22-7281

---

## RS/6000 SP hardware publications

This section lists the IBM product documentation for the IBM RS/6000 SP hardware.

*IBM RS/6000 SP:*

- *Planning, Volume 1, Hardware and Physical Environment*, GA22-7280
- *Planning, Volume 2, Control Workstation and Software Environment*, GA22-7281
- *Installation and Relocation*, GA22-7441
- *System Service Guide*, GA22-7442
- *SP Switch Service Guide*, GA22-7443
- *SP Switch2 Service Guide*, GA22-7444
- *Uniprocessor Node Service Guide*, GA22-7445
- *604 and 604e SMP High Node Service Guide*, GA22-7446
- *SMP Thin and Wide Node Service Guide*, GA22-7447
- *POWER3 SMP High Node Service Guide*, GA22-7448

---

## RS/6000 SP Switch Router publications

The RS/6000 SP Switch Router is based on the Ascend GRF switched IP router product from Lucent Technologies. You can order the SP Switch Router as the IBM 9077.

The following publications are shipped with the SP Switch Router. You can also order these publications from IBM using the order numbers shown.

- *Ascend GRF GateD Manual*, GA22-7327
- *Ascend GRF 400/1600 Getting Started*, GA22-7368
- *Ascend GRF Configuration and Management*, GA22-7366
- *Ascend GRF Reference Guide*, GA22-7367
- *SP Switch Router Adapter Guide*, GA22-7310

---

## Related hardware publications

For publications on the latest IBM pSeries and RS/6000 hardware products, see the Web site:

[http://www.ibm.com/servers/eserver/pseries/library/hardware\\_docs/](http://www.ibm.com/servers/eserver/pseries/library/hardware_docs/)

That site includes links to the following:

- General service documentation
- Guides by system (pSeries and RS/6000)
- Installable options
- IBM Hardware Management Console for pSeries guides

---

## RS/6000 SP software publications

This section lists the IBM product documentation for software products related to the IBM RS/6000 SP. These products include:

- IBM Parallel System Support Programs for AIX (PSSP)
- IBM LoadLeveler for AIX 5L (LoadLeveler)
- IBM Parallel Environment for AIX (Parallel Environment)
- IBM General Parallel File System for AIX (GPFS)
- IBM Engineering and Scientific Subroutine Library (ESSL) for AIX
- IBM Parallel ESSL for AIX
- IBM High Availability Cluster Multi-Processing for AIX (HACMP)

### PSSP Publications

*IBM RS/6000 SP:*

- *Planning, Volume 2, Control Workstation and Software Environment*, GA22-7281

*PSSP:*

- *Installation and Migration Guide*, GA22-7347
- *Administration Guide*, SA22-7348
- *Managing Shared Disks*, SA22-7349
- *Diagnosis Guide*, GA22-7350
- *Command and Technical Reference*, SA22-7351

- *Messages Reference*, GA22-7352
- *Implementing a Firewalled RS/6000 SP System*, GA22-7874

*RS/6000 Cluster Technology (RSCT):*

- *Event Management Programming Guide and Reference*, SA22-7354
- *Group Services Programming Guide and Reference*, SA22-7355
- *First Failure Data Capture Programming Guide and Reference*, SA22-7454

## **LoadLeveler Publications**

*LoadLeveler:*

- *Using and Administering*, SA22-7881
- *Diagnosis and Messages Guide*, GA22-7882
- *Installation Memo*, GI11-2819

## **GPFS Publications**

*GPFS:*

- *Problem Determination Guide*, GA22-7434
- *Administration and Programming Reference*, SA22-7452
- *Concepts, Planning, and Installation*, GA22-7453

## **Parallel Environment Publications**

*Parallel Environment:*

- *Installation Guide*, GA22-7418
- *Messages*, GA22-7419
- *MPI Programming Guide*, SA22-7422
- *MPI Subroutine Reference*, SA22-7423
- *Hitchhiker's Guide*, SA22-7424
- *Operation and Use, Volume 1*, SA22-7425
- *Operation and Use, Volume 2*, SA22-7426

## **Parallel ESSL and ESSL Publications**

- *ESSL Products: General Information*, GC23-0529
- *Parallel ESSL: Guide and Reference*, SA22-7273
- *ESSL: Guide and Reference*, SA22-7272

## **HACMP Publications**

*HACMP:*

- *Concepts and Facilities*, SC23-4276
- *Planning Guide*, SC23-4277
- *Installation Guide*, SC23-4278
- *Administration Guide*, SC23-4279
- *Troubleshooting Guide*, SC23-4280
- *Programming Locking Applications*, SC23-4281
- *Programming Client Applications*, SC23-4282
- *Master Index and Glossary*, SC23-4285

- *HANFS for AIX Installation and Administration Guide*, SC23-4283
- *Enhanced Scalability Installation and Administration Guide, Volume 1*, SC23-4284
- *Enhanced Scalability Installation and Administration Guide, Volume 2*, SC23-4306

---

## AIX publications

You can find links to the latest AIX publications on the Web site:

<http://www.ibm.com/servers/aix/library/techpubs.html>

---

## DCE publications

The DCE library consists of the following books:

- *IBM DCE for AIX: Administration Commands Reference*
- *IBM DCE for AIX: Administration Guide—Introduction*
- *IBM DCE for AIX: Administration Guide—Core Components*
- *IBM DCE for AIX: DFS Administration Guide and Reference*
- *IBM DCE for AIX: Application Development Guide—Introduction and Style Guide*
- *IBM DCE for AIX: Application Development Guide—Core Components*
- *IBM DCE for AIX: Application Development Guide—Directory Services*
- *IBM DCE for AIX: Application Development Reference*
- *IBM DCE for AIX: Problem Determination Guide*
- *IBM DCE for AIX: Release Notes*

You can view a DCE book or download a Portable Document Format (PDF) version of it from the IBM DCE Web site at:

<http://www.ibm.com/software/network/dce/library>

---

## Redbooks

IBM's International Technical Support Organization (ITSO) has published a number of redbooks related to the RS/6000 SP. For a current list, see the ITSO Web site at:

<http://www.ibm.com/redbooks>

---

## Non-IBM publications

Here are some non-IBM publications that you might find helpful.

- Almasi, G., Gottlieb, A., *Highly Parallel Computing*, Benjamin-Cummings Publishing Company, Inc., 1989.
- Foster, I., *Designing and Building Parallel Programs*, Addison-Wesley, 1995.
- Gropp, W., Lusk, E., Skjellum, A., *Using MPI*, The MIT Press, 1994.
- Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard, Version 1.1*, University of Tennessee, Knoxville, Tennessee, June 6, 1995.
- Message Passing Interface Forum, *MPI-2: Extensions to the Message-Passing Interface, Version 2.0*, University of Tennessee, Knoxville, Tennessee, July 18, 1997.
- Ousterhout, John K., *Tcl and the Tk Toolkit*, Addison-Wesley, Reading, MA, 1994, ISBN 0-201-63337-X.
- Pfister, Gregory, F., *In Search of Clusters*, Prentice Hall, 1998.

