IBM LoadLeveler for AIX 5L and Linux

Using and Administering

Version 3  Release 2
Note

Before using this information and the product it supports, read the information in “Notices” on page 475.


This edition applies to Version 3 Release 2 of IBM LoadLeveler for AIX 5L and Linux, program numbers 5765-E69 for AIX and 5724-I23 for Linux, and to all subsequent releases and modifications until otherwise indicated in new editions or Technical Newsletters.

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@us.ibm.com

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 2001, 2003, 2004. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
## Contents

About this book .................................................................................................................. xi

Who should use this book .................................................................................................. xiii

Typographic conventions. ................................................................................................... xv

Related information. .............................................................................................................. xvii

Summary of changes .......................................................................................................... xix

High performance switch support ...................................................................................... xix

Multi-adapter connection support. ......................................................................................... xix

Enhancements for running in an RSCT peer domain ........................................................... xix
  New command, llextrpd ..................................................................................................... xix
  The G$monitor daemon . ................................................................................................. xix

Dynamic adapter configuration support. ............................................................................... xx

Automatic resume for drained startd . ................................................................................... xx

Wait option for interactive jobs . ........................................................................................... xx

Enhancements to Ilmmodify .................................................................................................... xx

Enhancements to Data Access API . ....................................................................................... xx

AIX Cluster Security Services . ............................................................................................ xx

Support for Linux . .................................................................................................................. xx

Migration considerations. .................................................................................................... xxi

Moving from 3.1 to 3.2 .......................................................................................................... xxi
  Gang scheduling considerations . ...................................................................................... xxi
  LoadLeveler 3.1 and 3.2 coexistence . .............................................................................. xxi

Part 1. LoadLeveler overview ............................................................................................. 1

Chapter 1. What is LoadLeveler? ......................................................................................... 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

  Chapter 2. What operating systems are supported by LoadLeveler? .......................... 17
    AIX 5L . ........................................................................................................................ 17
    Linux . ........................................................................................................................... 17
    Features not supported in Linux ................................................................................... 17
    Restrictions on LoadLeveler AIX and Linux mixed clusters . ................................ 19

Part 2. LoadLeveler interfaces ........................................................................................... 21

Chapter 3. LoadLeveler command line interface ............................................................. 23

Summary of LoadLeveler commands . ............................................................................... 24

Chapter 4. Using the Graphical User Interface ................................................................. 25

Starting the Graphical User Interface . .............................................................................. 25

© Copyright IBM Corp. 2001, 2003, 2004
## Part 3. User tasks

### Chapter 6. Submitting and managing jobs

- Building a job command file
- Job command file syntax
- Submitting a job command file
- Managing jobs
  - Editing job command files
  - Querying the status of a job
  - Placing and releasing a hold on a job
  - Cancelling a job
  - Checkpointing a job
  - Setting and changing the priority of a job
  - Working with machines
- Run-time environment variables
- Managing jobs that consume resources
  - Specifying the consumption of resources by a job step
  - Displaying currently available resources

### Chapter 7. Special considerations for parallel jobs

- Supported parallel environments
- Keyword considerations for parallel jobs
- Scheduler considerations
- Task assignment considerations.
- Submitting jobs that use striping
  - Understanding striping over multiple networks
  - Understanding striping over a single network
  - Using striping
- Running interactive POE jobs
- Considerations for MPICH and MPICH-GM jobs under Linux
  - Job command file examples
  - Obtaining status of parallel jobs
  - Obtaining allocated host names.

### Part 4. Administrator tasks

### Chapter 8. Administering and configuring LoadLeveler

- Overview
- Planning considerations
  - Where to begin?
- Quick set up
Chapter 9. Administration tasks for parallel jobs ........................................... 79
Scheduling considerations for parallel jobs ................................................. 79
Allowing users to submit interactive POE jobs ........................................... 79
Allowing users to submit PVM jobs ............................................................ 80
Restrictions and limitations for PVM jobs .................................................... 81
Setting up a class for parallel jobs ............................................................... 81
Setting up a parallel master node ............................................................... 82
Configuring LoadLeveler for MPICH-GM ................................................... 82

Chapter 10. Gathering job accounting data .................................................. 83
Collecting job resource data on serial and parallel jobs ............................... 83
Collecting job resource data based on machines ......................................... 83
Collecting job resource data based on events .............................................. 84
Collecting job resource information based on user accounts ....................... 84
Collecting the accounting information and storing it into files ..................... 85
Accounting reports ....................................................................................... 85
Job accounting setup procedure .................................................................. 86

Chapter 11. Routing jobs to NQS machines ................................................ 87
Setting up the NQS environment .................................................................. 88
Designating machines to which jobs will be routed ..................................... 88
NQS scripts .................................................................................................. 88
NQS machine job routing procedure ........................................................... 89

Part 5. Detailed descriptions ................................................................. 91

Chapter 12. Job command file keywords .................................................. 97
account_no ................................................................................................. 97
arguments ................................................................................................. 97
blocking ..................................................................................................... 97
checkpoint ................................................................................................. 98
ckpt_dir ..................................................................................................... 99
ckpt_file .................................................................................................... 99
ckpt_time_limit ........................................................................................ 100
class ........................................................................................................... 100
comment ................................................................................................. 100
core_limit ................................................................................................. 101
cpu_limit .................................................................................................. 101
data_limit ............................................................................................... 101
dependency ............................................................................................... 102
environment ............................................................................................ 103
error .......................................................................................................... 104
executable ................................................................................................. 104
file_limit .................................................................................................. 104
group ......................................................................................................... 105
hold ............................................................................................................ 105
image_size ............................................................................................... 105
initialdir ................................................................................................. 106
input ......................................................................................................... 106
job_cpu_limit .......................................................................................... 106
job_name .................................................................................................. 107
Chapter 13. Administration and Configuration file keywords ........................................ 125
Administration file keywords .................................................................................. 125
Configuration file keywords and LoadLeveler variables ........................................... 130
  Keywords ............................................................................................................... 130
  User-defined keywords ......................................................................................... 143
  LoadLeveler variables ............................................................................................ 144

Chapter 14. LoadLeveler daemons and job states ...................................................... 147
Daemons .................................................................................................................... 147
  The master daemon .................................................................................................. 147
  The schedd daemon .............................................................................................. 148
  The startd daemon .................................................................................................. 149
  The negotiator daemon ........................................................................................... 151
  The kbdd daemon ................................................................................................... 152
  The gsmonitor daemon ......................................................................................... 152
Job states ................................................................................................................... 153

Chapter 15. Commands ............................................................................................... 157
llacctmrg - Collect machine history files .................................................................... 158
llcancel - Cancel a submitted job ............................................................................. 160
llckpt - Checkpoint a running job step ...................................................................... 162
llclass - Query class information ............................................................................ 165
llctl - Control LoadLeveler daemons ....................................................................... 169
lldcgrpmt - LoadLeveler DCE group maintenance utility ......................................... 174
llxtrpd - Extract data from an RSCT peer domain .................................................... 177
llxtsdr - Extract adapter information from the SDR ................................................. 180
llfavorjob - Reorder system queue by job .................................................................. 184
llfavoruser - Reorder system queue by user .............................................................. 186
llhold - Hold or release a submitted job .................................................. 187
llinit - Initialize machines in the LoadLeveler cluster .......................... 189
llmodify - Change attributes of a submitted job step ....................... 191
llpreempt - Preempt a submitted job step ........................................ 194
llprio - Change the user priority of submitted job steps ................. 196
llq - Query job status ................................................................. 198
llstatus - Query machine status ...................................................... 216
llsubmit - Submit a job .................................................................. 226
llsummary - Return job resource information for accounting............. 228

Chapter 16. Application Programming Interfaces (APIs) ......................... 239

Accounting API ................................................................. 239
  - Account validation user exit .............................................. 239
  - Report generation subroutine ........................................ 240
Checkpointing API ........................................................................ 242
  - ckpt subroutine .............................................................. 242
  - ll_init_ckpt ........................................................................ 242
  - llckpt ................................................................................ 242
  - ll_set_ckpt_callbacks .................................................... 246
  - ll_unset_ckpt_callbacks ................................................ 246
Data Access API ........................................................................... 247
  - Using the data access API .................................................. 247
  - ll_query subroutine .......................................................... 248
  - ll_set_request subroutine ................................................ 249
  - ll_reset_request subroutine ............................................. 251
  - ll_get_objs subroutine ...................................................... 252
  - Understanding the LoadLeveler job object model ................. 254
      - ll_get_data subroutine ................................................. 257
      - ll_next_obj subroutine ............................................... 276
      - ll_free_objs subroutine ............................................. 277
      - ll_deallocate subroutine ............................................ 277
  - Examples of using the Data Access API ................................ 278
Error Handling API ........................................................................ 285
  - ll_error subroutine .......................................................... 285
Parallel Job API ............................................................................ 286
  - Interaction between LoadLeveler and the parallel API ............. 286
      - ll_get_hostlist subroutine ........................................... 287
      - ll_start_host subroutine ............................................. 289
  - Examples ............................................................................. 290
Query API .................................................................................... 291
  - ll_get_jobs subroutine .......................................................... 291
  - ll_free_jobs subroutine .......................................................... 292
  - ll_get_nodes subroutine ....................................................... 293
  - ll_free_nodes subroutine ....................................................... 294
Submit API ................................................................................... 295
  - llsubmit subroutine ............................................................ 295
  - llfree_job_info subroutine .................................................. 296
  - Monitoring programs .......................................................... 296
Workload Management API .............................................................. 297
  - ll_control subroutine .......................................................... 298
  - ll_modify subroutine ............................................................ 302
  - ll_preempt subroutine .......................................................... 305
  - ll_start_job subroutine .......................................................... 306
  - ll_start_job_ext subroutine .................................................. 308
  - ll_terminate_job subroutine .............................................. 312
  - Usage notes ........................................................................ 314
User exits ...................................................................................... 315
  - Handling DCE security credentials ...................................... 315
  - Handling an AFS token ....................................................... 317
  - Filtering a job script ............................................................ 318
  - Using your own mail program .............................................. 318
Chapter 17. Procedures .................................................. 325

Using the Graphical User Interface .................................. 325
Step 1: Building jobs ...................................................... 325
Step 2: Edit the job command file .................................... 326
Step 3: Submit a job command file ..................................... 326
Step 4: Display, refresh, and obtain job status ....................... 326
Step 5: Sort the Jobs window ........................................... 327
Step 6: Change priorities of jobs in a queue ......................... 328
Step 7: Hold a job ......................................................... 328
Step 8: Release a hold on a job ........................................ 329
Step 9: Cancel a job ....................................................... 329
Step 10: Modify consumable CPUs, consumable memory, class, and account number ................................. 339
Step 11: Take checkpoint .................................................. 339
Step 12: Display and refresh machine status ......................... 340
Step 13: Sort the Machines window ................................... 341
Step 14: Find the location of the central manager .................... 341
Step 15: Find the location of the public scheduling machines ... 341
Step 16: Find the type of scheduler in use ......................... 342
Step 17: Specify which jobs appear in the Jobs window ............. 342
Step 18: Specify which machines appear in Machines window .... 343
Step 19: Save LoadLeveler messages in a file ....................... 343

Customizing the administration file .................................. 344
Step 1: Specify machine stanzas ....................................... 344
Step 2: Specify user stanzas ............................................. 350
Step 3: Specify class stanzas ............................................ 353
Step 4: Specify group stanzas .......................................... 363
Step 5: Specify adapter stanzas ........................................ 365

Customizing the global and local configuration file ................. 368
Step 1: Define LoadLeveler administrators .......................... 368
Step 2: Define LoadLeveler cluster characteristics ................. 369
Step 3: Define LoadLeveler machine characteristics ............... 374
Step 4: Define consumable resources .................................. 376
Step 5: Specify how many jobs a machine can run ................. 378
Step 6: Prioritize the queue maintained by the negotiator ........ 378
Step 7: Prioritize the order of executing machines maintained by the negotiator .............................................. 380
Step 8: Manage a job’s status using control expressions ............ 383
Step 9: Define job accounting ............................................ 384
Step 10: Specify alternate central managers ........................ 385
Step 11: Specify where files and directories are located ......... 386
Step 12: Record and control log files .................................. 387
Step 13: Define network characteristics .............................. 391
Step 14: Enable checkpointing .......................................... 392
Planning considerations for checkpointing jobs ...................... 395
How to checkpoint a job .................................................. 397
Remove old checkpoint files ............................................ 400
Step 15: Specify process tracking ...................................... 400
Step 16: Configuring LoadLeveler to use a security service ....... 401
Step 17: Specify additional configuration file keywords ............ 410
Setting up job accounting files ......................................... 415
Task 1: Update the configuration file .................................. 415
Task 2: Merge multiple files collected from each machine into one file ......................................................... 415
Task 3: Report job information on all the jobs in the history file ................................. 416
Task 4: Using account numbers and setting up account validation ................................................................. 416
Task 5: Specifying machines and their weights ...................... 417
Routing jobs to NQS machines ........................................... 417
Task 1: Modify the administration file ................................ 418
Task 2: Modify the configuration file .................................. 418
Task 3: Submit the jobs .................................................... 418
Task 4: Obtain status of NQS jobs ..................................... 420
## Chapter 18. Preemption using the Gang scheduler 421

Overview 421
Supported hardware 422
Application support 422
Keywords specific to Gang scheduling 422
- Configuration file keywords for Gang scheduling preemption 422
- Sample configuration file 425
- Administration file keywords for Gang 425
- Sample administration file 426
Gang scheduling interactions and restrictions 428
- Reconfiguration 428
- Circular preemption 428
- Restrictions for Gang scheduling preemption 428
- Implied START_CLASS values 428
- Last one wins rule 429
Job command file and Gang scheduling 429
LoadLeveler commands for Gang 429
APIs used with Gang scheduling 430

## Chapter 19. Support for 64-bit applications 431

64-bit support for Job Command, Configuration, and Administration keywords 431
- 64-bit support for Job Command file keywords 431
- 64-bit support for Administration keywords 432
- 64-bit support for Configuration keywords and expressions 432
64-bit support for Command line interfaces and the GUI 433
- 64-bit support for Command line interfaces 433
- 64-bit support for the GUI 434
64-bit support for the LoadLeveler APIs 434
64-bit support for Accounting functions 434

## Part 6. Appendixes 437

### Appendix A. Examples 439

User tasks: building job command files 439
- Using commands 439
- Additional examples of building job command files 441
User tasks: building parallel job command files 445
- POE 445
- PVM 3.3 (non-SP) 447
- PVM 3.3.11+ (SP2MPI architecture) 447
- MPICH 450
- MPICH-GM 452
Administrator tasks: using an external scheduler 453
- Getting information 453
- Assign resources and dispatch 457

### Appendix B. Troubleshooting 461

Troubleshooting LoadLeveler 461
- Frequently Asked Questions 461
- Helpful hints 470
- Getting help from IBM 474

### Notices 475

Trademarks 477

### Glossary 479
## Bibliography

- Information formats 481
- Finding documentation on the World Wide Web 481
- System planning publications 481
- RS/6000 SP Switch Router publications 482
- Related hardware publications 482
- Related software publications 482
- AIX publications 482
- Reliable Scalable Cluster Technology (RSCT) publications 483
- Cluster Systems Management (CSM) publications 483
- DCE publications 483
- Redbooks 483
- Non-IBM publications 483

## Index

- Index 485
About this book

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 21** 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 43** describes how to create and submit serial and parallel job command files.

- **Part 4, “Administrator tasks,” on page 65** describes how to perform administration tasks, such as configuring LoadLeveler, gathering accounting data, and routing jobs to NQS.

- **Part 5, “Detailed descriptions,” on page 91** provides reference information for keywords, daemons, commands, APIs, and procedures.

- The appendices contains sections with examples, case studies, and information on troubleshooting LoadLeveler.

References within this book to RS/6000® SP™ or SP should be read to also include currently supported @server Cluster 1600 hardware.

A glossary and index are also included.

Who should use this book

This book 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® commands. Administrators should be familiar with system management techniques such as SMIT, as it is used in the AIX® environment. Knowledge of networking and NFS or AFS® protocols is helpful.
# Typographic conventions

This book uses the following typographic conventions:

<table>
<thead>
<tr>
<th>Typographic</th>
<th>Usage</th>
</tr>
</thead>
</table>
| **Bold**     | • **Bold** words or characters represent system elements that you must use literally, such as commands, flags, and path names.  
• **Bold** words also indicate the first use of a term included in the glossary. |
| **Italic**   | • *Italic* words or characters represent variable values that you must supply.  
• *Italics* are also used for book titles and for general emphasis in text. |
| **Constant width** | Examples and information that the system displays appear in constant width typeface. |
| [ ]          | Brackets enclose optional items in format and syntax descriptions. |
| { }          | Braces enclose a list from which you must choose an item in format and syntax descriptions. |
| | A vertical bar separates items in a list of choices. (In other words, it means “or.”) |
| < >          | Angle brackets (less-than and greater-than) enclose the name of a key on the keyboard. For example, `<Enter>` refers to the key on your terminal or workstation that is labeled with the word Enter. |
| ...          | An ellipsis indicates that you can repeat the preceding item one or more times. |
| `<Ctrl-x>`   | The notation `<Ctrl-x>` indicates a control character sequence. For example, `<Ctrl-c>` means that you hold down the control key while pressing `<c>`. |
| \           | The continuation character is used in coding examples in this book for formatting purposes. |
Related information

In addition to this publication, the following books are also part of the LoadLeveler library:
- *Diagnosis and Messages Guide*, GA22-7882
- *Installation Memo*, GI11-2819

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`.

<table>
<thead>
<tr>
<th>Man Page Name</th>
<th>What it Describes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoadL_acctapi</td>
<td>Accounting API</td>
</tr>
<tr>
<td>LoadL_ckptapi</td>
<td>Checkpointing API</td>
</tr>
<tr>
<td>LoadL_dataapi</td>
<td>Data Access API</td>
</tr>
<tr>
<td>LoadL_jobctlapi</td>
<td>Workload Management API</td>
</tr>
<tr>
<td>LoadL_parallelapi</td>
<td>Parallel job API</td>
</tr>
<tr>
<td>LoadL_queryapi</td>
<td>Query API</td>
</tr>
<tr>
<td>LoadL_submitapi</td>
<td>Submit API</td>
</tr>
<tr>
<td>LoadL_errorapi</td>
<td>Error handling API</td>
</tr>
</tbody>
</table>
Summary of changes

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

**High performance switch support**

LoadLeveler now supports the IBM® eserver pSeries® High Performance Switch (pSeries HPS). The pSeries HPS provides significantly greater communication bandwidth along with reduced latency and improved fault tolerance. The HPS can support more than one adapter connected to a network.