- Barrett, D., Silverman, R., *SSH The Secure Shell The Definitive Guide*, O'Reilly, 2001.

---

## 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 references 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 on any of IBM's 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 in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

Director of Licensing  
IBM Corporation  
500 Columbus Avenue  
Thornwood, NY 10594  
USA

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, 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  
Mail Station P131  
2455 South Road  
Poughkeepsie, NY 12601-5400  
USA  
Attention: Information Request

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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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 product and cannot confirm the accuracy or 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 illustrates 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. 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 IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:



---

## Trademarks and service marks

The following terms are trademarks of the IBM Corporation in the United States and/or other countries or both:

AIX

IBM

LoadLeveler

RISC System/6000

RISC System/6000 Scalable POWERparallel Systems

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names, which may be denoted by a double asterisk (\*\*), may be trademarks or service marks of others.

LoadLeveler incorporates Condor, which was developed at the University of Wisconsin-Madison, and uses it with the permission of its authors.



---

# Glossary

---

This section contains some of the terms that are commonly used in the LoadLeveler books and in this book in particular.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the American National Standard *Vocabulary for Information Processing* (Copyright 1970 by American National Standards Institute, Incorporated), which was prepared by Subcommittee X3K5 on Terminology and Glossary of the American National Standards Committee X3. ANSI definitions are preceded by an asterisk (\*).

Other definitions in this glossary are taken from *IBM Vocabulary for Data Processing, Telecommunications, and Office Systems* (GC20-1699), *IBM DATABASE 2 Application Programming Guide for TSO Users* (SC26-4081), and *Internetworking With TCP/IP, Principles, Protocols, and Architecture*, by Douglas Comer, Copyright 1988 by Prentice Hall, Incorporated

## A

**AFS.** Andrew File System.

**AIX.** Abbreviation for Advanced Interactive Executive, IBM's licensed version of the UNIX operating system. AIX is particularly suited to support technical computing applications, including high function graphics and floating point computations.

**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.** A general purpose programming language. It was formalized by ANSI standards committee for the C language (X3J11) in 1984 and by Uniform in 1983.

**client.** \*(1) A function that requests services from a server, and makes them available to the user. \*(2) An

address space in MVS that is using TCP/IP services.

\*(3) A term used in an environment to identify a machine that uses the resources of the network.

**cluster.** (1) A group of processors interconnected through a high speed network that can be used for high performance computing. (2) A group of jobs submitted from the same job command file. (3) A set of machines with something in common between them. This commonality could be that they are all backed up by one machine or they are all in the LoadLeveler administration file.

## D

**daemon.** A process, not associated with a particular user, that performs system-wide functions such as administration and control of networks, execution of time-dependent activities, line printer spooling, and so on.

**datagram.** A protocol known as the User Datagram Protocol (UDP). It is an internet standard protocol that allows an application program on one machine to send a datagram to an application program on another machine. UDP uses the Internet Protocol to deliver datagrams. Conceptually, the important difference between UDP and IP is that UDP messages include a protocol port number, allowing the sender to distinguish among multiple destinations (application programs) on the remote machines. In practice, UDP also includes a checksum over the data being sent.

**DCE.** Distributed Computing Environment.

**default.** An alternative value, attribute, or option that is assumed when none has been specified.

**DFS.** Distributed File System. A subset of the IBM Distributed Computing Environment.

## H

**host.** A computer connected to a network, and providing an access method to that network. A host provides end-user services.

## M

**menu.** A display of a list of available functions for selection by the user.

**Motif.** The UNIX industry's standard user interface, originally developed by the Open Systems Foundation. Motif is based on the X-Window system and is a Presentation Manager look-alike. Motif is available for all IBM AIX workstations.

## N

**network.** An interconnected group of nodes, lines, and terminals. A network provides the ability to transmit data to and receive data from other systems and users.

**NFS.** Network File System.

**node.** In a network, the point where one or more functional units interconnect transmission lines. A computer location defined in a network.

**NQS.** Network Queueing System.

## P

**parameter.** \*(1) A variable that is given a constant value for a specified application and that may denote the application. \*(2) An item in a menu for which the operator specifies a value or for which the system provides a value when the menu is interpreted. \*(3) A name in a procedure that is used to refer to an argument that is passed to the procedure. \*(4) A particular piece of information that a system or application program needs to process a request.

**process.** \*(1) A unique, finite course of events defined by its purpose or by its effect, achieved under defined conditions. \*(2) Any operation or combination of operations on data. \*(3) A function being performed or waiting to be performed. \*(4) A program in operation. For example, a daemon is a system process that is always running on the system.

## S

**SDR.** Abbreviation for System Data Repository. A repository of system information describing SP hardware and operating characteristics.

**server.** (1) A function that provides services for users. A machine may run client and server processes at the same time. (2) A machine that provides resources to the network. It provides a network service, such as disk storage and file transfer, or a program that uses such a service.

**shell.** The shell is the primary user interface for the UNIX operating system. It serves as command language interpreter, programming language, and allows foreground and background processing. There are three different implementations of the shell concept: Bourne, C and Korn.

**stream.** An internet standard transport level protocol that provides the reliable, full duplex, stream service on which many application protocols depend. TCP allows a process on one machine to send a stream of data to a process on another. It is connection-oriented in the sense that before transmitting data, participants must establish a connection. Software implementing TCP

usually resides in the operating system and uses the IP protocol to transmit information across the Internet. It is possible to terminate (shut down) one direction of flow across a TCP connection, leaving a one-way (simplex) connection. The Internet protocol suite is often referred to as TCP/IP because TCP is one of the two most fundamental protocols.

**System Administrator.** The user who is responsible for setting up, modifying, and maintaining LoadLeveler.

## U

**user.** Anyone who is using LoadLeveler.

## W

**working directory.** All files without a fully qualified path name are relative to this directory.

**workstation.** \*(1) A configuration of input/output equipment at which an operator works. \*(2) A terminal or microcomputer, usually one that is connected to a mainframe or to a network, at which a user can perform applications.

---

# Index

## Special Characters

/etc/LoadL.cfg file 43, 63  
/etc/services file 355  
.llrc script 129  
!var 91  
\$var 91

## Numerics

64-bit  
    command line interfaces  
        llclass -l 397  
        llq -l 398  
        llstatus -l 397  
    keywords supported  
        administration file 396  
        configuration file 396  
        job command file 395  
    support for accounting functions 399  
    support for command line interfaces 397  
    support for command line interfaces and GUI 397  
    support for GUI 398  
    support for LoadLeveler APIs 399

## A

account 316  
account\_no 85  
accounting  
    API 215  
    collecting data 75  
    in job command file 85  
    llacctmrg command 138  
    llsummary command 202  
    reports 77  
accounting, job setup 78, 374  
accounting functions  
    64-bit support 399  
ACCT 349  
ACCT\_VALIDATION 215, 350  
ACTION\_ON\_MAX\_REJECT 370  
ACTION\_ON\_SWTICH\_TABLE\_ERROR 371  
adapter  
    dedicated 97  
    shared 97  
    specifying in job command file 96, 101  
adapter information  
    extracting from SDR 155  
adapter\_name 332  
adapter stanza keywords  
    adapter\_name 332  
    css\_type 332  
    interface\_address 333  
    interface\_name 333  
    multilink\_address 333  
    multilink\_list 333  
    network\_type 333

adapter stanza keywords *(continued)*  
    switch\_node\_number 333  
adapter stanzas  
    examples 333  
    format 332  
adapter\_stanzas 311  
admin 320, 330  
ADMIN\_FILE 352  
admin keywords 330  
    class stanza 320  
administering LoadLeveler  
    administering LoadLeveler 62  
    administration file 59, 61  
    customizing the administration file 310  
    LoadL\_admin file 62  
    overview 59  
    Quick Set Up 61  
    stanzas 310  
administration file  
    customizing 310  
    keywords 111  
    multiple statements 393  
    structure and syntax 62  
administration file keyword details 390  
administration file keyword summary 390  
administration file keywords 389  
administrative actions  
    GUI 29  
administrators 61  
AFS authentication 371  
AFS authentication user exit 284  
AFS\_GETNEWTOKEN 371  
AFS token handling 284  
alias 311  
alternate central manager 350  
API scheduler 335, 337  
APIs 215  
    Gang scheduling 394  
    ll\_ckpt 219  
    ll\_init\_ckpt 218  
    ll\_set\_ckpt\_callbacks 221  
    ll\_unset\_ckpt\_callbacks 222  
APIs, LoadLeveler 33  
appendix contents 403  
application programming interfaces  
    accessing LoadLeveler objects 223  
    accounting 215  
    checkpointing serial jobs 218  
    job control 270  
    ll\_error 259  
    querying jobs and machines 265  
    running parallel jobs 260  
    scheduling 270  
    submitting jobs 268  
    workload management 270  
Application Programming Interfaces 33, 215  
application support 384

- Arch
  - requirement in job command file 101
  - variable 66
- ARCH configuration file keyword 339
- arguments 85
- attributes (of job step)
  - change 168
- authentication process, DCE 284
- authentication programs 283, 284

## B

- backfill scheduler 336
- BACKFILL scheduler 335
- basic gui functions 23
- basics, LoadLeveler 4
- batch parallel jobs
  - naming files for checkpointing 358
- BIN 352
- blocking 50, 85
- blocking factor 50
- building jobs
  - using a job command file 39
  - using the GUI 293
- building jobs using the GUI 24

## C

- canceled job state 134
- cancelling jobs
  - using llcancel 406
  - using the GUI 305
- capture data
  - GUI 31
- central manager 7, 308, 311, 350, 406, 440
- CENTRAL\_MANAGER\_HEARTBEAT\_INTERVAL 351
- central\_manager keyword 312
- CENTRAL\_MANAGER\_TIMEOUT 351
- changing attributes of job steps
  - using llmodify 168
- changing job priority
  - example 406
  - using llprio 171
  - using the GUI 305
- checklist
  - Gang scheduler 437
  - parallel jobs 437
- checkpoint 86
  - file naming 360
  - limitations 360
  - removing old files 364
  - restarting a job 438
- checkpoint, take 306
- checkpoint and restart limitations 360
- checkpoint files, removing 364
- checkpoint keywords 357
  - administration file 357
  - configuration file 357
  - job command file 357
- checkpointing
  - API 218

- checkpointing (*continued*)
  - how to checkpoint a job 362
  - naming files for interactive parallel jobs 359
  - naming serial and batch files 358
  - planning considerations 359
  - system-initiated 86, 356
  - user-initiated 86, 356
- checkpointing a job step
  - llckpt 142
- checkpointing job state 134
- choice button 296
- circular preemption 392
- ckpt (subroutine) 218
- ckpt\_dir 87, 320
- ckpt\_file 87
- ckpt\_time\_limit 88, 320, 326
- class
  - job command file keyword 88
  - multiple job classes 446
  - querying class information 144
- Class
  - defining for a machine 339
  - keyword 339
- class\_comment 320
- class stanza keywords
  - admin 320
  - ckpt\_dir 320
  - ckpt\_time\_limit 320
  - class\_comment 320
  - core\_limit 327
  - cpu\_limit 327
  - data\_limit 327
  - default\_resources 320
  - exclude\_groups 321
  - exclude\_users 321
  - execution\_factor 321
  - file\_limit 327
  - include\_groups 322
  - include\_users 322
  - master\_node\_requirement 322
  - max\_node 322
  - max\_processors 322
  - max\_total\_tasks 322
  - maxjobs 322
  - nice 322
  - NQS\_class 323
  - NQS\_query 323
  - NQS\_submit 323
  - priority 323
  - rss\_limit 328
  - stack\_limit 328
  - total\_tasks 323
  - wall\_clock\_limit 328
- class stanzas
  - examples 328
  - format 319
- ClassSysprio 343
- CLIENT\_TIMEOUT 356
- cluster
  - definition 3
  - querying multiple clusters 43