**Multi-adapter connection support**

In an environment where a machine has more than one adapter connected to a network, LoadLeveler will distribute the adapter usage across all of the available adapters. This prevents the resources on one adapter from being totally consumed before the next adapter is considered. For more information see “network” on page 109 and “Understanding striping over multiple networks” on page 60.

A new API interface, ll_start_job_ext, is added for external schedulers. This allows external control of the specific adapters and their resources such as window number and adapter memory that are used by job steps. For more information see “ll_start_job_ext subroutine” on page 308.

**Enhancements for running in an RSCT peer domain**

When running LoadLeveler without PSSP, services formerly provided to LoadLeveler by PSSP can now be provided by IBM’s Reliable Scalable Cluster Technology (RSCT). Refer to IBM Reliable Scalable Cluster Technology for AIX 5L™ Guide and Reference, SA22-7889-01 for further details.

**New command, llextRPD**

Similar to llextSDR command, a new command is added to help you set up your administration files. The llextRPD command extracts machine and adapter data from the RSCT peer domain and formats it into stanzas for use in building administration files. For more information on the llextRPD command see “llextRPD - Extract data from an RSCT peer domain” on page 177.

**The GSmonitor daemon**

The GSmonitor daemon can now operate when LoadLeveler is configured to run on RSCT peer domain clusters in addition to the existing clusters running PSSP software. Refer to “The gsmonitor daemon” on page 152 for further details.

**Dynamic adapter configuration support**

LoadLeveler can now be configured to dynamically determine adapter characteristics, including the Switch Network Interface for HPS. This allows the administrator to omit adapter information from the LoadLeveler administration files. Any changes which occur in the adapter configurations on OSIs in the LoadLeveler cluster will be recognized by LoadLeveler without altering the
LoadLeveler administration file and without reconfiguring LoadLeveler. Refer to “Planning considerations” on page 67 for more information.

Automatic resume for drained startd

A new configuration keyword, RESUME_ON_SWITCH_TABLE_ERROR_CLEAR is provided to instruct LoadLeveler to automatically resume the startd if it is in the drained state due to switch table unload errors. When so configured, the startd will automatically resume once the unload errors have cleared. Refer to page 414 for more information.

Wait option for interactive jobs

Support is added to LoadLeveler for a new POE option to queue interactive jobs. This allows interactive POE jobs to be queued in LoadLeveler instead of being terminated right away if resources to run the job are not immediately available.

Enhancements to llmodify

New options in the llmodify command allow the Job class and account number to be changed. A LoadLeveler administrator can also now extend the job step wall clock limits. Refer to “llmodify - Change attributes of a submitted job step” on page 191 for further details.

Enhancements to Data Access API

Job class information is now available through the LoadLeveler Data Access API. This new option will return the same job class information that is currently provided by the llclass command.

AIX Cluster Security Services

LoadLeveler now supports an additional security mechanism. AIX Cluster Security Services (CtSec) allows for identity authentication of LoadLeveler daemons and users. These identities are then used to determine authorization. This prevents unauthorized users and programs from misusing resources or disrupting services. Like DCE, CtSec provides enhanced security. For more information on security and CtSec, see “Configuring LoadLeveler to use Cluster Security Services” on page 407.

Support for Linux

LoadLeveler now supports Red Hat Enterprise Linux 3.0 on xSeries® and SUSE LINUX Enterprise Server 8 on pSeries. Linux machines and AIX 5L machines can now exist in the same LoadLeveler cluster. See “Linux” on page 17 and “Restrictions on LoadLeveler AIX and Linux mixed clusters” on page 19 for more information.
Migration considerations

The LoadLeveler Installation Memo has additional information about and procedures for migration. The information contained there applies only to AIX since there have been no previous Linux releases.

Moving from 3.1 to 3.2

Gang scheduling considerations

When the Gang scheduler option is used you cannot run a mixed cluster where some machines are running LoadLeveler 3.1 and others are running LoadLeveler 3.2. Installations running the Gang scheduler in LoadLeveler 3.1 must move all the nodes in the cluster to LoadLeveler 3.2 at the same time.

LoadLeveler 3.1 and 3.2 coexistence

LoadLeveler 3.2 can coexist with LoadLeveler 3.1. It is possible to operate a "mixed" LoadLeveler cluster consisting of some machines running the new 3.2 software while the remaining machines continue to run LoadLeveler 3.1. The operation of this mixed cluster is subject to a number of restrictions. Most of these restrictions exist because LoadLeveler 3.2 has new features that are not supported by LoadLeveler 3.1. Other restrictions are due to the incompatibilities of the Parallel Environment software levels that support LoadLeveler 3.2 and LoadLeveler 3.1.

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

- The Central Manager must run on a LoadLeveler 3.2 machine.
- 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 3.2.
- No special action is required by LoadLeveler administrators to provide support for serial jobs. Jobs submitted from a 3.1 machine can be executed on a 3.2 machine and jobs submitted from a 3.2 machine can run on a 3.1 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 4.1 supports only LoadLeveler 3.2, and PE 3.2 supports only LoadLeveler 3.1. Therefore, a POE parallel job can not run some of its tasks on LoadLeveler 3.1 machines and the remaining tasks on LoadLeveler 3.2 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 >= "3.2") && (OpSys == "AIX52")" is included in the job command file, LoadLeveler's Central Manager will select only 3.2.0.0 or higher machines with the appropriate OpSys level for this job step. The requirements expression should contain the OpSys specification because the llsSubmit 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 == "AIX52")"

   is specified and "hostname1" and "hostname2" run the same levels of PE and LoadLeveler software.

3. If the mixed cluster has been partitioned into 3.1 and 3.2 LoadLeveler pools then requirement specifications such as
   
   "# @ requirements = (Pool == 31) &&
   (OpSys == "AIX51")" or "# @ requirements = (Pool == 32) && (OpSys ==
   "AIX52")"

   can be used to select machines running the same levels of software. Here, it is assumed that all the 3.1 machines in this mixed cluster
   are assigned to pool 31 and all 3.2 machines are assigned to pool 32. A LoadLeveler administrator can use the "pool_list" keyword of the machine
   stanza of the LoadLeveler administration file to assign machines to pools.

   If a statement such as "# @ executable = /bin/poe" is specified in a job
   command file, and if the job is intended to be run on 3.2 machines, then it is
   important that the job be submitted from a 3.2 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

Chapter 1. What is LoadLeveler? ........................................ 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
  What is LoadLeveler? .................................................. 17
  AIX 5L ................................................................. 17
  Linux ..................................................................... 17
  Features not supported in Linux .................................. 17
  Restrictions on LoadLeveler AIX and Linux mixed clusters ... 19
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).

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 3, “LoadLeveler command line interface,” on page 23

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 4, “Using the Graphical User Interface,” on page 25

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 5, “LoadLeveler API interface,” on page 39

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 6, “Submitting and managing jobs,” on page 45
  - Chapter 7, “Special considerations for parallel jobs,” on page 55
- For detailed information on administrator tasks see:
  - Chapter 8, “Administering and configuring LoadLeveler,” on page 67
  - Chapter 9, “Administration tasks for parallel jobs,” on page 79
  - Chapter 10, “Gathering job accounting data,” on page 83
  - Chapter 11, “Routing jobs to NQS machines,” on page 87

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 6, “Submitting and managing jobs,” on page 45
For more information on keywords used in job command files see:

- “Building a job command file” on page 45
- “Keyword considerations for parallel jobs” on page 55
- Chapter 12, “Job command file keywords,” on page 97
- Chapter 13, “Administration and Configuration file keywords,” on page 125

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

- “Submitting a job command file” on page 48
- “Managing jobs” on page 48
- Chapter 15, “Commands,” on page 157

For more information on daemons see:

- “LoadLeveler daemons” on page 8
- Chapter 14, “LoadLeveler daemons and job states,” on page 147

For more information on job classes see:

- “How LoadLeveler schedules jobs” on page 7
- “Administration file structure and syntax” on page 70
- “Setting up a class for parallel jobs” on page 81

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 55
- “Scheduling considerations for parallel jobs” on page 79
- “Choosing a scheduler” on page 370

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 54
- Chapter 10, “Gathering job accounting data,” on page 83

---

**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
How LoadLeveler works

You can use LoadLeveler to submit jobs which are made up of one or more job steps, where each job step depends upon the completion status of a previous job step. For example, Figure 2 illustrates a stream of job steps:

![Diagram of 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:

```plaintext
job_type = PVM3
job_type = parallel
```

With a job_type of PVM3, 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.

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...
How LoadLeveler works

(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 cluster—until it is ready to be dispatched to run on another machine, as shown in Figure 3 on page 8.
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 353.

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 344.

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 51.

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

<table>
<thead>
<tr>
<th>Daemon</th>
<th>Description</th>
<th>More information on page:</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoadL_master</td>
<td>Referred to as the master daemon. Runs on all machines in the LoadLeveler cluster and manages other daemons.</td>
<td>&quot;The master daemon&quot; on page 147</td>
</tr>
<tr>
<td>LoadL_schedd</td>
<td>Referred to as the schedd daemon. Receives jobs from the llsubmit command and manages them on machines selected by the negotiator daemon.</td>
<td>&quot;The schedd daemon&quot; on page 148</td>
</tr>
<tr>
<td>LoadL_startd</td>
<td>Referred to as the startd daemon. Monitors job and machine resources on local machines and forwards information to the negotiator daemon. <strong>Note:</strong> The startd daemon spawns the starter process (LoadL_starter) which manages running jobs on the executing machine. For more information see &quot;The starter process&quot; on page 151.</td>
<td>&quot;The startd daemon&quot; on page 149</td>
</tr>
<tr>
<td>LoadL_negotiator</td>
<td>Referred to as the negotiator daemon. Monitors the status of each job and machine in the cluster. Responds to queries from llstatus and llq commands. Runs on the central manager machine.</td>
<td>&quot;The negotiator daemon&quot; on page 151</td>
</tr>
<tr>
<td>LoadL_kbdd</td>
<td>Referred to as the keyboard daemon. Monitors keyboard and mouse activity.</td>
<td>&quot;The kbdd daemon&quot; on page 152</td>
</tr>
<tr>
<td>LoadL_GSmonitor</td>
<td>Referred to as the gsmonitor daemon. Monitors for down machines based on the heartbeat responses of the MACHINE_UPDATE_INTERVAL time period.</td>
<td>&quot;The gsmonitor daemon&quot; on page 152</td>
</tr>
</tbody>
</table>

The LoadLeveler job cycle

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

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:

1. Arrow 1 indicates that a job has been submitted to LoadLeveler.
2. 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.
3. 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.
4. 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:
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:

![Diagram showing the process of preparing to run a job]

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:

![Diagram showing the process of initiating a job]

Figure 8. LoadLeveler starts the job

The arrows in Figure 8 illustrate the following:
• 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:

![Diagram](image)

*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 153. For more information about the daemons controlling these job states see “Daemons” on page 147.

<table>
<thead>
<tr>
<th>Job state</th>
<th>Abbreviation</th>
<th>Details on page:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canceled</td>
<td>CA</td>
<td>153</td>
</tr>
<tr>
<td>Checkpointing</td>
<td>CK</td>
<td>153</td>
</tr>
<tr>
<td>Completed</td>
<td>C</td>
<td>153</td>
</tr>
<tr>
<td>Complete Pending</td>
<td>CP</td>
<td>153</td>
</tr>
</tbody>
</table>
### How LoadLeveler works

#### Table 2. Job states (continued)

<table>
<thead>
<tr>
<th>Job state</th>
<th>Abbreviation</th>
<th>Details on page:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deferred</td>
<td>D</td>
<td>153</td>
</tr>
<tr>
<td>Idle</td>
<td>I</td>
<td>153</td>
</tr>
<tr>
<td>Not Queued</td>
<td>NQ</td>
<td>153</td>
</tr>
<tr>
<td>Not Run</td>
<td>NR</td>
<td>153</td>
</tr>
<tr>
<td>Pending</td>
<td>P</td>
<td>153</td>
</tr>
<tr>
<td>Preempted</td>
<td>E</td>
<td>154</td>
</tr>
<tr>
<td>Preempt Pending</td>
<td>EP</td>
<td>154</td>
</tr>
<tr>
<td>Rejected</td>
<td>X</td>
<td>154</td>
</tr>
<tr>
<td>Reject Pending</td>
<td>XP</td>
<td>154</td>
</tr>
<tr>
<td>Removed</td>
<td>RM</td>
<td>154</td>
</tr>
<tr>
<td>Remove Pending</td>
<td>RP</td>
<td>154</td>
</tr>
<tr>
<td>Resume Pending</td>
<td>MP</td>
<td>154</td>
</tr>
<tr>
<td>Running</td>
<td>R</td>
<td>154</td>
</tr>
<tr>
<td>Starting</td>
<td>ST</td>
<td>154</td>
</tr>
<tr>
<td>System Hold</td>
<td>S</td>
<td>154</td>
</tr>
<tr>
<td>User &amp; System Hold</td>
<td>HS</td>
<td>154</td>
</tr>
<tr>
<td>Terminated</td>
<td>TX</td>
<td>154</td>
</tr>
<tr>
<td>User Hold</td>
<td>H</td>
<td>155</td>
</tr>
<tr>
<td>Vacated</td>
<td>V</td>
<td>155</td>
</tr>
<tr>
<td>Vacate Pending</td>
<td>VP</td>
<td>155</td>
</tr>
</tbody>
</table>

**Note:** 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
How LoadLeveler works

- 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 318.

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 a job is preempted, the job continues to use all consumable resources except for ConsumableCpus and ConsumableMemory (real memory) which are made available to other jobs.

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

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

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

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

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

For more information on integrating LoadLeveler with AIX Workload Manager, see “Considerations for integrating LoadLeveler with AIX Workload Manager” on page 77.
Chapter 2. What operating systems are supported by LoadLeveler?

Currently, LoadLeveler supports two operating systems:

- AIX 5L
- Linux

AIX 5L

IBM’s AIX 5L is an open UNIX operating environment that conforms to The Open Group UNIX 98 Base Brand industry standard. It provides high levels of integration, flexibility, and reliability and operates on IBM eServer™ pSeries, IBM eServer Cluster 1600, and IBM RS/6000 servers and workstations.

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

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

Linux

LoadLeveler supports the following distributions of Linux:

- Red Hat Enterprise Linux 3.0 AS running on IBM eServer xSeries machines
- Red Hat Enterprise Linux 3.0 ES running on IBM eServer xSeries machines
- Red Hat Enterprise Linux 3.0 WS running on IBM eServer xSeries machines
- SUSE Linux Enterprise Server 8 running on IBM eServer pSeries machines

LoadLeveler 3.2 for Linux is compatible with LoadLeveler 3.2 for AIX. Its command line interfaces, graphical user interfaces, and application programming interfaces (APIs) are the same as they have been for AIX. The formats of the job command file, configuration file, and administration file also remain the same.

System administrators can set up and maintain a LoadLeveler cluster consisting of some machines running LoadLeveler for AIX and some machines running LoadLeveler for Linux. This is called a mixed cluster. In this mixed cluster jobs can be submitted from either AIX or Linux machines. Jobs submitted to a Linux job queue can be dispatched to an AIX machine for execution, and jobs submitted to an AIX job queue can be dispatched to a Linux machine for execution.

Features not supported in Linux

LoadLeveler 3.2 for Linux supports a subset of the features that are available in the LoadLeveler 3.2 for AIX product. The following features are not supported:

- Checkpoint/restart

LoadLeveler for AIX uses a number of features that are specific to the AIX kernel to provide support for checkpoint/restart of user applications running under LoadLeveler. Checkpoint/restart is not available in this release of LoadLeveler for Linux.
- Process tracking

On AIX, the process tracking feature is implemented as a kernel extension. Using this feature, LoadLeveler for AIX is able to ensure that when a job managed by LoadLeveler has terminated no processes or threads associated with this job are left behind and continue to consume or hold resources. Process tracking is not supported in this release of LoadLeveler for Linux.

- GANG scheduler

The LoadLeveler for AIX GANG scheduler supports Job preemption. GANG scheduling and Job preemption are not supported in this release of LoadLeveler for Linux.

- AIX Workload management (WLM)

WLM can strictly control use of system resources. LoadLeveler for AIX uses WLM to enforce the use of a number of consumable resources defined by LoadLeveler (such as ConsumableCpus and ConsumableMemory). This enforcement of consumable resources usage through WLM is not available in this release of LoadLeveler for Linux.

- CtSec and DCE security

LoadLeveler for AIX can exploit CtSec (Cluster Security Services) or DCE (Distributed Computing Environment) security functions. These functions authenticate the identity of users and programs interacting with LoadLeveler. These features are not available in this release of LoadLeveler for Linux.

- Parallel environment (PE) and Parallel operating environment (POE)

The Parallel Environment for AIX product supports the execution and management of parallel applications. Its major components are:
1. The Parallel Operating Environment (POE) package for submitting and managing jobs
2. A message passing library (MPI) for communication among the tasks that make up a parallel program
3. Parallel Environment (PE) utilities

Support for PE and POE is not available in this release of LoadLeveler for Linux.

- Parallel virtual machine (PVM) and Network queuing system (NQS)

The Parallel Virtual Machine (PVM) package is a software bundle developed at Oak Ridge National Labs to allow a heterogeneous collection of UNIX and Windows® computers to work together as a single large parallel computer. LoadLeveler for AIX is compatible with PVM 3.3. LoadLeveler for AIX is also compatible with Cosmic Network Queuing System (NQS) version 2.0 and NQS systems that have similar interfaces. NQS scripts can be submitted to LoadLeveler for execution. Alternatively, LoadLeveler for can be used to route NQS jobs to an NQS cluster that is external to the LoadLeveler cluster. PVM and NQS are not supported by this release of LoadLeveler for Linux.

- LoadL_GSmonitor daemon

The LoadL_GSmonitor daemon in the LoadLeveler for AIX product uses the Group Services Application Programming Interface (GSAPI) to monitor machine availability and notify the LoadLeveler Central Manager when a machine is no longer reachable. This daemon is not available in the LoadLeveler for Linux product.

- Task guide tool

- Dynamic adapter setup
LoadLeveler for AIX can be configured to dynamically determine adapter characteristics, including those associated with the Switch Network Interface for the IBM eServer pSeries High Performance Switch. This feature is not supported by LoadLeveler for Linux.

- 64-bit applications
  This release of LoadLeveler for Linux supports only 32-bit APIs. On Linux, applications linked to the LoadLeveler APIs must be 32-bit applications.