- cluster *(continued)*
  - submitting jobs to multiple clusters 43
- CM\_COLLECTOR\_PORT 356
- coexistence
  - LoadLeveler 2.2 and 3.1 xix
  - mixed cluster xix
  - POE in a mixed cluster xx
  - requirements, restrictions, and operating characteristics xx
- collect account data
  - GUI 31
- COLLECTOR\_DGRAM\_PORT 356
- command
  - details 137
- command line interface 19
  - 64-bit support 397
  - llclass -l 397
  - llq -l 398
  - llstatus -l 397
- commands
  - Gang scheduling 394
  - llacctmrg 138
  - llcancel 140
  - llckpt 142
  - llclass 144
  - llctl 148
  - lldcegrpmaint 153
  - llextSDR 155
  - llfavorjob 159
  - llfavoruser 160
  - llhold 161
  - llinit 163
  - llmatrix 165
  - llmodify 168
  - llpreempt 170
  - llprio 171
  - llq 173
  - llstatus 191
  - llsubmit 200
  - llsummary 202
  - summary 20
- comment 88
- common name space 59
- communication, hierarchical 383
- communication level 96
- complete pending job state 134
- completed job state 134
- concepts, Gang 381
- configuration file
  - customizing 64, 333
  - keywords 115
  - multiple statements 393
  - structure and syntax 64
- configuration file keyword details 386
- configuration file keyword summary 385
- configuration file keywords 385
- configuration files
  - global and local 64
- configuration tasks
  - GUI 30
- configuration wizard
  - lltg 30
- configuring
  - DCE 365
- configuring LoadLeveler
  - global configuration file 64
  - introduction 63
  - LoadLeveler user ID 64
  - local configuration file 64
- considerations
  - checkpointing 359
  - parallel jobs 71
  - POE 71
  - POE in a mixed cluster xx
  - PVM 72
- consumable CPUs, modify 306
- consumable memory, modify 306
- consumable resource enforcement 392
- consumable resources
  - introduction 14
  - job scheduling 14
  - Workload Manager 15
- Consumable Resources 341
  - when submitting and managing jobs 47
- ConsumableCpus
  - variable 66
- ConsumableMemory
  - variable 66
- ConsumableVirtualMemory
  - variable 66
- contents, appendix 403
- CONTINUE expression 348
- control functions 347
- copy 326
- COPY\_ALL 91
- core\_limit 88, 320, 327
- cpu\_limit 89, 320, 327
- cpu\_speed\_scale 311, 313, 375
- Cpus
  - using with MACHPRIO 345
  - variable 66
- CPUs, modify consumable 306
- create account report
  - GUI 32
- css\_type 332
- CurrentTime 66
- CUSTOM\_METRIC 334
- CUSTOM\_METRIC\_COMMAND 334
- customizing
  - administration file 310
  - configuration file 64
  - global and local configuration file 333
- CustomMetric 66, 345

## D

- daemons 129
  - gsmonitor 133
  - kbdd 133
  - master 129
  - negotiator 133

- daemons (*continued*)
  - schedd 129
  - startd 130
- daemons, LoadLeveler 8
- data access
  - API 223
- data\_limit 89, 320, 327
- DCE
  - authentication process 284
  - authentication programs 283, 284
  - handling security credentials 283
- DCE (Distributed Computing Environment) 365
- DCE\_ADMIN\_GROUP 365
- DCE Authentication 371
- DCE\_AUTHENTICATION\_PAIR 365, 371
- DCE\_ENABLEMENT 365
- dce groups
  - generating 153
  - maintaining 153
- dce\_host\_name 311, 313
- DCE\_SERVICES\_GROUP 365
- debugging
  - controlling output 353
- dedicated adapters 96
- default\_class 316
- default\_group 316, 317
- default\_interactive\_class 316, 317
- default\_resources 320
- default scheduler 336
- deferred job state 134
- definitions, scheduler keywords 335
- dependency 89, 445
- details
  - administration file keyword 390
  - API scheduler 337
  - backfill scheduler 386
  - configuration file keyword 386
  - default scheduler 336
  - execution\_factor 390
  - GANG\_MATRIX\_BROADCAST\_CYCLE 386
  - GANG\_MATRIX\_NODE\_SUBSET\_SIZE 386
  - GANG\_MATRIX\_REORG\_CYCLE 386
  - GANG\_MATRIX\_TIME\_SLICE 386
  - gang scheduler 337
  - HIERARCHICAL\_FANOUT 386
  - max\_smp\_tasks 390
  - max\_total\_tasks 390
  - PREEMPT\_CLASS 387
  - scheduler options 336
  - START\_CLASS 388
- details, command 137
- diagnosing problems 435
- directories
  - naming for checkpointing 358
- Disk
  - requirement in job command file 101
  - using with MACHPRIO 345
  - variable 66
- displaying job status
  - using the command llq 405
  - using the GUI 303

- displaying machine status
  - details 307
  - floating resources 307
  - machine resources 307
  - public submit machines 308
  - scheduler in use 308
  - using llstatus 406
  - using the GUI 306
- Distributed Computing Environment (DCE) 365
- domain 66
- drain
  - GUI 30
- DRAIN\_ON\_SWITCH\_TABLE\_ERROR 371
- dsh command (in PSSP) 443

## E

- editing jobs 43, 301
- ENFORCE\_RESOURCE\_SUBMISSION 342
- ENFORCE\_RESOURCE\_USAGE 342
- EnteredCurrentState 66
- environment 91
- environment variable
  - MALLOCTYPE 61, 150, 273
- environment variables 46
- epilog programs 286
- error job command file keyword 91
- exclude\_groups 320, 321
- exclude\_users 320, 321, 330
- executable 41, 92
  - job command file 409
  - specified in a job command file 39
- EXECUTE 352
- executing machine 7
- execution\_factor 320, 321, 390
- execution window for jobs 444
- exit status 99, 200
  - prolog program 289
- expressions
  - CONTINUE 347
  - KILL 347
  - START 347
  - SUSPEND 347
  - VACATE 347
- extended accounting report 77
- external scheduler 270, 335

## F

- favor jobs 28
  - llfavorjob command 159
- favor users 27
  - llfavoruser command 160
- feature
  - requirement in job command file 101
- Feature
  - configuration file keyword 341
- file
  - customizing administration file 310
  - customizing configuration file 64
- file\_limit 92, 320, 327



- file structure and syntax
  - administration file 62
- file system monitoring 337
  - FS\_INTERVAL 337
  - FS\_NOTIFY 338
  - FS\_SUSPEND 338
  - FS\_TERMINATE 338
- files 358
  - naming checkpoint files 360
  - naming checkpointing files for interactive parallel jobs 359
  - naming checkpointing files for serial and batch jobs 358
- filtering a job script 285
- FLOATING\_RESOURCES 342
- flush
  - GUI 30
- FreeRealMemory 66
- FS\_INTERVAL 337
- FS\_NOTIFY 338
- FS\_SUSPEND 338
- FS\_TERMINATE 338

## G

- Gang matrix
  - query 165
- GANG\_MATRIX\_BROADCAST\_CYCLE 386
- GANG\_MATRIX\_NODE\_SUBSET\_SIZE 386
- GANG\_MATRIX\_REORG\_CYCLE 386
- GANG\_MATRIX\_TIME\_SLICE 386
- Gang scheduler 337
  - checklist 437
- GANG scheduler 335
- Gang scheduling
  - administration file keyword details 390
  - administration file keyword summary 390
  - administration file keywords 389
  - APIs 394
  - application support 384
  - circular preemption 392
  - commands 394
  - concepts 381
  - configuration file keyword details 386
  - configuration file keyword summary 385
  - configuration file keywords 385
  - consumable resource enforcement 392
  - hierarchical communication 383
  - implied START\_CLASS values 393
  - interactions 392
  - job command file 394
  - keywords 385
  - last one wins rule 393
  - NTP 392
  - overview 381
  - preemption 384
  - reconfiguration 392
  - restrictions 392
  - sample administration file 390
  - sample configuration file 388
  - supported hardware 384

- Gang scheduling (*continued*)
  - task switching 384
- GetHistory 78
- GetHistory (subroutine) 217
- global configuration file
  - configuring 64
  - customizing 333
- GLOBAL\_HISTORY 77, 350
- graphical user interface
  - basic functions 23
  - building jobs 24
  - customizing 21, 24
  - overview 21
  - starting 21
  - tasks 293
  - Xloadl 24
  - Xloadl\_so 24
- group 92
  - default 317
  - UNIX 317
- group stanza keywords
  - admin 330
  - exclude\_users 330
  - include\_users 330
  - max\_node 331
  - max\_processors 331
  - max\_total\_tasks 331
  - maxidle 330
  - maxjobs 330
  - maxqueued 330
  - priority 331
  - total\_tasks 331
- group stanzas
  - examples 331
  - format 329
- GroupQueuedJobs 343
- GroupRunningJobs 344
- GroupSysprio 344
- GroupTotalJobs 344
- gsmonitor daemon 133
- GUI
  - 64-bit support 397, 398
  - building jobs 24
  - configuration tasks 30
  - customizing 21, 24
  - machine administration 29
    - capture data 31
    - collect account data 31
    - configuration tasks 30
    - create account report 32
    - drain 30
    - flush 30
    - purge 31
    - reconfig 30
    - recycle 30
    - resume 31
    - start all 29
    - start LoadLeveler 29
    - stop all 30
    - stop LoadLeveler 30
    - version 32

- GUI (*continued*)
  - overview 21
  - starting 21
  - tasks 293
  - Xloadl 24
  - Xloadl\_so 24
- GUI (see graphical user interface) 27

## H

- hardware, supported 384
- help
  - calling IBM 447
  - in the GUI 23
- hierarchical communication 383
- HIERARCHICAL\_FANOUT 386
- hints for running LoadLeveler 443
- HISTORY 352
- history file
  - troubleshooting 447
- HISTORY\_PERMISSION 350, 371
- hold 92
- holding jobs
  - using llhold 44, 406
  - using the GUI 305
- host 67
- hostname 67
- how to checkpoint a job 362

## I

- idle job state 134
- image\_size 93
- implied START\_CLASS values 393
- include\_groups 320, 322
- include\_users 320, 322, 330
- initialdir 93
- initiators 342
- input 94
- integer blocking 50
- integrating LoadLeveler with WLM 68
- interactions
  - Gang scheduling 392
    - circular preemption 392
    - consumable resource enforcement 392
    - implied values 393
    - job command file 394
    - last one wins rule 393
    - NTP 392
    - reconfiguration 392
    - restrictions and preemption 392
- interactive jobs
  - planning considerations 71
- interactive parallel jobs
  - naming files for checkpointing 359
- interface\_address 332
- interface\_address keyword 333
- interface\_name 332
- interface\_name keyword 333
- Interfaces, Application Programming 33, 215

## J

- job
  - accounting 75
  - batch 6
  - building a job command file 39, 293
  - building using the GUI 24
  - cancelling 44, 305
  - change step attributes 168
  - class name 88
  - definition 5
  - diagnosing problems with 435, 436, 438
  - editing 43, 301
  - environment variables 42
  - exit status 99, 200
  - filter 285
  - holding 44, 305
  - interactive 71
  - parallel 49, 436
  - preempt a step 170
  - priority 44, 171, 305, 318, 323, 331
  - releasing a hold 305
  - running 443
  - samples 405
  - serial 39
  - states 13
  - status 43, 173, 176, 303
  - submit-only 438
  - submitting 39, 42, 303
- job accounting setup procedure 78, 374
- JOB\_ACCT\_Q\_POLICY 75
- job command file
  - building 39
  - environment keyword
    - COPY\_ALL 91
  - environment keywords
    - !var 91
    - \$var 91
    - var=value 91
  - example 40, 407, 408
  - executable example 409
  - keywords 85
    - account\_no 85
    - arguments 85
    - blocking 85
    - checkpoint 86
    - ckpt\_dir 87
    - ckpt\_file 87
    - ckpt\_time\_limit 88
    - class 88
    - comment 88
    - core\_limit 88
    - cpu\_limit 89
    - data\_limit 89
    - dependency 89
    - environment 91
    - error 91
    - executable 92
    - file\_limit 92
    - group 92
    - hold 92
    - image\_size 93

job command file *(continued)*

keywords *(continued)*

- initialdir 93
- input 94
- job\_cpu\_limit 94
- job\_name 94
- job\_type 95
- max\_processors 95
- min\_processors 95
- network 96
- node 97
- node\_usage 98
- notification 99
- notify\_user 99
- output 99
- parallel\_path 100
- preferences 100
- queue 100
- requirements 101
- resources 103
- restart 104
- restart\_from\_ckpt 104
- restart\_on\_same\_nodes 105
- rss\_limit 105
- shell 105
- stack\_limit 105
- startdate 106
- step\_name 106
- task\_geometry 106
- tasks\_per\_node 107
- total\_tasks 107
- user\_priority 108
- wall\_clock\_limit 108