- System error log
  Each LoadLeveler daemon has its own log file where information relevant to its operation is recorded. In addition to this feature which exists on all platforms, LoadLeveler for AIX also uses the errlog function to record critical LoadLeveler events into the AIX system log. Support for an equivalent Linux function is not available in this release.

### Restrictions on LoadLeveler AIX and Linux mixed clusters

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

- All nodes in a mixed LoadLeveler cluster must run LoadLeveler version 3.2 or higher
- The Central Manager `LoadL_negotiator` daemon must run on an AIX node
- The Central Manager node must run a version of LoadLeveler equal to or higher than any LoadLeveler version being run on a node in the cluster
- DCE or CtSec security features cannot be used
- NQS or PVM cannot be used
- GANG scheduling cannot be used
- AIX jobs that use checkpointing must be sent to AIX nodes for execution. This can be done by either defining and specifying job checkpointing for job classes that exist only on AIX nodes or by coding appropriate requirements expressions. Checkpointing jobs that are sent to a Linux node will be rejected by the `LoadL_startd` daemon running on the Linux node.
- POE jobs must be sent to AIX nodes only.
- WLM and Process tracking are supported in a mixed cluster. However, enforcement of the use of consumable resources will occur through WLM on AIX nodes only. Similarly, the functions associated with Process Tracking are effective only on AIX nodes.
Part 2. LoadLeveler interfaces
Chapter 3. 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 security services are not configured, then administrators are identified by the LOADL_ADMIN keyword in the configuration file. If security services are configured, the configuration file must identify the administrator’s group. Refer to “Step 16: Configuring LoadLeveler to use a security service” on page 401 for more information.

The administrator commands can operate on the entire LoadLeveler job queue and all machines configured. The user commands mainly affect those jobs submitted by that user. Some commands, such as llhold, include options that can only be performed by an administrator.

Table 3 on page 24 summarizes the LoadLeveler commands. Detailed descriptions of these commands including syntax, flags, and usage examples can be found in Chapter 15, “Commands,” on page 157.
## Summary of LoadLeveler commands

*Table 3. LoadLeveler command summary*

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Who Can Issue</th>
<th>For More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>llacctmrg</td>
<td>Collects all individual machine history files together into a single file.</td>
<td>Administrators</td>
<td>See page 158</td>
</tr>
<tr>
<td>llcancel</td>
<td>Cancels a submitted job.</td>
<td>Users and</td>
<td>See page 160</td>
</tr>
<tr>
<td>llclass</td>
<td>Returns information about LoadLeveler classes.</td>
<td>Users and</td>
<td>See page 165</td>
</tr>
<tr>
<td>llctl</td>
<td>Controls daemons on one or more machines in the LoadLeveler cluster.</td>
<td>Administrators</td>
<td>See page 169</td>
</tr>
<tr>
<td>llckpt</td>
<td>Used to checkpoint a single job step.</td>
<td>Users and</td>
<td>See page 162</td>
</tr>
<tr>
<td>llecgrpmaint</td>
<td>Sets up DCE groups and principal names.</td>
<td>DCE Administrators</td>
<td>See page 174</td>
</tr>
<tr>
<td>llextRDP</td>
<td>Extracts data from the RSCT peer domain.</td>
<td>Users and</td>
<td>See page 177</td>
</tr>
<tr>
<td>llextSDR</td>
<td>Extracts adapter information from the system data repository (SDR).</td>
<td>Users and</td>
<td>See page 180</td>
</tr>
<tr>
<td>llfavorjob</td>
<td>Raises one or more jobs to the highest priority, or restores original priority.</td>
<td>Administrators</td>
<td>See page 184</td>
</tr>
<tr>
<td>llfavoruser</td>
<td>Raises job(s) submitted by one or more users to the highest priority, or restores original priority.</td>
<td>Administrators</td>
<td>See page 186</td>
</tr>
<tr>
<td>llhold</td>
<td>Holds or releases a hold on a job.</td>
<td>Users and</td>
<td>See page 187</td>
</tr>
<tr>
<td>llinit</td>
<td>Initializes a new machine as a member of the LoadLeveler cluster.</td>
<td>Administrators</td>
<td>See page 189</td>
</tr>
<tr>
<td>llmodify</td>
<td>Changes attributes or characteristics of a submitted job step.</td>
<td>Users and</td>
<td>See page 191</td>
</tr>
<tr>
<td>llpreempt</td>
<td>Preempts a specified job step.</td>
<td>Administrators</td>
<td>See page 194</td>
</tr>
<tr>
<td>llprio</td>
<td>Changes the user priority of a submitted job step.</td>
<td>Users and</td>
<td>See page 196</td>
</tr>
<tr>
<td>llq</td>
<td>Queries the status of LoadLeveler jobs.</td>
<td>Users and</td>
<td>See page 198</td>
</tr>
<tr>
<td>llstatus</td>
<td>Queries the status of LoadLeveler machines.</td>
<td>Users and</td>
<td>See page 216</td>
</tr>
<tr>
<td>llsubmit</td>
<td>Submits a job.</td>
<td>Users and</td>
<td>See page 226</td>
</tr>
<tr>
<td>llsummary</td>
<td>Returns resource information on completed jobs.</td>
<td>Users and</td>
<td>See page 228</td>
</tr>
</tbody>
</table>
Chapter 4. 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 325.

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 28 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 28 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 26. Each of these sub-windows has its own menu bar.
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.

Figure 10. Main window of the LoadLeveler GUI
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 executables. 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 GPUs. 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:

<table>
<thead>
<tr>
<th>To</th>
<th>Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submit a job</td>
<td>Refer to “Step 3: Submit a job command file” on page 336.</td>
</tr>
<tr>
<td>Cancel a job</td>
<td>Refer to “Step 9: Cancel a job” on page 339.</td>
</tr>
</tbody>
</table>

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?

<table>
<thead>
<tr>
<th>Yes</th>
<th>Submit another job.</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Check the status of the job.</td>
</tr>
</tbody>
</table>

Menu selection conventions
Selections from a menu bar are indicated with an →. For example, if a menu bar included an option called Actions and Actions included an option called Cancel, the instructions would read:
SELECT Actions → 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 325.

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.
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>
<thead>
<tr>
<th>Window</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account Report Data</td>
<td>reporter</td>
</tr>
<tr>
<td>Build a Job</td>
<td>builder</td>
</tr>
<tr>
<td>Checkpoint Fields</td>
<td>ckpt</td>
</tr>
<tr>
<td>Jobs</td>
<td>job_status</td>
</tr>
<tr>
<td>Limits</td>
<td>limits</td>
</tr>
<tr>
<td>Machines</td>
<td>machine_status</td>
</tr>
<tr>
<td>Messages</td>
<td>message_area</td>
</tr>
<tr>
<td>Network</td>
<td>network</td>
</tr>
<tr>
<td>Nodes</td>
<td>nodes</td>
</tr>
<tr>
<td>Preferences</td>
<td>preferences</td>
</tr>
<tr>
<td>PVM</td>
<td>pvm</td>
</tr>
<tr>
<td>Requirements</td>
<td>requirements</td>
</tr>
<tr>
<td>Script</td>
<td>script</td>
</tr>
<tr>
<td>Submit a Job</td>
<td>submit</td>
</tr>
<tr>
<td>Task Geometry</td>
<td>tgeometry</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Resource Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>The background color of the object</td>
</tr>
<tr>
<td>foreground</td>
<td>The foreground color of the object</td>
</tr>
<tr>
<td>geometry</td>
<td>The location of the object</td>
</tr>
<tr>
<td>height</td>
<td>The height of the object</td>
</tr>
<tr>
<td>labelString</td>
<td>The text associated with the object</td>
</tr>
<tr>
<td>width</td>
<td>The width of the object</td>
</tr>
</tbody>
</table>

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.
Customizing the GUI

For more information, refer to the comments in the *Xload* file.

To add a pull-down menu to the Machines window, in the *Xload* 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 *Xload* 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 *Xload* 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 *Xload* file. Look in the *Xload* 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:

<table>
<thead>
<tr>
<th>Resource Variable</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>help*work_area.width</td>
<td>Any integer*</td>
<td>The width of the help panel.</td>
</tr>
<tr>
<td>help*work_area.height</td>
<td>Any integer*</td>
<td>The height of the help panel.</td>
</tr>
<tr>
<td>help*scrollHorizontal</td>
<td>[true</td>
<td>false]</td>
</tr>
<tr>
<td>help*wordWrap</td>
<td>[true</td>
<td>false]</td>
</tr>
</tbody>
</table>

Sets the scrolling option on or off.
Sets word wrapping on or off.

Note:

* 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 368.

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 lctl command.

The main window of the GUI, as shown in Figure 10 on page 26, 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
  - Favor User

<> The Order by User window appears.
Administrative uses of the GUI

Type in
The name of the user whose jobs you want to favor.

Press OK

Unfavor Users
Allows you to unfavor users. This means that you want to unfavor the user’s jobs which you previously favored. This corresponds to the llfavoruser command.

Select Admin from the Jobs window
Select Unfavor User

<> The Order by User window appears.

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 or external schedulers. 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 -r` 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

**Undo Prevent Preempt**
Available only when using the Gang scheduler, Undo Prevent Preempt makes the unpreemptable job preemptable again. This corresponds to the `llmodify -x1` command.

Select One or more jobs from the Jobs window

Select Admin pull-down menu from the Jobs window

Select Undo Prevent Preempt

**Extend Wallclock Limits**
Allows you to extend the wallclock limits by the number of minutes specified. This corresponds to the `llmodify -W` command.

Select Admin pull-down window from the Jobs window

Select Extend Wallclock Limit

<> The Extend Wallclock Limits window appears.

Type in
The number of minutes to extend the wallclock limit.

Press OK

**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.
Administrative uses of the GUI

**Start All**
Starts LoadLeveler on all machines listed in machine stanzas beginning with the central manager. Submit-only machines are skipped. Use this option when specifying alternate central managers in order to ensure the primary central manager starts before any alternate central manager attempts to serve as central manager.

Select Admin from the Machines window.
Select Start All

**Start LoadLeveler**
Allows you to start LoadLeveler on selected machines.

Select One or more machines on which you want to start LoadLeveler.
Select Admin from the Machines window.
Select Start LoadLeveler

**Start Drained**
Allows you to start LoadLeveler with startd drained on selected machines.

Select One or more machines on which you want startd drained.
Select Admin from the Machines window.
Select Start Drained

**Stop LoadLeveler**
Allows you to stop LoadLeveler on selected machines.

Select One or more machines on which you want to stop LoadLeveler.
Select Admin from the Machines window.
Select Stop LoadLeveler.

**Stop All**
Stops LoadLeveler on all machines listed in machine stanzas. Submit-only machines are skipped.

Select Admin from the Machines window.
Select Stop All

**reconfig**
Forces all daemons to reread the configuration files

Select The machine on which you want to operate. To reconfigure this xloadl session, choose reconfig but do not select a machine.
Select Admin from the Machines window.
Select reconfig

**recycle**
Stops all LoadLeveler daemons and restarts them.

Select The machine on which you want to operate.
Select Admin from the Machines window.
Select recycle

**Configuration Tasks**
Starts Configuration Tasks wizard

Select Admin from the Machines window.
Select  Config Tasks

Note: Use the invoking script `lltg` to start the wizard outside of `xloadl`. This option will appear on the pull-down only if the LoadL.tguides 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 system queue to await redispatch. No new jobs are redispatched 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.
Administrative uses of the GUI

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.

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
Administrative uses of the GUI
Chapter 5. 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 16, “Application Programming Interfaces (APIs)”:
- “Accounting API” on page 239
- “Error Handling API” on page 285
- “Checkpointing API” on page 242
- “Submit API” on page 295
- “Data Access API” on page 247
- “Parallel Job API” on page 286
- “Workload Management API” on page 297
- “Query API” on page 291
- “User exits” on page 315

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

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
<th>Who Can Issue?</th>
<th>For More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounting</td>
<td>Generates accounting reports</td>
<td>GetHistory — Users and LoadLeveler Administrators</td>
<td>See page 239</td>
</tr>
<tr>
<td>Checkpointing</td>
<td>Checkpoint LoadLeveler jobs</td>
<td>All APIs (ll_init_ckpt, ll_ckpt, ll_set_ckpt_callbacks, ll_unset_ckpt_callbacks) — Users and LoadLeveler Administrators</td>
<td>See page 242</td>
</tr>
<tr>
<td>Data Access</td>
<td>Accesses LoadLeveler objects and allows you to retrieve data from objects</td>
<td>All subroutines (ll_query, ll_set_request, ll_reset_request, ll_get_objs, ll_get_data, ll_next_obj, ll_free_objs, ll_deallocate) — Users and LoadLeveler Administrators</td>
<td>See page 247</td>
</tr>
<tr>
<td>API</td>
<td>Description</td>
<td>Who Can Issue</td>
<td>For More Information</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td><strong>Error Handling</strong></td>
<td>Converts an error object into an error message</td>
<td><strong>ll_error</strong> — Users and LoadLeveler Administrators</td>
<td>See page 285</td>
</tr>
<tr>
<td><strong>Parallel Job</strong></td>
<td>Tools for parallel job submission</td>
<td>All subroutines (<strong>ll_get_hostlist, ll_start_host</strong>) — Users and LoadLeveler Administrators</td>
<td>See page 286</td>
</tr>
<tr>
<td><strong>Query</strong></td>
<td>Provides information about the jobs and machines in the LoadLeveler cluster</td>
<td>All subroutines (<strong>ll_get_jobs, ll_free_jobs, ll_get_nodes, ll_free_nodes</strong>) — Users and LoadLeveler Administrators</td>
<td>See page 291</td>
</tr>
<tr>
<td><strong>Submit</strong></td>
<td>Submits jobs to LoadLeveler</td>
<td>• <strong>llsubmit</strong> — Users and LoadLeveler Administrators</td>
<td>See page 295</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>llfree_job_info</strong> — Users and LoadLeveler Administrators</td>
<td></td>
</tr>
<tr>
<td><strong>Workload Management</strong></td>
<td>Performs LoadLeveler control operations and provides tools to be used with an external scheduler</td>
<td><strong>ll_control</strong>&lt;br&gt;- LoadLeveler Administrators only:&lt;br&gt;  - LL_CONTROL_DRAIN&lt;br&gt;  - LL_CONTROL_DRAIN_SCHEDD&lt;br&gt;  - LL_CONTROL_DRAIN_STARTD&lt;br&gt;  - LL_CONTROL_FAVOR_JOB&lt;br&gt;  - LL_CONTROL_FAVOR_USER&lt;br&gt;  - LL_CONTROL_FLUSH&lt;br&gt;  - LL_CONTROL_HOLD_SYSTEM&lt;br&gt;  - LL_CONTROL_PURGE_SCHEDD&lt;br&gt;  - LL_CONTROL_RECONFIG&lt;br&gt;  - LL_CONTROL_RECYCLE&lt;br&gt;  - LL_CONTROL_RESUME&lt;br&gt;  - LL_CONTROL_RESUME_SCHEDD&lt;br&gt;  - LL_CONTROL_RESUME_STARTD&lt;br&gt;  - LL_CONTROL_STOP&lt;br&gt;  - LL_CONTROL_SUSPEND&lt;br&gt;  - LL_CONTROL_UNFAVOR_JOB&lt;br&gt;  - LL_CONTROL_UNFAVOR_USER&lt;br&gt;- Users and LoadLeveler Administrators:&lt;br&gt;  - LL_CONTROL_HOLD_USER&lt;br&gt;  - LL_CONTROL_PRIO_ABS&lt;br&gt;  - LL_CONTROL_PRIO_ADJ&lt;br&gt;  - LL_CONTROL_START&lt;br&gt;  - LL_CONTROL_START_DRAINED&lt;br&gt;- <strong>ll_modify</strong> — Users and LoadLeveler Administrators (some functions limited to Gang scheduling)&lt;br&gt;- <strong>ll_preempt</strong> — LoadLeveler Administrators (function limited to Gang and external schedulers)&lt;br&gt;- <strong>ll_start_job</strong> — LoadLeveler Administrators, for use with external schedulers only&lt;br&gt;- <strong>ll_terminate_job</strong> — LoadLeveler Administrators only</td>
<td>See page 297</td>
</tr>
<tr>
<td>API</td>
<td>Description</td>
<td>Who Can Issue?</td>
<td>For More Information</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------------------------------------------</td>
<td>----------------------</td>
</tr>
</tbody>
</table>
| User exits | Allows:  
  • Handling DCE security  
  • Handling AFS tokens  
  • Filtering job scripts  
  • Overriding default mail program  
  • Job prolog  
  • Job epilog | LoadLeveler administrators only  
  **Note:** Other User exits are available with functions limited to specific APIs. | See page 315          |
Part 3. User tasks
Chapter 6. Submitting and managing jobs . . .
Building a job command file . . . . . . . . .
Job command file syntax . . . . . . . . .
Serial job command file . . . . . . . .
Parallel job command file . . . . . . . .
Using multiple steps in a job command file . .
Submitting a job command file . . . . . . . .
Managing jobs . . . . . . . . . . . . .
Editing job command files . . . . . . . .
Querying the status of a job . . . . . . . .
Querying multiple LoadLeveler clusters . . .
Placing and releasing a hold on a job . . . . .
Cancelling a job . . . . . . . . . . . .
Checkpointing a job . . . . . . . . . .
Setting and changing the priority of a job . . .
User priority . . . . . . . . . . . .
System priority . . . . . . . . . . .
How does a job’s priority affect dispatching
order? . . . . . . . . . . . . . .
Working with machines . . . . . . . . .
Run-time environment variables . . . . . . .
Managing jobs that consume resources . . . . .
Specifying the consumption of resources by a job
step . . . . . . . . . . . . . . . .
Displaying currently available resources . . . .

|
|

Chapter 7. Special considerations for parallel
jobs . . . . . . . . . . . . . . .
Supported parallel environments . . . . . .
Keyword considerations for parallel jobs . . .
Scheduler considerations . . . . . . . .
Task assignment considerations . . . . . .
node and total_tasks . . . . . . . .
node and tasks_per_node . . . . . . .
blocking . . . . . . . . . . . .
unlimited blocking . . . . . . . . .
task_geometry . . . . . . . . . .
Submitting jobs that use striping . . . . . .
Understanding striping over multiple networks
Understanding striping over a single network .
Using striping . . . . . . . . . . .
Requesting striping using IP mode . . .
Requesting striping using user space mode .
Requesting striping over a single network .
Requesting striping over multiple networks
and multiple adapters on the same network
Running interactive POE jobs . . . . . . .
Considerations for MPICH and MPICH-GM jobs
under Linux . . . . . . . . . . . . .
Job command file examples . . . . . . . .
Obtaining status of parallel jobs . . . . . .
Obtaining allocated host names . . . . . . .