parallel 40

serial 40

submitting 42

syntax 39

Job command file

Gang scheduling 394

job\_cpu\_limit 94, 320

JOB\_EPILOG 286

job files

naming for checkpointing 358

naming for checkpointing interactive parallel

jobs 359

JOB\_LIMIT\_POLICY 75

job\_name 94

job object 130, 230

JOB\_PROLOG 286

job queue

definition 7

job routing procedure, NQS 81, 375

job scheduling

consumable resources 14

job state 134

canceled 134

checkpointing 134

complete pending 134

completed 134

deferred 134

idle 134

job state *(continued)*

not run 135

NotQueued 134

pending 135

preempt pending 135

preempted 135

reject pending 135

rejected 135

remove pending 135

removed 135

resume pending 135

running 135

starting 135

system and user hold 136

system hold 135

terminated 135

user hold 136

vacate pending 136

vacated 136

job\_type 95

JOB\_USER\_EPILOG 286

JOB\_USER\_PROLOG 286

## K

kbdd daemon 133

KeyboardIdle 67

keyword

environment

COPY\_ALL 91

keyword, administration file details 390

keyword, administration file summary 390

keyword, configuration file details 386

keyword, configuration file summary 385

keywords

adapter stanza 332

administration file 70, 111, 310, 389

64-bit support 396

checkpoint 357

administration file 357

configuration file 357

job command file 357

class stanza 320

configuration file 64, 70, 115, 370, 385

64-bit support 396

LoadLeveler variables 66, 126

user-defined 125

environment

!var 91

\$var 91

var=value 91

Gang scheduling 385

group stanza 330

job command file 85

64-bit support 395

machine stanza 311

node\_usage 394

reserved 70

SCHEDULER\_API 335

SCHEDULER\_TYPE 335

user stanza 316

KILL expression 348

## L

LAPI 96

last one wins rule 393

LIB 352

libllapi.a 33, 215

limitations

checkpoint and restart 360

PVM 73

limits 323, 326

ll\_ckpt 219

ll\_control (subroutine) 271

ll\_deallocate (subroutine) 252

LL\_DEFAULT scheduler 335

ll\_error 259

ll\_free\_jobs (subroutine) 266

ll\_free\_nodes (subroutine) 268

ll\_free\_objs (subroutine) 251

ll\_get\_data (subroutine) 233

ll\_get\_hostlist (subroutine) 261

ll\_get\_jobs (subroutine) 265

ll\_get\_nodes (subroutine) 267

ll\_get\_objs (subroutine) 228

ll\_init\_ckpt 218

ll\_modify (subroutine) 275

ll\_next\_obj (subroutine) 251

ll\_preempt (subroutine) 277

ll\_query (subroutine) 224

ll\_reset\_request (subroutine) 227

ll\_set\_ckpt\_callbacks 221

ll\_set\_request (subroutine) 224

ll\_start\_host (subroutine) 263

ll\_start\_job (subroutine) 278

ll\_terminate\_job (subroutine) 280

ll\_unset\_ckpt\_callbacks 222

LL\_Version

requirement in job command file 102

llacctmrg 138

llacctlval (user exit) 215

llapi.h 33, 215

llcancel 140

llckpt 142

llclass 144

llctl 148

lldcegrpmaint 153

llxtSDR 155

llfavorjob 159

llfavoruser 160

llfree\_job\_info (subroutine) 269

llhold 161

llinit 163

llmatrix 165

llmodify 168

llpreempt 170

llprio 171

llq 173

llstatus 191

llsubmit (command) 200

llsubmit (subroutine) 268

llsummary 202

lltg, configuration wizard 30

load average 446

LoadAvg

using with MACHPRIO 345

variable 67

LoadL\_admin file 62, 417, 425

LOADL\_ADMIN keyword 334

LOADL\_CONFIG 43

LoadL\_config file 64

LoadL\_config.local file 64, 426, 429

LOADL\_INTERACTIVE\_CLASS 317

LOADL\_PROCESSOR\_LIST 55

loadl user ID 63, 64

LoadLeveler

commands 137

integrating with WLM 68

job states 13

LoadLeveler 2.2 and 3.1, coexistence xix

LoadLeveler APIs 33

64-bit support 399

LoadLeveler basics 4

LoadLeveler daemons 8

LoadLeveler user ID 63

LoadLeveler variables 66

Arch 66

ConsumableCpus 66

ConsumableMemory 66

ConsumableVirtualMemory 66

Cpus 66

CurrentTime 66

CustomMetric 66

Disk 66

domain 66

EnteredCurrentState 66

FreeRealMemory 66

host 67

in a job command file 109

KeyboardIdle 67

LoadAvg 67

Machine 67

MasterMachPriority 67

Memory 67

OpSys 67

PagesFreed 67

PagesScanned 67

QDate 67

Speed 67

state 67

tilde 67

UserPrio 67

VirtualMemory 67

LOCAL\_CONFIG 352

local configuration file

configuring 64

customizing the 333

LOG 352

log files 352

GSMONITOR\_LOG 353

KBDD\_LOG 353

MASTER\_LOG 353

log files (*continued*)

- MAX\_KBDD\_LOG 353
- MAX\_NEGOTIATOR\_LOG 353
- MAX\_STARTER\_LOG 353
- NEGOTIATOR\_LOG 353
- SCHEDD\_LOG 353
- STARTD\_LOG 353
- STARTER\_LOG 353

## M

machine

- administrative actions 29
- GUI 29
- public scheduling 315
- scheduling 7

Machine

- requirement in job command file 102
- variable 67

MACHINE\_AUTHENTICATE 334

machine\_mode 311, 313

machine stanza keywords

- adapter\_stanzas 311
- alias 311
- central\_manager 312
- cpu\_speed\_scale 313, 375
- dce\_host\_name 313
- machine\_mode 313
- master\_node\_exclusive 313
- max\_adapter\_windows 313
- max\_jobs\_scheduled 313
- max\_smp\_tasks 313
- name\_server 314
- pool\_list 314
- pvm\_root 314
- resources 314
- schedd\_fenced 315
- schedd\_host 315
- spacct\_exclude\_enable 315
- speed 315
- submit\_only 315