© Copyright IBM Corp. 2001, 2003, 2004

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

45
45
45
46
46
46
48
48
49
49
49
50
50
50
51
51
51
51
52
53
54
54
54

55
55
55
55
56
56
57
57
57
58
58
60
61
61
62
62
62

. 62
. 62
.
.
.
.

63
63
63
64

43


Chapter 6. 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 7, “Special considerations for parallel jobs,” on page 55.

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

- Using LoadLeveler commands (this chapter)
- Using the LoadLeveler Graphical User Interface
  - For additional information on the LoadLeveler GUI, see Chapter 4, “Using the Graphical User Interface,” on page 25 and “Using the Graphical User Interface” on page 325.
- Using LoadLeveler APIs
  - For additional information on LoadLeveler APIs, see Chapter 5, “LoadLeveler API interface,” on page 39 and Chapter 16, “Application Programming Interfaces (APIs),” on page 239.

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 12, “Job command file keywords,” on page 97.
- 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 123 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.
Building a job command file

- Keyword statements begin with # @. There can be any number of blanks between the # and the @.
- Comments begin with #. Any line whose first non-blank character is a pound sign (#) and is not a LoadLeveler keyword statement is regarded as a comment.
- Statement components are separated by blanks. You can use blanks before or after other delimiters to improve readability but they are not required if another delimiter is used.
- The back-slash (\) is the line continuation character. Note that the continued line must not begin with # @. If your job command file is the script to be executed, you must start the continued line with a #. See Figure 40 on page 443 and Figure 41 on page 445 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.

```bash
# 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
```

**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 7, “Special considerations for parallel jobs,” on page 55](#).

**Using multiple steps in a job command file**

To specify a stream of job steps, you need to list each job step in the job command file. You must specify one `queue` statement for each job step. Also, the executables for all job steps in the job command file must exist when you submit the job. For most keywords, if you specify the keyword in a job step of a multi-step job, its value is inherited by all proceeding job steps. Exceptions to this are noted in the keyword description.

LoadLeveler treats all job steps as independent job steps unless you use the `dependency` keyword. If you use the `dependency` keyword, LoadLeveler determines whether a job step should run based upon the exit status of the previously run job step.
For example, Figure 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.

```sh
# 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
# @ 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:

```sh
# 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 441 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 49 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 ISubmit 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 439.

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 87. For more information on the ISubmit command, see “llsubmit - Submit a job” on page 226.

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 ISubmit command.

To install submit-only LoadLeveler, follow the procedure in the LoadLeveler Installation Memo.

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 49
• “Querying the status of a job” on page 49
• “Placing and releasing a hold on a job” on page 50
• “Cancelling a job” on page 50
• “Checkpointing a job” on page 50
Managing jobs

- “Setting and changing the priority of a job” on page 51
- “Working with machines” on page 52

Editing job command files
After you build a job command file, you can edit it using the editor of your choice. You may want to change the name of the executable or add or delete some statements.

When you create a job command file, it is considered the job executable unless you specify otherwise by using the executable keyword in the job command file. LoadLeveler copies the executable to the spool directory unless the checkpoint keyword was set to yes or interval. Jobs that are to be checkpointed cannot be moved to the spool directory. Do not make any changes to the executable while the job is still in the queue—it could affect the way that job runs.

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 439.

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 198.

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

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

The default name for the master configuration file is /etc/LoadL.cfg. The format for the LOADL_CONFIG environment variable is LOADL_CONFIG=/etc/base_name.cfg or LOADL_CONFIG=base_name. When you
Managing jobs

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

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

For example, you can enter the following query command:

$ llq

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

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

$ env LOADL_CONFIG=othercluster llq

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

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 440 and "Step 7: Release a hold on a job" on page 440.

As a user or an administrator, you can also use thestartdate keyword described in "startdate" on page 120 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 392.
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 `Ilprio` command to change job priorities. See "Ilprio - Change the user priority of submitted job steps" on page 196 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:

```
SYSPRIO : (ClassSysprio * 100) + (UserSysprio * 10) + (GroupSysprio * 1) - (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 378.

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 on page 52 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).
Managing jobs

Table 6. How LoadLeveler handles job priorities

<table>
<thead>
<tr>
<th>Job</th>
<th>User Priority</th>
<th>System Priority (SYSPRIO)</th>
<th>Class</th>
<th>Adjusted System Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rich1</td>
<td>50</td>
<td>10</td>
<td>A</td>
<td>6</td>
</tr>
<tr>
<td>Joe1</td>
<td>10</td>
<td>9</td>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>Joe2</td>
<td>20</td>
<td>8</td>
<td>A</td>
<td>9</td>
</tr>
<tr>
<td>Rich2</td>
<td>100</td>
<td>7</td>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>Rich3</td>
<td>90</td>
<td>6</td>
<td>A</td>
<td>10</td>
</tr>
</tbody>
</table>

Working with machines

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

- 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 440.

- 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 319.

**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 353.

**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:

\[
\text{resources} = \text{ConsumableCpus}(1) \ \text{FRMlicense}(1)
\]

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 56.

Displaying currently available resources

The LoadLeveler user can get information about currently available resources by using the llstatus command with either the -F, or -R options. The -F option displays a list of all of the floating resources associated with the LoadLeveler cluster. The -R option lists all of the consumable resources associated with all of the machines in the LoadLeveler cluster. The user can specify a hostlist with the llstatus command to display only the consumable resources associated with specific hosts.
Chapter 7. 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 "Administration tasks for parallel jobs," on page 79.

Supported parallel environments

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

- AIX 5L
  - IBM Parallel Environment (PE) 4.1
  - Parallel Virtual Machine (PVM) 3.3 (RS/6000 architecture)

- Linux
  - MPICH 1.2.5 (an open-source, portable implementation of the Message-Passing Interface Standard developed by Argonne National Laboratory)
  - MPICH-GM 1.2.5.11 (a port of MPICH on top of Myrinet GM code)

Note: This is the last release that will support the PVM3 job type.

- Parallel Virtual Machine (PVM) 3.3.11+ (SP2MPI architecture)

Linux notes

Support for PE and PVM is not available in this release of LoadLeveler for Linux.

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

<table>
<thead>
<tr>
<th>job_type=parallel</th>
<th>job_type=pvm3</th>
</tr>
</thead>
<tbody>
<tr>
<td>network</td>
<td>Adapter requirement</td>
</tr>
<tr>
<td>node</td>
<td>max_processors</td>
</tr>
<tr>
<td>node_usage</td>
<td>min_processors</td>
</tr>
<tr>
<td>tasks_per_node</td>
<td>network</td>
</tr>
<tr>
<td>total_tasks</td>
<td>network</td>
</tr>
<tr>
<td>task_geometry</td>
<td>parallel_path</td>
</tr>
<tr>
<td>blocking</td>
<td></td>
</tr>
<tr>
<td>All keywords supported for job_type=pvm3 (supported for compatibility reasons)</td>
<td></td>
</tr>
</tbody>
</table>
Keyword considerations for parallel jobs

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

<table>
<thead>
<tr>
<th>job_type=parallel</th>
<th>job_type=pvm3</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_processors</td>
<td>max_processors</td>
</tr>
<tr>
<td>min_processors</td>
<td>min_processors</td>
</tr>
<tr>
<td>Adapter requirement</td>
<td>parallel_path</td>
</tr>
<tr>
<td></td>
<td>Adapter requirement</td>
</tr>
</tbody>
</table>

These keywords are used in the examples in this chapter, and are described in more detail in Chapter 12, “Job command file keywords,” on page 97.

If you disable the default LoadLeveler scheduler to run an external scheduler, see “Usage notes” on page 314 for an explanation of which keywords are supported.

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 are listed in each column

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Valid Combinations</th>
</tr>
</thead>
<tbody>
<tr>
<td>total_tasks</td>
<td>X</td>
</tr>
<tr>
<td>tasks_per_node</td>
<td></td>
</tr>
<tr>
<td>node = &lt;min, max&gt;</td>
<td>X</td>
</tr>
<tr>
<td>node = &lt;number&gt;</td>
<td></td>
</tr>
<tr>
<td>min_processors</td>
<td>X</td>
</tr>
<tr>
<td>max_processors</td>
<td></td>
</tr>
<tr>
<td>task_geometry</td>
<td></td>
</tr>
<tr>
<td>blocking</td>
<td></td>
</tr>
</tbody>
</table>

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
```
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.

```plaintext
# @ 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:

```plaintext
# @ 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**

<table>
<thead>
<tr>
<th>Machine</th>
<th>Available Initiators</th>
<th>Assigned Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>N2</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>N3</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

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
Keyword considerations for parallel jobs

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

<table>
<thead>
<tr>
<th>Machine</th>
<th>Available Initiators</th>
<th>Assigned Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>N3</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>N2</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>N1</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

The assignment function begins with N3 (because N3 has the most initiators available), and assigns 8 tasks, N2 takes six, and N1 takes the remaining 3.

task_geometry

The task_geometry keyword allows you to specify which tasks run together on the same machines, although you cannot specify which machines. In this example, the task_geometry keyword groups 7 tasks to run on 3 nodes:

# @ task_geometry = ((5,2) (1,3) (4,6,0))

The entire task_geometry expression must be enclosed within braces. The task IDs for each node must be enclosed within parenthesis, and must be separated by commas. The entire range of task IDs that you specify must begin with zero, and must end with the task ID which is one less than the total number of tasks. You can specify the task IDs in any order, but you cannot skip numbers (the range of task IDs must be complete). Commas may only appear between task IDs, and spaces may only appear between nodes and task IDs.

Submitting jobs that use striping

When communication between parallel tasks occurs only over a single device such as 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 transmits the buffer and the application must wait for the device to transmit and empty the buffer before it can refill the buffer. Thus the application and the device must wait for each other and this wastes time.

The technique of striping refers to using two or more communication paths to implement a single communication path as perceived by the application. As the application sends data, it fills up a buffer on one device. As that buffer is transmitted over the first device, the application's data begins filling up a second buffer and the application perceives no delay in being able to write. When the second buffer is full, it begins transmission over the second device and the application moves on to the next device. When all devices have been used, the application returns to the first device. Much, if not all of the buffer on the first device has been transmitted while the application wrote to the buffers on the other devices so the application waits for a minimal amount of time or possibly does not wait at all.

LoadLeveler supports striping in two ways. When multiple networks or switch planes are present, striping over them is indicated by requesting the device csss (multiple switch planes) or sn_all (multiple networks). If multiple adapters are present on the same network and the communication subsystem, such as LAPI,
Keyword considerations for parallel jobs

supports striping over multiple adapters on the same network, specifying the
instances keyword on the network statement requests striping over adapters on
the same network. The instances keyword is new and specifies the number of
adapters on a single network to stripe on. It is possible to stripe over multiple
networks and over multiple adapters on each network by specifying both sn_all
and a value for instances greater than one.

- **User space striping:** When sn_all is specified on a network statement with US
  mode, LoadLeveler commits an equivalent set of adapter resources (adapter
  windows and memory) on each of the networks present in the system to the job
  on each node where the job runs. The communication subsystem is initialized to
  indicate that it should use the user space communication protocol on all the
  available switch adapters to service communication requests on behalf of the
  application.

- **IP striping:** When the 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 IP address that is used is different each time a choice has to be made in an
  attempt to balance the adapter use. Multi-link addresses must be configured on
  the system prior to running LoadLeveler and they are specified with the
  multilink_address keyword on the switch adapter stanza in the administration
  file. If a multi-link address is specified for a node, LoadLeveler assigns the
  multi-link address and multi-link IP name to the striping adapter on that node.
  If a multi-link address is not present on a node, the 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.
  Jobs that request striping (both user space and IP) can be submitted to nodes
  with only one switch adapter. In that situation, the result is the same as if the
  job requested no striping.

**Note:** When configured, a multi-link address is associated with the virtual ml0
device. The IP address of this device is the multi-link address. The
illexSDR and illexRPD programs will create a stanza for the ml0 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
ml0 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 ml0 only to have the job fail because
the switch adapters that ml0 stripes over were down.

- **Striping over one network:** If the instances keyword is specified on a network
  statement with a value greater than one, LoadLeveler allocates multiple sets of
  resources for the protocol using as many sets as the instances keyword
  specified. For User Space jobs, these sets are adapter windows and memory. For
  IP jobs, these sets are IP addresses. If multiple adapters exist on each node on
  the same network, then these sets of adapter resources will be distributed among
  all the available adapters on the same network. Even though LoadLeveler will
  allocate resources to support striping over a single network, the communication
  subsystem must be capable of exploiting these resources in order for them to be
  used.
Understanding striping over multiple networks

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 network or fabric.

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

- If a three node job requests striping over networks, it will be dispatched to Node 1, Node 2 and Node 4 where it can communicate on Network 1. It cannot run on Node 3 because that node only has a common communication path with Node 2, namely Network 0.
- If a three node job requests no striping, it will not be run because there are not enough adapters connected to Network 0 to run the job. Notice both the adapter connected to Network 0 on Node 1 and the adapter connected to Network 0 on Node 4 are both at fault.
- 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 communication path to run the job. If enough nodes connected to a common communication path cannot be found, no communication can take place and the job will not run.
Understanding striping over a single network

The figure above shows a network configuration where the adapters support striping over a single network. Both Adapter A and Adapter B on a node are connected to Network 0. The entire oval represents the physical network and the concentric ovals (shaded differently) represent the separate communication paths created for a job by the instances keyword on the network statement. In this case a three node job requests two instances for communication. On Node 1, adapter A is used for instance 0 and adapter B is used for instance 1. There is no requirement to use the same adapter for the same instance so on Node 2, adapter B was used for instance 0 and adapter A for instance 1.

On Node 3, where a fault is keeping adapter B from connecting to the network, adapter A is used for both instance 0 and instance 1 and Node 3 is available for the job to use.

The network itself does not impose any limitation on the total number of communication paths that can be active at a given time for either a single job or all the jobs using the network. As long as nodes with adapter resources are available, additional communication paths can be created.

Using striping

You request that a job be run using striping with the network statement in your job command file. The default when instances is not specified for a job in the network statement is controlled by the class stanza keyword max_protocol_instances for csss and sn_all. Refer to 356 for more information.

Shown here are examples of IP and user space network modes.

---

**Keyword considerations for parallel jobs**

Chapter 7. Special considerations for parallel jobs  61
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
```

Requesting striping over a single network
To request IP striping over multiple adapter on a single network, the network statement would look like this:

```
network.MPI = sn_single,,IP,,instances=2
```

If the nodes on which the job runs have two or more adapters on the same network, two different IP addresses will be allocated to each task for MPI communication. If only one adapter exists per network, the same IP address will be used twice for each task for MPI communication.

Requesting striping over multiple networks and multiple adapters on the same network
To submit a user space job that will stripe MPI communication over multiple adapters on all networks present in the system the network statement would look like this:

```
network.MPI = sn_all,,US,,instances=2
```

If, on a node where the job runs, there are two adapters on each of the two networks, one adapter window would be allocated from each adapter for MPI communication by the job. If only one network were present with two adapters, one adapter window from each of the two adapters would be used. If two networks were present but each only had one adapter on it, two adapter windows from each adapter would be used to satisfy the request for two instances.

For more information on the `network` statement, see `network` on page 109.

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 3.2 and others running LoadLeveler 3.1) see `LoadLeveler 3.1 and 3.2 coexistence` on page xxi.

For information on setting up POE for interactive use see `Allowing users to submit interactive POE jobs` on page 79.
Running MPICH and MPICH-GM jobs under Linux

Considerations for MPICH and MPICH-GM jobs under Linux

MPICH is an open-source, portable implementation of the Message-Passing Interface (MPI) Standard developed by Argonne National Laboratory. It contains a complete implementation of version 1.2 of the MPI Standard and also significant parts of MPI-2, particularly in the area of parallel I/O. MPICH is currently used by a large number of providers of MPI implementations. Additional documentation for MPICH is available from the Argonne National Laboratory web site at http://www-unix.mcs.anl.gov/mpi/mpich/docs.html. MPICH-GM is a port of MPICH on top of GM (ch_gm) and is supported by Myrinet. Additional documentation for MPICH-GM is available from the Myrinet web site at http://www.myri.com/scs/.

MPICH and MPICH-GM are the two MPI implementations supported by LoadLeveler for Linux. For either, LoadLeveler allocates the machines to run the parallel job and starts the implementation specific script. LoadLeveler has no interaction with the parallel tasks started with the script.

When using MPICH, the mpirun script is run on the first machine allocated to the job. Cancelling the job with the llcancel command may not completely remove the MPICH application since the llcancel command terminates only the mpirun script, not the MPICH tasks. The mpirun script will manage the actual execution of the parallel tasks on the other nodes included in the LoadLeveler cluster. A sample MPICH job command file is shown in “MPICH” on page 450.

When using MPICH-GM, the mpirun.ch_gm script is run on the first machine allocated to the job. As with MPICH, llcancel only terminates the mpirun.ch_gm script. In order to terminate the parallel tasks, mpirun.ch_gm should be started with the --gm-kill 0 option. The mpirun.ch_gm script will manage the actual execution of the parallel tasks on the other nodes included in the LoadLeveler cluster. A sample MPICH-GM job command file is shown in “MPICH-GM” on page 452.

LoadLeveler does not manage the GM ports on the Myrinet switch. For LoadLeveler to keep track of the GM ports they must be identified as LoadLeveler consumable resources. See “Configuring LoadLeveler for MPICH-GM” on page 82 for more details.

Job command file examples

Examples of building job command files for the parallel environment, including MPICH and MPICH-GM jobs, can be found under “User tasks: building parallel job command files” on page 445.

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 198 for samples of output using the -x and -t options with the llq command. As an alternative, you can also use the GUI and select: Job Actions ➤ Extended Details.
Obtaining status of parallel jobs

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.