machine stanzas

- examples 316
- format 310

machine status 191

MACHINE\_UPDATE\_INTERVAL 371, 443

MACHPRIO 345

MAIL keyword 286

mail program 286

MALLOCTYPE 61, 150, 273

manual pages for public code 450

master daemon 129

MASTER\_DGRAM\_PORT 356

master node 74

master\_node\_exclusive 311, 313

master\_node\_requirement 320, 322

MASTER\_STREAM\_PORT 356

MasterMachPriority 345

- variable 67

max\_adapter\_windows 311, 313

MAX\_CKPT\_INTERVAL 362

MAX\_JOB\_REJECT 371

max\_jobs\_scheduled 311, 313

max\_node 316, 318, 320, 322, 330, 331

max\_processors 95, 316, 318, 320, 322, 330, 331

max\_smp\_tasks 311, 313, 390

MAX\_STARTERS 340, 342

- limits set by 343

max\_total\_tasks 316, 318, 320, 322, 330, 331, 390

maxidle 316, 317, 330, 444

maxjobs 316, 317, 320, 322, 330, 444

maxqueued 316, 318, 330, 444

Memory

- requirement in job command file 102

- using with MACHPRIO 345

- variable 67

memory, modify consumable 306

menu bar 21

messages 310

migration considerations xix

MIN\_CKPT\_INTERVAL 362

min\_processors 95

mixed cluster

- coexistence xix

- requirements, restrictions, and operating characteristics xx

- using POE xx

modify consumable CPUs and consumable memory 306

monitor\_program 270

monitoring, file system 337

monitoring programs 270

MPI 96

multilink\_address keyword 333

multilink\_list keyword 333

multiple statements

- administration file 393

- configuration file 393

## N

name\_server 311, 314

naming

- checkpoint files 360

- checkpointing files and directories 358

- checkpointing files for interactive parallel jobs 359

- checkpointing files serial and batch 358

naming for checkpointing 358

NEGOTIATOR\_CYCLE\_DELAY 371

negotiator daemon 133

- job states 13

- keywords 371

NEGOTIATOR\_INTERVAL 371, 443

NEGOTIATOR\_LOADAVG\_INCREMENT 371, 372

NEGOTIATOR\_PARALLEL\_DEFER 372

NEGOTIATOR\_PARALLEL\_HOLD 372

NEGOTIATOR\_RECALCULATE\_SYSPRIO\_INTERVAL 372

NEGOTIATOR\_REJECT\_DEFER 372

NEGOTIATOR\_REMOVE\_COMPLETED 372

NEGOTIATOR\_RESCAN\_QUEUE 372

NEGOTIATOR\_STREAM\_PORT 356

network 96

- Network Time Protocol 392
- network\_type 332
- network\_type keyword 333
- nice 320, 322
- node keyword 50, 97
- node\_usage 98, 394
- not run job state 135
- notification 99
- notify\_user 99
- NotQueued job state 134
- NQS
  - options 377
  - routing jobs to NQS machines 42, 79
  - scripts 80
- NQS\_class 80, 320, 323
- NQS\_DIR 80, 352
- NQS jobs
  - cancelling 379
  - obtaining status 379
  - submitting 376
- NQS machine job routing procedure 81, 375
- NQS\_query 80, 320, 323
- NQS scripts 379
- NQS\_submit 80, 320, 323
- NTP 392

## O

- OBITUARY\_LOG\_LENGTH 373
- obtaining status, parallel jobs 54
- online information xv
- operating characteristics, coexistence xx
- operators 65
- OpSys
  - requirement in job command file 102
  - variable 67
- output 99, 445
  - debugging 353
- overview, Gang 381

## P

- PagesFreed
  - variable 67
- PagesScanned
  - variable 67
- parallel job command files 40
- parallel jobs 73
  - administration 71
  - API 260
  - checklist 437
  - Class keyword 73
  - class stanza 73
  - interactive, naming files for checkpointing 359
  - job command file examples 411
  - master node 74
  - obtaining status 54
  - overview 49
  - scheduling considerations 71
  - supported keywords 71

- parallel jobs, batch
  - naming files for checkpointing 358
- parallel\_path 100
- pending job state 135, 438
- performance 60
- planning
  - checkpointing 359
  - POE 71
  - PVM 72
- POE
  - environment variables 54
  - job command file 411
  - mixed cluster considerations xx
  - planning considerations 71
- POLLING\_FREQUENCY 373
- POLLS\_PER\_UPDATE 373
- Pool
  - requirement in job command file 102
- pool\_list 311, 314
- port numbers 355
- preempt
  - job step 170
- PREEMPT\_CLASS 387
- preempt pending job state 135
- preempted job state 135
- preemption 384
  - resources 385
  - restrictions 392
  - two types 384
- preemption, circular 392
- preferences 100
- priority 44, 316, 320, 330
- priority (of jobs)
  - keyword in class stanza 323
  - keyword in group stanza 331
  - keyword in user stanza 318
  - system priority 44
  - user priority 44, 171, 318
- procedure
  - job accounting setup 78, 374
  - NQS machine job routing 81, 375
- process
  - starter 132
- PROCESS\_TRACKING 364
- PROCESS\_TRACKING\_EXTENSION 364
- productivity aids 443
- Programming Interfaces, Application 33, 215
- prolog programs 286
- Protocol, Network Time 392
- public scheduling machine 315
- public scheduling machines 7, 46, 406
- PUBLISH\_OBITUARIES 373
- pull-down menus 22
  - creating 25
- purge
  - GUI 31
- PVM 96
  - job command file 413
  - planning considerations 72
  - restrictions 73
- pvm\_root 311, 314

## Q

- QDate 67, 344
- Query
  - Gang matrix 165
- query a job
  - llq command 173
  - using the GUI 303
- query API 265
- querying class information
  - llclass command 144
- querying multiple clusters 43
- questions and answers 435
- queue 100
- queue, see job queue 7

## R

- reconfig
  - GUI 30
- reconfiguration 392
- recycle
  - GUI 30
- reject pending job state 135
- rejected job state 135
- release from hold 28
- RELEASEDIR 352
- remove pending job state 135
- removed job state 135
- requirements 101
- requirements, coexistence xx
- resources 311, 314
  - job command file keyword 103
- resources, consumable
  - job scheduling 14
  - Workload Manager 15
- restart 104
  - limitations 360
  - restarting a checkpointed job 438
- restart\_from\_ckpt 104
- restart\_on\_same\_nodes 105
- RESTARTS\_PER\_HOUR 373
- restrictions
  - checkpoint and restart 360
  - checkpointing 359
  - Gang scheduling and preemption 392
  - PVM 73
- restrictions, coexistence xx
- resume
  - GUI 31
- resume pending job state 135
- rlim\_infinity 326
- routing jobs, NQS machines 81, 375
- rss\_limit 105, 320, 328
- rule, last one wins 393
- running job state 135
- running jobs at a specific time of day 444

## S

- SAVELOGS keyword 355
- scaling considerations 443
- schedd
  - troubleshooting 447
- schedd daemon 129, 438
  - recovery 441
- schedd\_fenced 311, 315
- schedd\_host 311, 315, 443
- SCHEDD\_INTERVAL 373
- SCHEDD\_RUNS\_HERE 341
- SCHEDD\_STATUS\_PORT 356
- SCHEDD\_STREAM\_PORT 356
- SCHEDD\_SUBMIT\_AFFINITY 341, 443
- SCHEDULE\_BY\_RESOURCES 342
- scheduler
  - keyword definitions 335
  - option details 336
- SCHEDULER\_API 335
- SCHEDULER\_TYPE 335
- schedulers
  - API 270, 335
  - Backfill 335
  - choosing 335
  - Default 335
  - external 270, 335
  - Gang 335
  - supported keywords 49
- scheduling
  - Gang
    - administration file keywords 389
    - APIs 394
    - circular preemption 392
    - configuration file keywords 385
    - consumable resource enforcement 392
    - implied START\_CLASS values 393
    - Job command file 394
    - reconfiguration 392
    - restrictions 392
  - Gang keywords 385
  - parallel jobs 71
- scheduling, job
  - consumable resources 14
- scheduling interactions, Gang 392
- scheduling machine 7
  - public 315
- script not executing
  - troubleshooting 442
- SDR
  - extracting information from 155
- security
  - configuring DCE 365
- security credentials
  - DCE 283
- serial checkpointing
  - ckpt subroutine 218
- serial job command files 40
- serial jobs
  - naming files for checkpointing 358
- service\_class 96
- service numbers 355



- shell 105, 296
- short report, accounting 77
- signals 261
- spacct\_exclude\_enable 311, 315
- speed 311, 315, 375
- Speed 67, 345
- SPOOL
  - log 352
- stack\_limit 105, 320, 328
- stanzas
  - adapter 332
  - class 319
  - default 63
  - label 63
  - machine 310
  - type 63
  - user 310
- start all
  - GUI 29
- START\_CLASS 388
  - implied values 393
- START\_DAEMONS 341
- START expression 348
- start failure
  - MALLOCTYPE 61, 150, 273
- start LoadLeveler
  - GUI 29
- startd daemon 130, 443
- STARTD\_RUNS\_HERE 341
- STARTD\_STREAM\_PORT 356
- startdate 106
- starter process 132
- starting job state 135
- state 67
- states, job 13, 134
- status 191, 200
- status, obtaining
  - parallel jobs 54
- step\_name 106
- stop all
  - GUI 30
- stop LoadLeveler
  - GUI 30
- striping
  - definition of 52
  - submitting jobs 52
- Striping
  - examples of requesting striping in network statement 54
  - Understanding fabric connectivity 53
- structure
  - administration file 62
- SUBMIT\_FILTER 285
- submit\_only keyword 311, 315
- submit-only machine
  - cancelling jobs from 44
  - definition 3
  - keywords 315
  - master daemon interaction 129
  - querying jobs from 43
  - querying multiple clusters 43
- submit-only machine (*continued*)
  - schedd daemon interaction 129
  - submitting jobs from 42
  - troubleshooting 438
  - types 7
- submitting jobs
  - across multiple clusters 43
  - using a job command file 42
  - using an API 268
  - using llsbmit 405
  - using llsbmit command 200
  - using the GUI 303
- subroutines
  - ckpt 218
  - GetHistory 216
  - ll\_control 271
  - ll\_deallocate 252
  - ll\_free\_jobs 266
  - ll\_free\_nodes 268
  - ll\_free\_objs 251
  - ll\_get\_data 233
  - ll\_get\_hostlist 261
  - ll\_get\_jobs 265
  - ll\_get\_nodes 267
  - ll\_get\_objs 228
  - ll\_modify 275
  - ll\_next\_obj 251
  - ll\_preempt 277
  - ll\_query 224
  - ll\_reset\_request 227
  - ll\_set\_request 224
  - ll\_start\_host 263
  - ll\_start\_job 278
  - ll\_terminate\_job 280
  - llacval (user exit) 215
  - llfree\_job\_info 269
  - llsubmit 268
- summary
  - commands 20
- support, 64-bit keywords 395, 396
- support, application 384
- support services 447
- SUSPEND expression 348
- switch\_node\_number 332
- switch\_node\_number keyword 333
- switching, tasks 384
- syntax
  - administration file 62
- sys/wait.h 289
- syshold 28
- SYSPRIO 44, 343
- system and user hold job state 136
- system hold job state 135
- system-initiated checkpointing 86, 356
- system priority 44

**T**

- take checkpoint 306
- task assignment 50
- task\_geometry 50, 106



- task switching 384
- tasks\_per\_node 107
- tasks\_per\_node keyword 50
- TCP/IP service and port numbers 355
- terminated job state 135
- tilde 67
- Time Protocol, Network 392
- tm\_hour 68
- tm\_isdst 68
- tm\_mday 68
- tm\_min 68
- tm\_mon 68
- tm\_sec 68
- tm\_wday 68
- tm\_yday 68
- tm\_year 68
- tm4\_year 68
- total\_tasks 107, 316, 318, 320, 323, 330, 331
- total\_tasks keyword 50
- troubleshooting 435
  - .login script not executing 442
  - .profile script not executing 442
  - central manager isn't operating 440
  - checkpointed job won't restart 438
  - Gang scheduler checklist 437
  - history file and schedd 447
  - job stays in pending or starting state 438
  - job won't run 435
  - llstatus does not agree with llq 440
  - mksysb created when running jobs 442
  - parallel job won't run 436
  - PVM 437
  - recovering resources 441
  - running jobs when a machine goes down 438
  - set up problems with parallel jobs 437
  - setuid = 0 442
  - submit-only job won't run 438
- TRUNC\_GSMONITOR\_LOG\_ON\_OPEN 353
- TRUNC\_KBDD\_LOG\_ON\_OPEN 353
- TRUNC\_MASTER\_LOG\_ON\_OPEN 353
- TRUNC\_NEGOTIATOR\_LOG\_ON\_OPEN 353
- TRUNC\_SCHEDD\_LOG\_ON\_OPEN 353
- TRUNC\_STARTD\_LOG\_ON\_OPEN 353
- TRUNC\_STARTER\_LOG\_ON\_OPEN 353

## U

- unfavor jobs 28
- unfavor users 28
- UNIX group 317
- unlimited blocking 50, 85
- user-defined variables 65
- user exit
  - llacctval 215
- user exits 282
  - monitoring programs 270
- user hold job state 136
- user-initiated checkpointing 86, 356
- user name 59
- user priority 44
- user\_priority 108

- user stanza keywords 316
  - account 316
  - default\_class 316
  - default\_group 317
  - default\_interactive\_class 317
  - max\_node 318
  - max\_processors 318
  - max\_total\_tasks 318
  - maxidle 317
  - maxjobs 317
  - maxqueued 318
  - total\_tasks 318
- user stanzas
  - examples 319
  - format 310
- UserPrio 67, 344
- UserQueuedJobs 344
- UserRunningJobs 344
- UserSysprio 344
- UserTotalJobs 344

## V

- VACATE expression 348
- vacate pending job state 136
- vacated job state 136
- var=value 91
- variables
  - configuration file
    - user-defined 65
  - LoadLeveler 109
  - user-defined 65, 66
- version
  - GUI 32
- VirtualMemory
  - using with MACHPRIO 345
  - variable 67
- VM\_IMAGE\_ALGORITHM 373

## W

- wall\_clock\_limit 108, 320, 328
- WALLCLOCK\_ENFORCE 373
- WLM
  - consumable resources 15
  - Gang scheduling interactions
    - consumable resource enforcement 392
    - integrating with LoadLeveler 68
- workload manager
  - consumable resources 15
- Workload Manager
  - Gang scheduling interactions
    - consumable resource enforcement 392
    - integrating with LoadLeveler 68
- world wide web information xv

## X

- X\_RUNS\_HERE 341
- xloadl 21
- Xloadl 24



---

# Reader's comments – We'd like to hear from you

IBM LoadLeveler for AIX 5L  
Using and Administering  
Version 3 Release 1

Publication No. SA22-7881-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

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.

---

Name

---

Address

---

Company or Organization

---

Phone No.



Cut or Fold  
Along Line

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

Cut or Fold  
Along Line





Program Number: 5765-E69



Printed in the United States of America  
on recycled paper containing 10%  
recovered post-consumer fiber.

SA22-7881-00