```bash
#!/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
Part 4. Administrator tasks

Chapter 8. Administering and configuring LoadLeveler

Overview .................................................. 67
Planning considerations .................................. 67
  Where to begin? ......................................... 68
    Intermediate or beginner ........................... 69
    Expert .................................................. 69
Quick set up .............................................. 69
Administering LoadLeveler ............................. 70
  Administration file structure and syntax .......... 70
Configuring LoadLeveler ................................ 71
  The configuration files ............................... 72
    Customizing the configuration file ............... 72
  Configuration file structure and syntax .......... 73
    Numerical and alphabetical constants .......... 73
    Mathematical operators ............................ 73
    User-defined variables ............................ 73
    LoadLeveler variables ............................. 74
  Considerations for integrating LoadLeveler with
    AIX Workload Manager .............................. 77
Keyword summary ....................................... 78

Chapter 9. Administration tasks for parallel jobs

Scheduling considerations for parallel jobs .......... 79
Allowing users to submit interactive POE jobs ....... 79
Allowing users to submit PVM jobs ................. 80
  Restrictions and limitations for PVM jobs ...... 81
Setting up a class for parallel jobs ................. 81
Setting up a parallel master node ................... 82
Configuring LoadLeveler for MPICH-GM .............. 82

Chapter 10. Gathering job accounting data

Collecting job resource data on serial and parallel
  jobs .................................................... 83
Collecting job resource data on machines .......... 83
Collecting job resource data on events ............. 84
Collecting job resource information based on user
  accounts .............................................. 84
Collecting the accounting information and storing it
  into files ............................................ 85
Accounting reports .................................... 85
Job accounting setup procedure ..................... 86

Chapter 11. Routing jobs to NQS machines

Setting up the NQS environment ..................... 87
Designating machines to which jobs will be routed 88
NQS scripts ............................................ 88
NQS machine job routing procedure .................. 89
Chapter 8. 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 9, “Administration tasks for parallel jobs,” on page 79.

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 71.

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 user ID number for a user) and gid (the group ID number for a group) on every machine in the cluster.

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

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

Dynamic adapter configuration

LoadLeveler can dynamically determine the adapters in any OSI that has RSCT installed. This is true for an OSI that is in either a PEER domain or a PSSP domain. LoadLeveler must be told on an OSI basis if it is to handle dynamic adapter configuration changes for that OSI. The specification of whether to use dynamic or static adapter configuration for an OSI is done through the presence or absence of the machine stanza’s adapter_stanzas keyword.

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

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” on page 69. If you are relatively new to job scheduling tasks, begin by reading “Intermediate or beginner” on page 69.
Planning considerations

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 70 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.”

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.” 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 79.

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 or Cluster Security Services. For more information about configuring security services for LoadLeveler, see “Step 16: Configuring LoadLeveler to use a security service” on page 401. 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 71 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 368 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 344 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.)

```bash
llctl -h hostname start
```

Issue this command for the central manager machine first. See “llctl - Control LoadLeveler daemons” on page 169 for more information.

You can also issue the following command to start LoadLeveler on all machines, except submit-only machines, listed in the administration file. The central manager machine is the first started, followed by other machines in the order listed in the administration file.

```bash
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 344 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 344 for more information.

**User stanza**

Defines LoadLeveler users and their characteristics. See “Step 2: Specify user stanzas” on page 350 for more information.

**Class stanza**

Defines the characteristics of the job classes. See “Step 3: Specify class stanzas” on page 353 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 363 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 365 for more information.

Stanzas have the following general format:
The following is a simple example of an administration file illustrating several stanzas:

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

*Figure 16. Format of 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
Configuring LoadLeveler

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:

**LoadLUserID**
Specifies the LoadLeveler user ID.

**LoadLGroupID**
Specifies the LoadLeveler group ID.

**LoadLConfig**
Specifies the full path name of the configuration file.

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

If Cluster Security Services is enabled, make sure you update the **unix.map** file if the **LoadLUserID** is specified as something other than **loadl**. Refer to the "Enabling Cluster Security Services" on page 407 for more details.

You can also override the `/etc/LoadL.cfg` file. For an example of when you might want to do this, see the "Querying multiple LoadLeveler clusters" on page 49.

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 the "Step 11: Specify where files and directories are located" on page 386 for more information.

Customizing the configuration file

All LoadLeveler commands, daemons, and processes read the administration and configuration files at start up time. If you change the administration or configuration files after LoadLeveler has already started, any LoadLeveler command or process, such as the **LoadL_starter** process, will read the newer
version of the files while the running daemons will continue to use the data from the older version. To ensure that all LoadLeveler commands, daemons, and processes use the same configuration data, run the reconfiguration command on all machines in the cluster each time the administration or configuration files are changed. “Customizing the global and local configuration file” on page 368 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 130 and in “Step 17: Specify additional configuration file keywords” on page 410
- User-defined variables, described in “User-defined variables”
- LoadLeveler variables, described in “LoadLeveler variables” on page 74

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.

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)
```

**Connectivity**

The ratio of the number of active switch adapters on a node to the total number of switch adapters on the node. The value ranges from 0.0 (all switch adapters are down) to 1.0 (all switch adapters are active). A node with no switch adapters has a connectivity of 0.0. Connectivity can be used in a MACHPRIO expression to favor nodes that do not have any down switch adapters or in a job’s REQUIREMENTS to require only nodes with a certain connectivity.

**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 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.
**Configuring LoadLeveler**

**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 **llsubmit**.

**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.

**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:
- \textbf{tm\_mday}  
  The number of the day of the month (1-31).
- \textbf{tm\_wday}  
  Number of days since Sunday (0-6).
- \textbf{tm\_yday}  
  Number of days since January 1 (0-365).
- \textbf{tm\_mon}  
  Number of months since January (0-11).
- \textbf{tm\_year}  
  The number of years since 1900 (0-9999). For example:
  \begin{verbatim}
  tm\_year == 100
  \end{verbatim}
  Denotes the year 2000.
- \textbf{tm\_year}  
  The integer representation of the current year. For example:
  \begin{verbatim}
  tm\_year == 2010
  \end{verbatim}
  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 \texttt{ConsumableCPUs} and \texttt{ConsumableMemory}. If you want to control these resources, AIX Workload Manager (WLM) can be integrated with LoadLeveler to balance workloads at the machine level.

| Linux notes | WLM is not supported in LoadLeveler for Linux. |

Workload balancing is done by assigning relative priorities to job processes. These job priorities prevent one job from monopolizing system resources when that resource is under contention. To integrate LoadLeveler and WLM you must:

1. Define ConsumableCpus, ConsumableMemory, or both as consumable resources in the \texttt{SCHEDULE\_BY\_RESOURCES} global configuration keyword. This enables the LoadLeveler scheduler to consider these consumable resources.
2. Define ConsumableCpus, ConsumableMemory, or both in the \texttt{ENFORCE\_RESOURCE\_USAGE} global configuration keyword. This enables enforcement of these consumable resources by AIX WLM.
3. Define hard, soft or shares in the \texttt{ENFORCE\_RESOURCE\_POLICY} configuration keyword. This defines what policy is used by LoadLeveler when setting WLM class resource entitlements.
4. Using the \texttt{resources} keyword of the machine stanza, define the CPU and real memory machine resources available for user jobs.
Configuring LoadLeveler

- 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 ENFORCERESOURCE_USAGE keyword enabled and issue wlmstat -v -m. Look at the npg column to determine how much memory is being used by these classes.

5. Decide if all jobs should have their CPU or real memory resources enforced and then define the ENFORCERESOURCE_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 ENFORCERESOURCE_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 ENFORCERESOURCE_USAGE and the ENFORCERESOURCE_SUBMISSION keywords, see Step 4: Define consumable resources” on page 376.

Keyword summary

Listings of the keywords used in administration and configuration files can be found under:
- “Administration file keywords” on page 125
- “Configuration file keywords and LoadLeveler variables” on page 130
Chapter 9. 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 8, “Administering and configuring LoadLeveler,” on page 67. For information on submitting parallel jobs, see Chapter 7, “Special considerations for parallel jobs,” on page 55.

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 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 370.

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 69 for information on defining LoadLeveler administrators.

2. If running user space jobs, LoadLeveler must be configured to use switch adapters. A way to do this is to run the llextSDR or llextRPD command to extract node and adapter information from the SDR or from the RSCT peer domain. See “llextSDR - Extract adapter information from the SDR” on page 180 and “llextRPD - Extract data from an RSCT peer domain” on page 177 for additional information on the commands.

3. 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 370 for more information.

4. 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 344 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.
Allowing users to submit interactive POE jobs

- 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 344 for more information on these keywords.

5. Consider setting up a class stanza for your interactive POE jobs. See “Setting up a class for parallel jobs” on page 81 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 350 for more information.

Allowing users to submit PVM jobs

**Note:** This is the last release to support PVM3 job type.

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 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
allowing users to submit pvm jobs

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 463 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:

```plaintext
long_parallel: type=class
wall_clock_limit = 1800
include_users = jack queen king ace
priority = 50
total_tasks = 120
max_node = 60
maxjobs = 2
```

Note the following about this class stanza:

- The wall_clock_limit keyword sets a wall clock limit of 1800 seconds (30 hours) for jobs in this class
- The include_users keyword allows four users to submit jobs in this class
- The priority keyword sets a relative priority of 50 for jobs in this class
- The total_tasks keyword specifies that a user can request up to 120 total tasks for a job in this class
- 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:

```plaintext
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:

```plaintext
Class = { "long_parallel" "long_parallel" "long_parallel" "long_parallel" }
```
Setting up a class for parallel jobs

or

Class = long_parallel(4)

For more information, see “Step 3: Define LoadLeveler machine characteristics” on page 374.

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 344 and “Step 3: Specify class stanzas” on page 353.

Configuring LoadLeveler for MPICH-GM

LoadLeveler does not manage the GM ports on the Myrinet switch. For LoadLeveler to keep track of the GM ports they must be identified as LoadLeveler consumable resources. To use consumable resources to manage GM ports, take the following steps:

1. Pick a name for the GM port resource. For this example we will use the name `gmports`. Users submitting MPICH-GM jobs need to know the name used for the GM port resource.

2. Add the following statement to the LoadLeveler configuration file:

   ```
schedule_by_resources = gmports
   
   or add `gmports` to an existing `schedule_by_resources` line.
   
3. Specify how many GM ports are available on each machine by adding the line:

   ```
   resources=gmports(n)
   ```

   to each machine stanza where `n` is the number of `gmports`. To determine the value of `n` use either the number specified in the GM documentation or the number of GM ports you have successfully used. Certain system configurations may not support all available GM ports, so you may need to specify a lower value for the `gmports` resource than what is actually available.

4. Users submitting LoadLeveler jobs that use GM ports must include `gmports(1)` on the `resources` statement. This may require adding the resources statement to the job command file.
Chapter 10. 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 ilq command. In order to enable ilq -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 384.

\[ JOB\_ACCT\_Q\_POLICY = number \]

Where \textit{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 ilq -x command. The default is 300 seconds.

\begin{tabular}{|l|}
\hline
\textbf{Linux notes} \\
\textit{MPICH and MPICH-GM job account information is not available in LoadLeveler for Linux.} \\
\hline
\end{tabular}

\[ JOB\_LIMIT\_POLICY = number \]

Where \textit{number} is an amount of time in seconds. The smaller of \textit{JOB\_LIMIT\_POLICY} and \textit{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 \textit{JOB\_LIMIT\_POLICY} is \textit{POLLING\_FREQUENCY} multiplied by \textit{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.
Collecting job resource data based on machines

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 344.

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   m  day  command
#
# 00  08  *   *   /u/loadl/bin/llctl -g capture graveyard
# 00  16  *   *   /u/loadl/bin/llctl -g capture first
# 00  00  *   *   /u/loadl/bin/llctl -g capture second
```

For more information on the `llctl` command, refer to “llctl - Control LoadLeveler daemons” on page 169. For more information on the collection of accounting records, see “llq - Query job status” on page 198.

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 12, “Job command file keywords,” on page 97. 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.
Collecting job resource information based on user accounts

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
- 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 384.

- 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.YYYYYMMDDHHmm**. 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
Accounting reports

A 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 228, for examples of the short and extended versions of the report.

For information on the accounting Application Programming Interfaces, refer to Chapter 5, “LoadLeveler API interface,” on page 39.

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 415.
Chapter 11. 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.

Linux notes

LoadLeveler for Linux does not support NQS. If NQS is enabled, the LoadLeveler daemons on the Linux nodes will not start.

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

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.

• 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:

- a  startdate
- e  error
- ke ignored
- ko ignored
- lc core_limit
- ld data_limit
- lf file_limit
- lm rss_limit
- lM ignored
- ln ignored
- Is ignored
NQS scripts

lt cpu_limit
IT ignored
lv ignored
lw ignored
mb notification (always)
me notification (complete)
mu notify_user
nr restart = no
o output
p user_priority
q class
r ignored
re ignored
ro ignored
s shell
x environment = copyall
z 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 417.
NQS machine job routing procedure
## Part 5. Detailed descriptions

<table>
<thead>
<tr>
<th>Chapter 12. Job command file keywords</th>
<th>Chapter 13. Administration and Configuration file keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>account_no</td>
<td>Administration file keywords</td>
</tr>
<tr>
<td>arguments</td>
<td>Configuration file keywords and LoadLeveler variables</td>
</tr>
<tr>
<td>blocking</td>
<td>· Keywords</td>
</tr>
<tr>
<td>checkpoint</td>
<td>· User-defined keywords</td>
</tr>
<tr>
<td>ckpt_dir</td>
<td>· LoadLeveler variables</td>
</tr>
<tr>
<td>ckpt_file</td>
<td>· Daemons</td>
</tr>
<tr>
<td>ckpt_time_limit</td>
<td>· The master daemon</td>
</tr>
<tr>
<td>class</td>
<td>· The sched daemon</td>
</tr>
<tr>
<td>comment</td>
<td>· The startd daemon</td>
</tr>
<tr>
<td>core_limit</td>
<td>· The starter process</td>
</tr>
<tr>
<td>cpu_limit</td>
<td>· The negotiator daemon</td>
</tr>
<tr>
<td>data_limit</td>
<td>· The kbdd daemon</td>
</tr>
<tr>
<td>dependency</td>
<td>· The gsmonitor daemon</td>
</tr>
<tr>
<td>environment</td>
<td>· Job states</td>
</tr>
<tr>
<td>error</td>
<td>· Daemons</td>
</tr>
<tr>
<td>executable</td>
<td>· The master daemon</td>
</tr>
<tr>
<td>file_limit</td>
<td>· The schedd daemon</td>
</tr>
<tr>
<td>group</td>
<td>· The startd daemon</td>
</tr>
<tr>
<td>hold</td>
<td>· The startd daemon</td>
</tr>
<tr>
<td>image_size</td>
<td>· The negotiator daemon</td>
</tr>
<tr>
<td>initialdir</td>
<td>· The kbdd daemon</td>
</tr>
<tr>
<td>input</td>
<td>· The gsmonitor daemon</td>
</tr>
<tr>
<td>job_cpu_limit</td>
<td>· Job states</td>
</tr>
<tr>
<td>job_name</td>
<td>· Daemons</td>
</tr>
<tr>
<td>job_type</td>
<td>· The master daemon</td>
</tr>
<tr>
<td>large_page</td>
<td>· The sched daemon</td>
</tr>
<tr>
<td>max_processors</td>
<td>· The startd daemon</td>
</tr>
<tr>
<td>min_processors</td>
<td>· The starter process</td>
</tr>
<tr>
<td>network</td>
<td>· The negotiator daemon</td>
</tr>
<tr>
<td>node</td>
<td>· The kbdd daemon</td>
</tr>
<tr>
<td>node_usage</td>
<td>· The gsmonitor daemon</td>
</tr>
<tr>
<td>notification</td>
<td>· Job states</td>
</tr>
<tr>
<td>notify_user</td>
<td>· Daemons</td>
</tr>
<tr>
<td>output</td>
<td>· The master daemon</td>
</tr>
<tr>
<td>parallel_path</td>
<td>· The schedd daemon</td>
</tr>
<tr>
<td>preferences</td>
<td>· The startd daemon</td>
</tr>
<tr>
<td>queue</td>
<td>· The negotiator daemon</td>
</tr>
<tr>
<td>requirements</td>
<td>· The kbdd daemon</td>
</tr>
<tr>
<td>resources</td>
<td>· Job states</td>
</tr>
<tr>
<td>restart</td>
<td>· Daemons</td>
</tr>
<tr>
<td>restart_from_ckpt</td>
<td>· The master daemon</td>
</tr>
<tr>
<td>restart_on_same_nodes</td>
<td>· The schedd daemon</td>
</tr>
<tr>
<td>rss_limit</td>
<td>· The startd daemon</td>
</tr>
<tr>
<td>shell</td>
<td>· The negotiator daemon</td>
</tr>
<tr>
<td>stack_limit</td>
<td>· The kbdd daemon</td>
</tr>
<tr>
<td>startdate</td>
<td>· Job states</td>
</tr>
<tr>
<td>step_name</td>
<td>· Daemons</td>
</tr>
<tr>
<td>task_geometry</td>
<td>· The master daemon</td>
</tr>
<tr>
<td>tasks_per_node</td>
<td>· The schedd daemon</td>
</tr>
<tr>
<td>total_tasks</td>
<td>· The startd daemon</td>
</tr>
<tr>
<td>user_priority</td>
<td>· The negotiator daemon</td>
</tr>
<tr>
<td>wall_clock_limit</td>
<td>· The kbdd daemon</td>
</tr>
<tr>
<td>Job command file variables</td>
<td>· Job states</td>
</tr>
<tr>
<td>· Example 1</td>
<td>· Daemons</td>
</tr>
<tr>
<td>· Example 2</td>
<td>· The master daemon</td>
</tr>
</tbody>
</table>

**Chapter 15. Commands**
- llaacctmg - Collect machine history files
- llcancel - Cancel a submitted job
- llckpt - Checkpoint a running job step
- llclass - Query class information
- llnctl - Control LoadLeveler daemons
- lldcergpmaint - LoadLeveler DCE group maintenance utility
- llfextRPD - Extract data from an RSCT peer domain
- llfextSDR - Extract adapter information from the SDR
- llfavorjob - Reorder system queue by job
- llfavoruser - Reorder system queue by user
- llhold - Hold or release a submitted job
- lltli - Initialize machines in the LoadLeveler cluster
- lltli - Change attributes of a submitted job step
- llipreempt - Preempt a submitted job step
- llpri - Change the user priority of submitted job steps
- llq - Query job status
- llstatus - Query machine status
- llsli - Submit a job
- llsli - Return job resource information for accounting

**Chapter 16. Application Programming Interfaces (APIs)**
- Accounting API
- Account validation user exit
- Purpose
- Syntax
- Parameters

© Copyright IBM Corp. 2001, 2003, 2004

91
| Step 16: Configuring LoadLeveler to use a security service | 401 |
| Configuring LoadLeveler to use DCE security services | 401 |
| Configuring LoadLeveler to use Cluster Security Services | 407 |
| Tips for configuring LoadLeveler to use Cluster Security Services | 409 |
| Limiting which security mechanisms LoadLeveler can use | 410 |
| Step 17: Specify additional configuration file keywords. | 410 |
| Setting up job accounting files. | 415 |
| Task 1: Update the configuration file | 415 |
| Task 2: Merge multiple files collected from each machine into one file | 415 |
| Task 3: Report job information on all the jobs in the history file | 416 |
| Task 4: Using account numbers and setting up account validation. | 416 |
| Task 5: Specifying machines and their weights | 417 |
| Routing jobs to NQS machines | 417 |
| Task 1: Modify the administration file | 418 |
| Task 2: Modify the configuration file | 418 |
| Task 3: Submit the jobs | 418 |
| Task 4: Obtain status of NQS jobs | 420 |
| Task 5: Cancel NQS jobs. | 420 |

Chapter 18. Preemption using the Gang scheduler

Overview. | 421 |
| Supported hardware | 422 |
| Application support | 422 |
| Keywords specific to Gang scheduling | 422 |
| Configuration file keywords for Gang scheduling preemption | 422 |
| Configuration file keyword summary | 422 |
| Configuration file keyword details | 422 |
| Sample configuration file | 425 |
| Administration file keywords for Gang | 425 |
| Administration file keyword summary | 426 |
| Administration keyword details | 426 |
| Sample administration file | 426 |
| Gang scheduling interactions and restrictions. | 428 |
| Reconfiguration | 428 |
| Circular preemption | 428 |
| Restrictions for Gang scheduling preemption | 428 |
| Implied START_CLASS values | 428 |
| Last one wins rule. | 429 |
| Job command file and Gang scheduling | 429 |
| LoadLeveler commands for Gang | 429 |
| APIs used with Gang scheduling. | 430 |

Chapter 19. Support for 64-bit applications

64-bit support for Job Command, Configuration, and Administration keywords | 431 |
| 64-bit support for Job Command file keywords | 431 |
| 64-bit support for Administration keywords | 432 |
| 64-bit support for Configuration keywords and expressions | 432 |
Chapter 12. 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.

If a blank value is used after the equal sign it is as if no keyword was specified.

account_no

Supports centralized accounting. Allows you to specify an account number to associate with a job. This account number is stored with job resource information in local and global history files. It may also be validated before LoadLeveler allows a job to be submitted. For more information, see Chapter 10, “Gathering job accounting data,” on page 83.

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 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:
Job command file keywords

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:

\[
\text{blocking} = 4 \\
\text{total\_tasks} = 17
\]

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.

Linux notes
If a job with \text{checkpoint} = \text{interval} or \text{checkpoint} = \text{yes} is dispatched to Linux machines it is rejected.

The syntax is:

\text{checkpoint} = \text{interval} \mid \text{yes} \mid \text{no}

Where:

\text{interval}
Specifies that LoadLeveler will automatically checkpoint your program at preset intervals. The time interval is specified by the settings in the \text{MIN\_CKPT\_INTERVAL} and \text{MAX\_CKPT\_INTERVAL} keywords in the configuration file. Since a job with a setting of \text{interval} is considered checkpointable, you can initiate a checkpoint using any method in addition to the automatic checkpoint. The difference between \text{interval} and \text{yes} is that \text{interval} enables LoadLeveler to automatically take checkpoints on the specified intervals while the value \text{yes} does not enable that ability.

\text{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 \text{yes}, LoadLeveler will not checkpoint on the intervals specified by the \text{MIN\_CKPT\_INTERVAL} and \text{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:

```plaintext
checkpoint = yes
```

For detailed information on checkpointing, see “Step 14: Enable checkpointing” on page 392.

**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 392.

The syntax is:

```plaintext
ckpt_dir = pathname
```

For example, if checkpoint files were to be stored in the `/tmp` directory the job command file would include:

```plaintext
ckpt_dir = /tmp
```

For more information on naming directories for checkpointing, see “Naming checkpoint files and directories” on page 393.

**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:

```plaintext
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:

```plaintext
ckpt_file = myckptfiles
```
Job command file keywords

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:

\texttt{ckpt\_file = /tmp/myckptfiles}

or the combination of \texttt{ckpt\_dir} and \texttt{ckpt\_file} keywords can be used, producing the same result.

\texttt{ckpt\_dir = /tmp}
\texttt{ckpt\_file = myckptfiles}

For more information on naming files for checkpointing, see “Naming checkpoint files and directories” on page 393.

Notes:
1. The value for the \texttt{ckpt\_file} keyword is case sensitive.
2. The keyword \texttt{ckpt\_file} is not allowed in the command file for interactive POE sessions.

\texttt{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:

\texttt{ckpt\_time\_limit = hardlimit,softlimit}

Some examples of the checkpoint time limit include:

\texttt{ckpt\_time\_limit = 00:10:00,00:05:00}
\texttt{ckpt\_time\_limit = 12:30,7:10}
\texttt{ckpt\_time\_limit = rlim\_infinity}
\texttt{ckpt\_time\_limit = unlimited}

For detailed information on limits and additional examples, see “Limit keywords” on page 357.

\texttt{class}

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

The syntax is:

\texttt{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:

\texttt{class = largejobs}

\texttt{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 357 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 357 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 357 for more information on the values and units you can use with this keyword.
Job command file keywords

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:

\[ \text{dependency} = \text{expression} \]

where the syntax for the expression is:

\[ \text{step_name} \text{ operator} \text{ value} \]

where \text{step_name} (as described in [“step_name” on page 121]) must be a previously defined job step and \text{operator} can be one of the following:

- \text{==} Equal to
- \text{!=} Not equal to
- \text{<=} Less than or equal to
- \text{>=} Greater than or equal to
- \text{<} Less than
- \text{>} Greater than
- \text{&&} And
- \text{||} Or

The \text{value} is usually a number which specifies the job return code to which the \text{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, \textit{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:

\[ \text{dependency} = (\text{step1} \text{ == } 0) \]

**Example 2:** In the following example, step1 will run with the executable called \textit{myprogram1}. Step2 will run only if LoadLeveler removes step1 from the system. If step2 does run, the executable called \textit{myprogram2} gets run.

```plaintext
# 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 login initial environment variables set by LoadLeveler when your job step
starts. If the same environment variables are set in the user's initialization files
(such as the .profile) those set by the login initialization files will supersede those
set by LoadLeveler.

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 no 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 40 on page 443 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 357 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:

\[ \text{group} = \text{group\_name} \]

For example:

\[ \text{group} = \text{my\_group\_name} \]

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:

\[ \text{hold} = \text{user|system|usersys} \]

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

\[ \text{hold} = \text{user} \]

To remove the hold on the job, you can use either the GUI or the \texttt{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:

\[ \text{image\_size} = \text{number} \]

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

\[ \text{image\_size} = 11 \]

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.
Job command file keywords

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:
\[ \text{initialdir} = \text{pathname} \]

For example:
\[ \text{initialdir} = /var/home/mike/ll_work \]

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:
\[ \text{input} = \text{filename} \]

For example:
\[ \text{input} = \text{input.$(process)} \]

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 83.

The syntax is:
\[ \text{job_cpu_limit} = \text{hardlimit,softlimit} \]

For example:
job_cpu_limit = 12:56,12:50

See "Limit keywords" on page 357 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.
  
  Note: This is the last release where PVM3 is supported.

- `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.

large_page

Specifies whether or not a job step requires Large Page support from AIX. The syntax is:

```
large_page = value
```
Job command file keywords

where value can be Y, M, or N. Y informs LoadLeveler to use Large Page memory, if available, but to otherwise use regular memory. M means use of Large Page memory is mandatory.

### Linux notes

Large Page memory is not supported in LoadLeveler for Linux. In this case, specifying M would cause the job to never be sent.

N, the default value, means to not use Large Page memory.

For example:

```
large_page = Y
```

would ask LoadLeveler to use Large Page memory for the job step, if available.

### max_processors

Specifies the maximum number of nodes requested for a parallel job, regardless of the number of processors contained in the node.

This keyword is equivalent to the maximum value you specify on the node keyword. In any new job command files you create for 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 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 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 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.

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 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:

\[ \text{network.protocol} = \text{type} [, \text{usage} [, \text{mode} [, \text{comm_level} [, \text{instances}=<\text{number}\text{|max}>]]]] \]

Where:

\text{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.
- **MPI|LAPI** Specifies sharing adapter windows between MPI and LAPI. When you specify network.MPI|LAPI in a job step you cannot specify any other network statements in that job step.
- **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.

<table>
<thead>
<tr>
<th>Linux notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifying network.PVM in LoadLeveler for Linux causes an error. The job will not be submitted.</td>
</tr>
</tbody>
</table>

\text{type}

This field is required and specifies one of the following:

- **adapter_name**
  The possible values are the names associated with the interface cards installed on a node (for example, en0, tk1 and css0 ). Note, when css0 is specified for the HPS switch, it takes on a different meaning described below.

- **network_type**
  Specifies a network_type as specified in the LoadLeveler administration file. The LoadLeveler administrator must specify values used as network_type in the adapter stanza of the LoadLeveler administration file using the network_type keyword. For example, an installation can define a network type of "switch" to identify css0 adapters. For more information on specifying network_type, see "Step 5: Specify adapter stanzas" on page 365.
Job command file keywords

sn_single
When used for the HPS switch it specifies that LoadLeveler use a common, single switch network. When used for a switch other than the HPS switch it is comparable to specifying an adapter_name of css0.

sn_all
Specifies that striped communication should be used over all available switch networks. The networks specified must be accessible by all machines selected to run the job. For more information on striping, see "Submitting jobs that use striping" on page 58.

css
May be used interchangeably with sn_all. This option is provided for compatibility with job command files created with older versions of LoadLeveler.

css0
May be used interchangeably with sn_single. This option is provided for compatibility with job command files created with older versions of LoadLeveler.

The following are optional and if omitted their position must be specified with a comma:

usage
Specifies whether the adapter can be shared with tasks of other job steps. Possible values are shared, which is the default, or not_shared. If not_shared is specified, LoadLeveler can only guarantee that the adapter will not be shared by other jobs running on the same OSI. If the adapter is shared by more than one OSI, LoadLeveler can not guarantee that the adapter is not shared with jobs running on a different OSI.

mode
Specifies the communication subsystem mode used by the communication protocol that you specify, and can be either IP (Internet Protocol), which is the default, or US (User Space). Note that each instance of the US mode requested by a task running on the 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: Administration Guide, SA22-7348 and RSCT Administration Guide, SA22-7889. Specifying a level that is higher than what the job actually needs will not speed up communication, but may make it harder to schedule a job (because it requires more resources). The comm_level keyword can only be specified with US mode. The three communication levels are:

LOW
Implies that minimal inter-task communication will occur.

AVERAGE
This is the default value. Unless you know the specific communication characteristics of your job, the best way to determine the comm_level is through trial-and-error.

HIGH
Implies that a great deal of inter-task communication will occur.

instances
If instances is specified as a number, it indicates the number of parallel
communication paths made available to the protocol on each network. The number actually used will depend on the implementation of the protocol subsystem. If instances are specified by `max`, the actual value used is determined by the `MAX_PROTOCOL_INSTANCES` for the class to which the job is submitted. The default value for instances is 1.

For the best performance set `MAX_PROTOCOL_INSTANCES` so that the communication subsystem uses every available adapter before it reuses any of the adapters.

**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 = sn_single,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 = sn_single,,IP
```

Because a shared adapter is the default, you do not need to specify `shared`.

**Example 3:** A communication level can only be specified if User Space mode is also specified:
```
network.MPI = sn_single,,US,AVERAGE
```

Note that LoadLeveler can ensure that an adapter is dedicated (not shared) if you request the adapter in US mode, since any user who requests a user space adapter must do so using the `network` statement. However, if you request a dedicated adapter in IP mode, the adapter will only be dedicated if all other LoadLeveler users who request this adapter do so using the `network` statement.

**Example 4:** `css0` can be used in place of `sn_single`:
```
network.MPI = css0, not_shared,US,HIGH
```

---

**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]
```

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:
Job command file keywords

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 56, “tasks_per_node” on page 121, and “total_tasks” on page 122.

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
   Has the same meaning as not_shared. It is provided for compatibility.

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.
ever
   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 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 to include group support in order for you to use group support within LoadLeveler. `$PVM_ROOT` 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" 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 when run under the LoadLeveler Backfill scheduler. It is also the way to specify when running with the default LoadLeveler scheduler. When using the default scheduler the Adapter requirement is specified as the physical name of the device, such as en0 or css0.
Note that you cannot specify both the **Adapter** requirement and the **network** statement in a job command file.

For the Backfill scheduler you can use the pre-defined network types. 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 platform for which your executable has been compiled. The default is the architecture of the submitting machine.

**Connectivity**

Connectivity is the ratio of the number of active switch adapters on a node to the total number of switch adapters on the node. The value ranges from 0.0 (all switch adapters are down) to 1.0 (all switch adapters are active). A node with no switch adapters has a connectivity of 0.0. Connectivity can be used in a MACHPRIOR expression to favor nodes that do not have any down switch adapters or in a job **REQUIREMENTS** statement to require only nodes with a certain connectivity.

**Disk**

Specifies the amount of disk space in kilobytes you believe is required in the LoadLeveler **execute** directory to run the job step.

*Note:* The Disk variable in an expression associated with the **requirements** and **preferences** keywords are 64-bit integers.

**Feature**

Specifies the name of a feature defined on a machine where you want your job step to run. Be sure to specify a feature in the same way in which the feature is specified in the configuration file. To find out what features are available, use the **Ilstatus** command.

**LargePageMemory**

Specifies the amount, in megabytes, of Large Page Memory required to run the job.

*Note:* The Memory variable in an expression associated with the **requirements** and **preferences** keywords are 64-bit integers.
Job command file keywords

**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 AIX52 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 == "AIX52") || (OpSys == "AIX51"))
```

**Memory**
Specifies the amount, in megabytes, of regular physical memory required in the machine where you want your job step to run.

**Note:** The Memory variable in an expression associated with the requirements and preferences keywords are 64-bit integers.

**OpSys**
Specifies the operating system on the machine where you want your job step to run. It describes the particular kind of platform for which your executable has been compiled. 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.

**TotalMemory**
Specifies the amount, in megabytes, of regular physical memory and Large Page memory required in the machine where you want your job step to run.

**Note:** The Memory variable in an expression associated with the requirements and preferences keywords are 64-bit integers.

**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.

Example 6: To specify the job runs if all switch connections are up, enter:

```
# @ requirements = (Connectivity == 1.0)
```

To specify the job runs if at least half of the switch connections are up, enter:

```
# @ requirements = (Connectivity >= .5)
```

To specify the job runs if there is at least some connectivity, enter:

```
# @ requirements = (Connectivity > 0)
```

resources

Specifies quantities of the consumable resources "consumed" by each task of a job step. The resources may be machine resources or floating resources. 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, except for the following instances where the integer must be greater than zero:

- `ConsumableMemory`
- `ConsumableVirtualMemory`
Job command file keywords

- **ConsumableCpus** when the enforcement policy is *hard* or *soft*

**ConsumableCpus** can have a value of zero when the administrator has not requested that consumable resources be enforced or if the enforcement policy is *shares*. When you set **ConsumableCpus** to zero, the meaning varies depending on whether use is being enforced. With no enforcement, zero means the job is requesting a negligible amount of CPU. With an enforcement policy of *shares*, it means the job is requesting a tiny percentage of available shares.

If the count is not valid then LoadLeveler will issue a message and the job will not be submitted. The allowable units are those normally used with LoadLeveler data limits:

- `b` bytes
- `w` words (4 bytes)
- `kb` kilobytes (2**10 bytes)
- `kw` kilowords (2**12 bytes)
- `mb` megabytes (2**20 bytes)
- `mw` megawords (2**22 bytes)
- `gb` gigabytes (2**30 bytes)
- `gw` gigawords (2**32 bytes)
- `tb` terabytes (2**40 bytes)
- `tw` terawords (2**42 bytes)
- `pb` petabytes (2**50 bytes)
- `pw` petawords (2**52 bytes)
- `eb` exabytes (2**60 bytes)
- `ew` exawords (2**62 bytes)

**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.

**Linux notes**

This keyword is ignored by LoadLeveler for Linux.

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 393 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
```
Job command file keywords

The above example specifies the limits in megabytes, but if no units are specified, then bytes are assumed. See “Limit keywords” on page 357 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

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 357 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 56.

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.
Job command file keywords

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.

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:

\[ \text{user_priority} = \text{number} \]

Where \text{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 \textit{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 \textit{unlimited}. For more information, see “Choosing a scheduler” on page 370.

The syntax is:

\[ \text{wall_clock_limit} = \text{hardlimit},\text{softlimit} \]

An example is:

\[ \text{wall_clock_limit} = 5:00,4:30 \]

See “Limit keywords” on page 357 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:

\[ $(\text{variable name}) \]

The following variables are available to you:

\[ $\text{(host)} \]

The hostname of the machine from which the job was submitted. In a job command file, the $\text{(host)} variable and the $\text{(hostname)} variable are equivalent.

\[ $\text{(domain)} \]

The domain of the host from which the job was submitted.
Job command file variables

$\text{(schedd\_host)}$

The hostname of the scheduling machine.

$\text{(schedd\_hostname)}$

The hostname and domain name of the scheduling machine.

$\text{(jobid)}$

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

$\text{(stepid)}$

The sequential number assigned to this job step when multiple queue statements are used with the job command file. The $\text{(stepid)}$ variable and the $\text{(process)}$ 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 12, “Job command file keywords,” on page 97.

- $\text{(executable)}$
- $\text{(class)}$
- $\text{(comment)}$
- $\text{(job\_name)}$
- $\text{(step\_name)}$

Note that for the $\text{(comment)}$ variable, the keyword definition must be a single string with no blanks. Also, the $\text{executable}$ statement automatically sets the $\text{(base\_executable)}$ variable, which is the file name of the executable without the directory component. See Figure 40 on page 443 for an example of using the $\text{(base\_executable)}$ variable.

Example 1

The following job command file creates an output file called $\text{stance.78.out}$, 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 $\text{computel.step1.March05}$.

```
#$ comment = March05
#$ job\_name = computel
#$ step\_name = step1
#$ executable = my_job
#$ output = $(job\_name).$(step\_name).$(comment)
#$ queue
```

Note: See “Additional examples of building job command files” on page 441.
Chapter 13. 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

<table>
<thead>
<tr>
<th>Admin. File Keyword</th>
<th>Stanzas</th>
<th>Brief Description</th>
<th>For Details See:</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>User, Group</td>
<td>A list of account numbers available to a user submitting jobs.</td>
<td>“Step 2: Specify user stanzas” on page 350</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“Step 4: Specify group stanzas” on page 363</td>
</tr>
<tr>
<td>adapter_name</td>
<td>Adapter</td>
<td>Specifies the name the operating system uses to refer to an interface card installed on a node (such as en0).</td>
<td>“Step 5: Specify adapter stanzas” on page 365</td>
</tr>
<tr>
<td>adapter_stanzas</td>
<td>Machine</td>
<td>A list of adapter stanza names that define the adapters on a machine which can be requested.</td>
<td>“Step 1: Specify machine stanzas” on page 344</td>
</tr>
<tr>
<td>admin</td>
<td>Class, Group</td>
<td>A list of administrators for a group or class.</td>
<td>“Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“Step 4: Specify group stanzas” on page 363</td>
</tr>
<tr>
<td>alias</td>
<td>Machine</td>
<td>Lists one or more alias names to associate with the machine name.</td>
<td>“Step 1: Specify machine stanzas” on page 344</td>
</tr>
<tr>
<td>central_manager</td>
<td>Machine</td>
<td>When true, this designates the machine as the LoadLeveler central manager.</td>
<td>“Step 1: Specify machine stanzas” on page 344</td>
</tr>
<tr>
<td>ckpt_dir</td>
<td>Class</td>
<td>Specifies the directory to be used for checkpoint files for jobs that did not specify this directory in the job command file.</td>
<td>“Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td>ckpt_time_limit</td>
<td>Class</td>
<td>Specifies the hard limit, soft limit, or both limits for the elapsed time that checkpointing a job can take.</td>
<td>“Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“Limit keywords” on page 357</td>
</tr>
<tr>
<td>class_comment</td>
<td>Class</td>
<td>Text characterizing the class</td>
<td>“Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td>core_limit</td>
<td>Class</td>
<td>Specifies the hard limit, soft limit, or both limits for the size of a core file a job can create.</td>
<td>“Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“Limit keywords” on page 357</td>
</tr>
</tbody>
</table>
Table 13. Administration file keywords (continued)

<table>
<thead>
<tr>
<th>Admin. File Keyword</th>
<th>Stanzas</th>
<th>Brief Description</th>
<th>For Details See:</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu_limit</td>
<td>Class</td>
<td>Specifies the hard limit, soft limit, or both limits for the CPU time a job can use.</td>
<td>“Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“Limit keywords” on page 357</td>
</tr>
<tr>
<td>cpu_speed_scale</td>
<td>Machine</td>
<td>Determines whether CPU time is normalized according to machine speed.</td>
<td>“Step 1: Specify machine stanzas” on page 344</td>
</tr>
<tr>
<td>data_limit</td>
<td>Class</td>
<td>Specifies the hard limit, soft limit, or both limits for the size of a data segment a job can use.</td>
<td>“Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“Limit keywords” on page 357</td>
</tr>
<tr>
<td>default_class</td>
<td>User</td>
<td>A class name that is the default value assigned to jobs submitted by users for which no class statement appears.</td>
<td>“Step 2: Specify user stanzas” on page 350</td>
</tr>
<tr>
<td>default_group</td>
<td>User</td>
<td>A group name to which the user belongs.</td>
<td>“Step 2: Specify user stanzas” on page 350</td>
</tr>
<tr>
<td>default_interactive_class</td>
<td>User</td>
<td>A class to which interactive jobs are assigned for jobs submitted by users who do not specify a class using LOADL_INTERACTIVE_CLASS.</td>
<td>“Step 2: Specify user stanzas” on page 350</td>
</tr>
<tr>
<td>default_resources</td>
<td>Class</td>
<td>Specifies the default amount of resources consumed by a task of a job step, provided that no resources keyword is coded for the step in the job command file.</td>
<td>“Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td>exclude_groups</td>
<td>Class</td>
<td>A list of groups names identifying those who cannot submit jobs of a particular class.</td>
<td>“Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td>exclude_users</td>
<td>Class, Group</td>
<td>A list of user names identifying those who cannot submit jobs of a particular class or who are not members of the group.</td>
<td>“Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td>execution_factor</td>
<td>Class</td>
<td>Specifies that a job step of this type is preemptable.</td>
<td>“Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td>file_limit</td>
<td>Class</td>
<td>Specifies the hard limit, soft limit, or both limits for the size of a file that a job can create.</td>
<td>“Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“Limit keywords” on page 357</td>
</tr>
<tr>
<td>include_groups</td>
<td>Class</td>
<td>A list of groups names identifying those who can submit jobs of a particular class.</td>
<td>“Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td>include_users</td>
<td>Class, Group</td>
<td>A list of user names identifying those who can submit jobs of a particular class or who do belong to the group.</td>
<td>“Step 3: Specify class stanzas” on page 353</td>
</tr>
</tbody>
</table>
Table 13. Administration file keywords (continued)

<table>
<thead>
<tr>
<th>Admin. File Keyword</th>
<th>Stanzas</th>
<th>Brief Description</th>
<th>For Details See:</th>
</tr>
</thead>
<tbody>
<tr>
<td>interface_address</td>
<td>Adapter</td>
<td>Specifies the IP address by which the adapter is known to other nodes in the network.</td>
<td>&quot;Step 5: Specify adapter stanzas&quot; on page 365</td>
</tr>
<tr>
<td>interface_name</td>
<td>Adapter</td>
<td>Specifies the name by which the adapter is known to other nodes in the network.</td>
<td>&quot;Step 5: Specify adapter stanzas&quot; on page 365</td>
</tr>
<tr>
<td>job_cpu_limit</td>
<td>Class</td>
<td>Specifies the hard limit, soft limit, or both limits for the amount of CPU time an individual job step can use per processor.</td>
<td>&quot;Step 3: Specify class stanzas&quot; on page 353</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;Limit keywords” on page 357</td>
</tr>
<tr>
<td>machine_mode</td>
<td>Machine</td>
<td>Specifies the type of jobs this machine can run (batch, interactive, or both).</td>
<td>&quot;Step 1: Specify machine stanzas&quot; on page 344</td>
</tr>
<tr>
<td>master_node_exclusive</td>
<td>Machine</td>
<td>When true, this machine is used only as a master node for parallel jobs.</td>
<td>&quot;Step 1: Specify machine stanzas&quot; on page 344</td>
</tr>
<tr>
<td>master_node_requirement</td>
<td>Class</td>
<td>When true, jobs in this class have the requirement that they run on a master node having the master_node_exclusive setting.</td>
<td>&quot;Step 3: Specify class stanzas&quot; on page 353</td>
</tr>
<tr>
<td>maxidle</td>
<td>User, Group</td>
<td>Maximum number of idle job steps this user or group can have simultaneously.</td>
<td>&quot;Step 2: Specify user stanzas” on page 350</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;Step 4: Specify group stanzas” on page 363</td>
</tr>
<tr>
<td>maxjobs</td>
<td>User, Class, Group</td>
<td>Maximum number of job steps this user, class, or group can have running simultaneously.</td>
<td>&quot;Step 2: Specify user stanzas” on page 350</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;Step 4: Specify group stanzas” on page 363</td>
</tr>
<tr>
<td>max_jobs_scheduled</td>
<td>Machine</td>
<td>The maximum number of job steps that this machine can run.</td>
<td>&quot;Step 1: Specify machine stanzas” on page 344</td>
</tr>
<tr>
<td>max_node</td>
<td>User, Class, Group</td>
<td>The maximum number of nodes a user can request for a parallel job.</td>
<td>&quot;Step 2: Specify user stanzas” on page 350</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;Step 4: Specify group stanzas” on page 363</td>
</tr>
<tr>
<td>max_processors</td>
<td>User, Class, Group</td>
<td>The maximum number of machines a user can request for a parallel job.</td>
<td>&quot;Step 2: Specify user stanzas” on page 350</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;Step 4: Specify group stanzas” on page 363</td>
</tr>
<tr>
<td>max_protocol_instances</td>
<td>Class</td>
<td>The maximum number of instances on the network statement.</td>
<td>&quot;Step 3: Specify class stanzas” on page 353</td>
</tr>
</tbody>
</table>
### Administration file keywords

<table>
<thead>
<tr>
<th>Admin. File Keyword</th>
<th>Stanzas</th>
<th>Brief Description</th>
<th>For Details See:</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxqueued</td>
<td>User, Group</td>
<td>The maximum number of job steps a single group or user can have queued at the same time.</td>
<td>&quot;Step 2: Specify user stanzas” on page 350</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;Step 4: Specify group stanzas” on page 363</td>
</tr>
<tr>
<td>max_smp_tasks</td>
<td>Machine</td>
<td>Obsolete</td>
<td></td>
</tr>
<tr>
<td>max_total_tasks</td>
<td>User, Class, Group</td>
<td>Defines the maximum number of tasks allowed to run by the scheduler for jobs of a specified class, user, or group.</td>
<td>&quot;Step 2: Specify user stanzas” on page 350</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;Step 4: Specify group stanzas” on page 363</td>
</tr>
<tr>
<td>multilink_address</td>
<td>Adapter</td>
<td>Specifies the multilink address used for IP striping on the associated adapter.</td>
<td>&quot;Submitting jobs that use striping” on page 58</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;Customizing the administration file” on page 344</td>
</tr>
<tr>
<td>multilink_list</td>
<td>Adapter</td>
<td>Specifies the IP addresses of the adapters that this multilink device stripes across.</td>
<td>&quot;Submitting jobs that use striping” on page 58</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;Customizing the administration file” on page 344</td>
</tr>
<tr>
<td>name_server</td>
<td>Machine</td>
<td>A list of nameservers used for a machine.</td>
<td>&quot;Step 1: Specify machine stanzas” on page 344</td>
</tr>
<tr>
<td>network_type</td>
<td>Adapter</td>
<td>The type of network the adapter supports (for example, Ethernet). This is an administrator defined name.</td>
<td>&quot;Step 5: Specify adapter stanzas” on page 365</td>
</tr>
<tr>
<td>nice</td>
<td>Class</td>
<td>Increments the <em>nice</em> value of a job.</td>
<td>&quot;Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td>NQS_class</td>
<td>Class</td>
<td>When <strong>true</strong>, any job submitted to this class is routed to an NQS machine.</td>
<td>&quot;Step 3: Specify class stanzas” on page 353</td>
</tr>
</tbody>
</table>

**Linux notes**

- Do not use this keyword. NQS jobs are not supported by LoadLeveler for Linux.

---

128  Using and Administering LoadLeveler
### Table 13. Administration file keywords (continued)

<table>
<thead>
<tr>
<th>Admin. File Keyword</th>
<th>Stanzas</th>
<th>Brief Description</th>
<th>For Details See:</th>
</tr>
</thead>
<tbody>
<tr>
<td>NQS_query</td>
<td>Class</td>
<td>A list of queue names to use to monitor and cancel jobs.</td>
<td>[&quot;Step 3: Specify class stanzas&quot; on page 353]</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Linux notes</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Do not use this keyword. NQS jobs are not supported by LoadLeveler for Linux.</td>
<td></td>
</tr>
<tr>
<td>NQS_submit</td>
<td>Class</td>
<td>A name that identifies the name of the NQS pipe queue to which the job will be routed.</td>
<td>[&quot;Step 3: Specify class stanzas&quot; on page 353]</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Linux notes</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Do not use this keyword. NQS jobs are not supported by LoadLeveler for Linux.</td>
<td></td>
</tr>
<tr>
<td>pool_list</td>
<td>Machine</td>
<td>Specifies a list of pool numbers to which the machine belongs. Do not use negative numbers in a machine pool_list.</td>
<td>[&quot;Step 1: Specify machine stanzas&quot; on page 344]</td>
</tr>
<tr>
<td>priority</td>
<td>User, Class, Group</td>
<td>A number that identifies the priority of the appropriate user, class, or group.</td>
<td>[&quot;Step 2: Specify user stanzas&quot; on page 350]</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Step 3:</strong> Specify class stanzas on page 353</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Step 4:</strong> Specify group stanzas on page 363</td>
<td></td>
</tr>
<tr>
<td>pvm_root</td>
<td>Machine</td>
<td>A directory in which PVM 3.3 is installed.</td>
<td>[&quot;Step 1: Specify machine stanzas&quot; on page 344]</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Linux notes</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Do not use this keyword. PVM jobs are not supported by LoadLeveler for Linux.</td>
<td></td>
</tr>
<tr>
<td>resources</td>
<td>Machine</td>
<td>Specifies quantities of the consumable resources initially available on the machine.</td>
<td>[&quot;Step 1: Specify machine stanzas&quot; on page 344]</td>
</tr>
<tr>
<td>rss_limit</td>
<td>Class</td>
<td>Specifies the hard limit, soft limit, or both limits for the resident set size for a job.</td>
<td>[&quot;Step 3: Specify class stanzas&quot; on page 353]</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Limit keywords</strong> on page 357</td>
<td></td>
</tr>
</tbody>
</table>
### Administration file keywords (continued)

<table>
<thead>
<tr>
<th>Admin. File Keyword</th>
<th>Stanzas</th>
<th>Brief Description</th>
<th>For Details See:</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedd_fenced</td>
<td>Machine</td>
<td>When true, the central manager ignores connections from the schedd daemon running on this machine.</td>
<td>“Step 1: Specify machine stanzas” on page 344</td>
</tr>
<tr>
<td>schedd_host</td>
<td>Machine</td>
<td>When true, this machine is used to help submit-only machines access LoadLeveler hosts that run LoadLeveler jobs.</td>
<td>“Step 1: Specify machine stanzas” on page 344</td>
</tr>
<tr>
<td>spacct_excluse_enable</td>
<td>Machine</td>
<td>Specifies whether the SP accounting function is informed whenever this machine is being used exclusively by a particular job.</td>
<td>“Step 1: Specify machine stanzas” on page 344</td>
</tr>
<tr>
<td>speed</td>
<td>Machine</td>
<td>The weight associated with the machine for scheduling purposes.</td>
<td>“Step 1: Specify machine stanzas” on page 344</td>
</tr>
<tr>
<td>stack_limit</td>
<td>Class</td>
<td>Specifies the hard limit, soft limit, or both limits for the size of a stack.</td>
<td>“Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“Limit keywords” on page 357</td>
</tr>
<tr>
<td>submit_only</td>
<td>Machine</td>
<td>When true, designates this as a submit-only machine.</td>
<td>“Step 1: Specify machine stanzas” on page 344</td>
</tr>
<tr>
<td>switch_node_number</td>
<td>Adapter</td>
<td>The node on which the SP switch adapter is installed.</td>
<td>“Step 5: Specify adapter stanzas” on page 365</td>
</tr>
<tr>
<td>total_tasks</td>
<td>User, Class,</td>
<td>The maximum number of tasks a user can request for a parallel job.</td>
<td>“Step 2: Specify user stanzas” on page 350</td>
</tr>
<tr>
<td></td>
<td>Group</td>
<td></td>
<td>“Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“Step 4: Specify group stanzas” on page 363</td>
</tr>
<tr>
<td>type</td>
<td>All</td>
<td>The type of stanza.</td>
<td>“Administrating LoadLeveler” on page 70</td>
</tr>
<tr>
<td>wall_clock_limit</td>
<td>Class</td>
<td>Specifies the hard limit, soft limit, or both limits for the amount of elapsed time for which a job can run.</td>
<td>“Step 3: Specify class stanzas” on page 353</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“Limit keywords” on page 357</td>
</tr>
</tbody>
</table>

## 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.

### 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.
### Configuration file keywords

<table>
<thead>
<tr>
<th>Configuration file keyword</th>
<th>Brief description</th>
<th>For details</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCT</td>
<td>Turns the accounting function on or off.</td>
<td>“Step 9: Define job accounting” on page 384</td>
</tr>
<tr>
<td>ACCT_VALIDATION</td>
<td>The module called to perform account validation.</td>
<td>“Step 9: Define job accounting” on page 384</td>
</tr>
<tr>
<td>ACTION_ON_MAX_REJECT</td>
<td>Specifies whether a job is canceled or put in User Hold or System Hold status when the job exceeds the MAX_JOB_REJECT value.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>ACTION_ON_SWITCH_TABLE_ERROR</td>
<td>Points to an administrator supplied program that will be run when DRAIN_ON_SWITCH_TABLE_ERROR is set to true and a switch table unload error occurs.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>ADMIN_FILE</td>
<td>Points to the administration file containing user, class, and machine list stanzas.</td>
<td>“Step 11: Specify where files and directories are located” on page 386</td>
</tr>
<tr>
<td>AFS_GETNEWTOKEN</td>
<td>A filter which can be used to renew an AFS token.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>AGGREGATE_ADAPTERS</td>
<td>Allows the external scheduler to specify per-window adapter usages.</td>
<td>372</td>
</tr>
<tr>
<td>ARCH</td>
<td>The standard architecture of the system.</td>
<td>“Step 3: Define LoadLeveler machine characteristics” on page 374</td>
</tr>
<tr>
<td>BIN</td>
<td>The directory where LoadLeveler binaries are kept.</td>
<td>“Step 11: Specify where files and directories are located” on page 386</td>
</tr>
<tr>
<td>CENTRAL_MANAGER_HEARTBEAT_INTERVAL</td>
<td>The amount of time in seconds that defines how frequently primary and alternate central manager communicate with each other.</td>
<td>“Step 10: Specify alternate central managers” on page 385</td>
</tr>
<tr>
<td>CENTRAL_MANAGER_TIMEOUT</td>
<td>The number of heartbeat intervals that an alternate central manager will wait before declaring that the primary central manager is not operating.</td>
<td>“Step 10: Specify alternate central managers” on page 385</td>
</tr>
<tr>
<td>CKPT_CLEANUP_INTERVAL</td>
<td>The interval, in seconds, at which CKPT_CLEANUP_PROGRAM will be run.</td>
<td>“Remove old checkpoint files” on page 400</td>
</tr>
<tr>
<td>CKPT_CLEANUP_PROGRAM</td>
<td>An administrator provided program designed to delete old checkpoint files.</td>
<td>“Remove old checkpoint files” on page 400</td>
</tr>
<tr>
<td>CLASS</td>
<td>The class of jobs that can run on the machine.</td>
<td>“Step 3: Define LoadLeveler machine characteristics” on page 374</td>
</tr>
<tr>
<td>CLIENT_TIMEOUT</td>
<td>The maximum time, in seconds, that a daemon waits for a response over TCP/IP from a process.</td>
<td>“Step 13: Define network characteristics” on page 391</td>
</tr>
<tr>
<td>COLLECTOR_DGRAM_PORT</td>
<td>The port number used when connecting to a daemon.</td>
<td>“Step 13: Define network characteristics” on page 391</td>
</tr>
</tbody>
</table>
### Table 14. Configuration file keywords (continued)

<table>
<thead>
<tr>
<th>Configuration file keyword</th>
<th>Brief description</th>
<th>For details</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMM</td>
<td>Specifies a local directory where LoadLeveler keeps special files used for UNIX domain sockets for communicating among LoadLeveler daemons running on the same machine. This keyword allows the administrator to choose a different file system other than /tmp for these files.</td>
<td>&quot;Step 11: Specify where files and directories are located&quot; on page 386</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>Continue expression. Determines if a job should continue.</td>
<td>&quot;Step 8: Manage a job’s status using control expressions&quot; on page 383</td>
</tr>
<tr>
<td>CUSTOM_METRIC</td>
<td>A machine’s relative priority to run jobs. Negative values are not allowed.</td>
<td>&quot;Step 2: Define LoadLeveler cluster characteristics&quot; on page 369</td>
</tr>
<tr>
<td>CUSTOM_METRIC_COMMAND</td>
<td>An executable whose exit code value is assigned to CUSTOM_METRIC.</td>
<td>&quot;Step 2: Define LoadLeveler cluster characteristics&quot; on page 369</td>
</tr>
<tr>
<td>DCE_ADMIN_GROUP</td>
<td>Specifies the DCE group containing the DCE ids of those users who will have administrator authority for the current cluster.</td>
<td>&quot;Configuring LoadLeveler to use DCE security services&quot; on page 401</td>
</tr>
<tr>
<td>Linux notes</td>
<td>DCE security is not supported by LoadLeveler for Linux. This keyword is ignored.</td>
<td></td>
</tr>
<tr>
<td>DCE_AUTHENTICATION_PAIR</td>
<td>A pair of installation supplied programs that are used to authenticate DCE security credentials.</td>
<td>&quot;Step 17: Specify additional configuration file keywords&quot; on page 410</td>
</tr>
<tr>
<td>Linux notes</td>
<td>DCE security is not supported by LoadLeveler for Linux. Do not use this keyword.</td>
<td></td>
</tr>
<tr>
<td>Configuration file keyword</td>
<td>Brief description</td>
<td>For details</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DCE_ENABLEMENT</td>
<td>Activates the exploitation of DCE security. <strong>Note:</strong> This keyword is mutually exclusive with the SEC_ENABLEMENT keyword. Both keywords cannot be configured at the same time.</td>
<td>“Configuring LoadLeveler to use DCE security services” on page 401</td>
</tr>
<tr>
<td></td>
<td><strong>Linux notes</strong></td>
<td>DCE security is not supported by LoadLeveler for Linux. Do not use this keyword.</td>
</tr>
<tr>
<td>DCE_SERVICES_GROUP</td>
<td>Specifies the DCE group containing all of the principal names of the LoadLeveler daemons that are authorized to run in the current cluster.</td>
<td>“Configuring LoadLeveler to use DCE security services” on page 401</td>
</tr>
<tr>
<td></td>
<td><strong>Linux notes</strong></td>
<td>DCE security is not supported by LoadLeveler for Linux. This keyword is ignored</td>
</tr>
<tr>
<td>DRAIN_ON_SWITCH_TABLE_ERROR</td>
<td>Specifies that the <code>startd</code> should be drained when the switch table fails to unload.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>ENFORCE_RESOURCE_POLICY</td>
<td>Specifies how the resource policy for jobs should be enforced using the AIX Workload Manager.</td>
<td>“Step 4: Define consumable resources” on page 376</td>
</tr>
<tr>
<td>ENFORCE_RESOURCE_SUBMISSION</td>
<td>Indicates whether jobs submitted should be checked for the resources keyword.</td>
<td>“Step 4: Define consumable resources” on page 376</td>
</tr>
<tr>
<td>ENFORCE_RESOURCE_USAGE</td>
<td>Specifies that the resource usage for jobs needs to be enforced using the AIX Workload Manager.</td>
<td>“Step 4: Define consumable resources” on page 376</td>
</tr>
<tr>
<td></td>
<td><strong>Linux notes</strong></td>
<td>WLM enforcement is ignored by LoadLeveler for Linux.</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>The local directory to store the executable checkpoints of jobs submitted by other machines.</td>
<td>“Step 11: Specify where files and directories are located” on page 386</td>
</tr>
</tbody>
</table>
### Table 14. Configuration file keywords (continued)

<table>
<thead>
<tr>
<th>Configuration file keyword</th>
<th>Brief description</th>
<th>For details</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEATURE</td>
<td>A string specifying unique characteristics of a machine.</td>
<td>“Step 3: Define LoadLeveler machine characteristics” on page 374</td>
</tr>
<tr>
<td>FLOATING_RESOURCES</td>
<td>Specifies which consumable resources are available collectively on all of the machines in the LoadLeveler cluster.</td>
<td>“Step 4: Define consumable resources” on page 376</td>
</tr>
<tr>
<td>FS_INTERVAL</td>
<td>Defines the number of seconds used as the interval for checking free file system space.</td>
<td>“Setting up file system monitoring” on page 372</td>
</tr>
</tbody>
</table>
| FS_NOTIFY                  | Specifies the amount of free file system space (in blocks) at which mail is sent to the administrator.  
• If the amount of free space is less than the lower threshold value, then the notification indicates that there is a problem.  
• When the amount of free space becomes greater than the upper threshold value, the notification indicates that the problem has been resolved. | “Setting up file system monitoring” on page 372 |
| FS_SUSPEND                 | Specifies the amount of free file system space (in blocks) at which LoadLeveler is either drained or resumed.  
• If the amount of free space is less than the lower threshold value, then LoadLeveler is drained on the node.  
• 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. | “Setting up file system monitoring” on page 372 |
| FS_TERMINATE               | Specifies the amount of free file system space (in blocks) at which LoadLeveler is terminated.  
• If the amount of free space is less than the lower threshold value, then LoadLeveler is terminated on the node.  
• An upper threshold value is required for this keyword. However, since LoadLeveler has been terminated at the lower threshold, no action occurs. | “Setting up file system monitoring” on page 372 |
| GANG_MATRIX_BROADCAST_CYCLE| Obsolete keyword. | |
| GANG_MATRIX_NODE_SUBSET_SIZE| Obsolete keyword. | |
| GANG_MATRIX_REORG_CYCLE    | Obsolete keyword. | |
| GANG_MATRIX_TIME_SLICE     | Obsolete keyword. | |
| GLOBAL_HISTORY             | The directory containing the global history files. | “Step 9: Define job accounting” on page 384 |
Table 14. Configuration file keywords (continued)

<table>
<thead>
<tr>
<th>Configuration file keyword</th>
<th>Brief description</th>
<th>For details</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSMONITOR</td>
<td>Location of the gsmonitor executable (LoadL_GSmonitor).</td>
<td>“The gsmonitor daemon” on page 152</td>
</tr>
<tr>
<td></td>
<td><strong>Linux notes</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keyword ignored by LoadLeveler for Linux.</td>
<td></td>
</tr>
<tr>
<td>GSMONITOR_DOMAIN</td>
<td>Specifies which PSSP or peer domain the GSMONITOR daemon will execute on.</td>
<td>“The gsmonitor daemon” on page 152</td>
</tr>
<tr>
<td></td>
<td><strong>Linux notes</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keyword ignored by LoadLeveler for Linux.</td>
<td></td>
</tr>
<tr>
<td>GSMONITOR_RUNS_HERE</td>
<td>When true, specifies that you want to start the gsmonitor daemon (you must have PSSP Groups Service).</td>
<td>“The gsmonitor daemon” on page 152</td>
</tr>
<tr>
<td></td>
<td><strong>Linux notes</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keyword ignored by LoadLeveler for Linux.</td>
<td></td>
</tr>
<tr>
<td>HIERARCHICAL_FANOUT</td>
<td>The number of children a hierarchical message is distributed to at each level of the hierarchy.</td>
<td>Chapter 18, “Preemption using the Gang scheduler,” on page 421</td>
</tr>
<tr>
<td>HISTORY</td>
<td>The pathname of the history file for local LoadLeveler jobs.</td>
<td>“Step 11: Specify where files and directories are located” on page 386</td>
</tr>
<tr>
<td>HISTORY_PERMISSION</td>
<td>Specifies the permissions associated with the history file of a LoadL_schedd daemon.</td>
<td>Chapter 10, “Gathering job accounting data,” on page 83</td>
</tr>
<tr>
<td>JOB_ACCT_Q_POLICY</td>
<td>The amount of time in seconds that determines how often the startd daemon updates the schedd daemon with accounting data of running jobs.</td>
<td>Chapter 10, “Gathering job accounting data,” on page 83</td>
</tr>
<tr>
<td>JOB_EPILOG</td>
<td>Pathname of the epilog program.</td>
<td>“Writing prolog and epilog programs” on page 319</td>
</tr>
<tr>
<td>JOB_LIMIT_POLICY</td>
<td>The amount of time in seconds that LoadLeveler checks to see if job_cpu_limit has been exceeded.</td>
<td>Chapter 10, “Gathering job accounting data,” on page 83</td>
</tr>
<tr>
<td>JOB_PROLOG</td>
<td>Pathname of the prolog program.</td>
<td>“Writing prolog and epilog programs” on page 319</td>
</tr>
</tbody>
</table>
## Configuration file keywords

<table>
<thead>
<tr>
<th>Configuration file keyword</th>
<th>Brief description</th>
<th>For details</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB_USER_EPILOG</td>
<td>Pathname of the user epilog program.</td>
<td>“Writing prolog and epilog programs” on page 319</td>
</tr>
<tr>
<td>JOB_USER_PROLOG</td>
<td>Pathname of the user prolog program.</td>
<td>“Writing prolog and epilog programs” on page 319</td>
</tr>
<tr>
<td>KBDD</td>
<td>Location of kbdd executable (LoadL_Kbdd).</td>
<td>“LoadLeveler daemons” on page 8</td>
</tr>
<tr>
<td>KILL</td>
<td>Kill expression. Determines if vacated jobs should be sent the SIGKILL signal.</td>
<td>“Step 8: Manage a job’s status using control expressions” on page 383</td>
</tr>
<tr>
<td>LIB</td>
<td>The directory where LoadLeveler libraries are kept.</td>
<td>“Step 11: Specify where files and directories are located” on page 386</td>
</tr>
<tr>
<td>LOADL_ADMIN</td>
<td>List of LoadLeveler administrators.</td>
<td>“Step 1: Define LoadLeveler administrators” on page 368</td>
</tr>
<tr>
<td>LOCAL_CONFIG</td>
<td>Pathname of the optional local configuration file containing information specific to a node in the LoadLeveler network.</td>
<td>“Step 11: Specify where files and directories are located” on page 386</td>
</tr>
<tr>
<td>LOG</td>
<td>Local directory for storing log files.</td>
<td>“Step 11: Specify where files and directories are located” on page 386</td>
</tr>
<tr>
<td>MACHINE_AUTHENTICATE</td>
<td>Specifies whether machine validation is performed.</td>
<td>“Step 2: Define LoadLeveler cluster characteristics” on page 369</td>
</tr>
<tr>
<td>MACHINE_UPDATE_INTERVAL</td>
<td>The time, in seconds, during which machines must report to the central manager.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>MACHPRIO</td>
<td>Machine priority expression</td>
<td>“Step 7: Prioritize the order of executing machines maintained by the negotiator” on page 380</td>
</tr>
<tr>
<td>MAIL</td>
<td>Name of a local mail program used to override default mail notification.</td>
<td>“Using your own mail program” on page 318</td>
</tr>
<tr>
<td>MASTER</td>
<td>Location of the master executable (LoadL_master).</td>
<td>“LoadLeveler daemons” on page 8</td>
</tr>
<tr>
<td>MASTER_DGRAM_PORT</td>
<td>The port number used when connecting to the daemon.</td>
<td>“Step 13: Define network characteristics” on page 391</td>
</tr>
<tr>
<td>MASTER_STREAM_PORT</td>
<td>The port number to used when connecting to the daemon.</td>
<td>“Step 13: Define network characteristics” on page 391</td>
</tr>
<tr>
<td>MAX_CKPT_INTERVAL</td>
<td>The maximum number of seconds between checkpoints for running jobs.</td>
<td>“Step 14: Enable checkpointing” on page 392</td>
</tr>
<tr>
<td>MAX_JOB_REJECT</td>
<td>The number of times a job is rejected before it is canceled or put in User Hold or System Hold status.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>MAX_STARTERS</td>
<td>The maximum number of jobs that can run simultaneously.</td>
<td>“Step 5: Specify how many jobs a machine can run” on page 378</td>
</tr>
<tr>
<td>MIN_CKPT_INTERVAL</td>
<td>The minimum number of seconds between checkpoints for running jobs.</td>
<td>“Step 14: Enable checkpointing” on page 392</td>
</tr>
<tr>
<td>Configuration file keyword</td>
<td>Brief description</td>
<td>For details</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>NEGOTIATOR</td>
<td>Location of the negotiator executable (LoadL_negotiator).</td>
<td>“<a href="#">LoadLeveler daemons” on page 8</a></td>
</tr>
<tr>
<td>NEGOTIATOR_CYCLE_DELAY</td>
<td>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.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>NEGOTIATOR_CYCLE_TIME_LIMIT</td>
<td>The maximum amount of time, in seconds, which the negotiator daemon will be allowed to spend in one cycle trying to schedule jobs.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>NEGOTIATOR_INTERVAL</td>
<td>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.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>NEGOTIATOR_LOADAVG_INCREMENT</td>
<td>The factor added to the startd machine's load average to compensate for the increased load caused by starting another machine.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>NEGOTIATOR_PARALLEL_DEFER</td>
<td>The length of time that a job is given to accumulate processors.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>NEGOTIATOR_PARALLEL_HOLD</td>
<td>The length of time a job attempts to collect machines before releasing them.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL</td>
<td>The amount of time in seconds between calculation of the SYSPRIO values for waiting jobs.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>NEGOTIATOR_REJECT_DEFER</td>
<td>The amount of time in seconds the negotiator waits before it considers scheduling a job to a machine that recently rejected the job.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>NEGOTIATOR_REMOVE_COMPLETED</td>
<td>The amount of time the negotiator keeps information on completed and removed jobs.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>NEGOTIATOR_RESCAN_QUEUE</td>
<td>The amount of time the negotiator waits to rescan the job queue for machines that temporarily have non-runnable jobs.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>NEGOTIATOR_STREAM_PORT</td>
<td>The port number used when connecting to the daemon.</td>
<td>“Step 13: Define network characteristics” on page 391</td>
</tr>
</tbody>
</table>
### Configuration file keywords

<table>
<thead>
<tr>
<th>Configuration file keyword</th>
<th>Brief description</th>
<th>For details</th>
</tr>
</thead>
<tbody>
<tr>
<td>NQS_DIR</td>
<td>The directory where NQS commands reside.</td>
<td>“Step 11: Specify where files and directories are located” on page 386</td>
</tr>
<tr>
<td></td>
<td><strong>Linux notes</strong></td>
<td>Do not use this keyword. NQS is not supported by LoadLeveler for Linux.</td>
</tr>
<tr>
<td>OBITUARY_LOG_LENGTH</td>
<td>The number of lines from the need of the file that are appended to the Master_Log.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>POLLING_FREQUENCY</td>
<td>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.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>POLLS_PER_UPDATE</td>
<td>The frequency, in POLLING_FREQUENCY intervals, with which the startd daemon updates the central manager.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>PREEMPT_CLASS [class]</td>
<td>Specifies the preemption rule for a job class.</td>
<td>Chapter 18, “Preemption using the Gang scheduler,” on page 421</td>
</tr>
<tr>
<td></td>
<td><strong>Linux notes</strong></td>
<td>Do not use this keyword. Preemption is not supported by LoadLeveler for Linux.</td>
</tr>
<tr>
<td>PREEMPTION_SUPPORT</td>
<td>Specifies whether preemption is enabled for a cluster.</td>
<td>“Step 2: Define LoadLeveler cluster characteristics” on page 369</td>
</tr>
<tr>
<td>PROCESS_TRACKING</td>
<td>When true ensures that when a job is terminated, no processes created by the job will continue running.</td>
<td>“Step 15: Specify process tracking” on page 400</td>
</tr>
<tr>
<td></td>
<td><strong>Linux notes</strong></td>
<td>Do not use this keyword. Process tracking is ignored by LoadLeveler for Linux.</td>
</tr>
</tbody>
</table>
### Configuration file keywords (continued)

<table>
<thead>
<tr>
<th>Configuration file keyword</th>
<th>Brief description</th>
<th>For details</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS_TRACKING_EXTENSION</td>
<td>The directory containing the kernel extension binary <code>LoadL_pt_ke</code>.</td>
<td><em>Step 15: Specify process tracking</em> on page 400</td>
</tr>
<tr>
<td></td>
<td><strong>Linux notes</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Do not use this keyword.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Process tracking is ignored by LoadLeveler for Linux.</td>
<td></td>
</tr>
<tr>
<td>PUBLISH_OBITUARIES</td>
<td>When <code>true</code>, specifies that the master daemon sends mail to the administrator(s) when any daemon it manages dies abnormally.</td>
<td><em>Step 17: Specify additional configuration file keywords</em> on page 410</td>
</tr>
<tr>
<td>REJECT_ON_RESTRICTED_LOGIN</td>
<td>When true, specifies that the AIX <code>loginrestrictions</code> checking will be performed on every node where the job is scheduled to run.</td>
<td><em>Step 17: Specify additional configuration file keywords</em> on page 410</td>
</tr>
<tr>
<td></td>
<td><strong>Linux notes</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Do not use this keyword.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Login restriction checking is ignored by LoadLeveler for Linux.</td>
<td></td>
</tr>
<tr>
<td>RELEASEDIR</td>
<td>The directory where all the LoadLeveler software resides.</td>
<td><em>Step 11: Specify where files and directories are located</em> on page 386</td>
</tr>
<tr>
<td>RESOURCES</td>
<td>Specifies quantities of the consumable resources “consumed” by each task of a job step.</td>
<td><em>Step 4: Define consumable resources</em> on page 376</td>
</tr>
<tr>
<td>RESTARTS_PER_HOUR</td>
<td>The number of times the master daemon attempts to restart a daemon that dies abnormally.</td>
<td><em>Step 17: Specify additional configuration file keywords</em> on page 410</td>
</tr>
<tr>
<td>RESUME_ON_SWITCH_TABLE_ERROR_CLEAR</td>
<td>If set to <code>true</code> specifies that the <code>startd</code> that was drained when the switch table failed to unload will automatically resume once the unload errors are cleared.</td>
<td><em>Step 17: Specify additional configuration file keywords</em> on page 410</td>
</tr>
<tr>
<td>SAVELOGS</td>
<td>Specifies the directory in which log files are archived.</td>
<td><em>Step 12: Record and control log files</em> on page 387</td>
</tr>
<tr>
<td>SCHEDD</td>
<td>Location of the <code>schedd</code> executable (<code>LoadL_schedd</code>).</td>
<td><em>LoadLeveler daemons</em> on page 8</td>
</tr>
<tr>
<td>SCHEDD_INTERVAL</td>
<td>Specifies the interval, in seconds, at which the <code>schedd</code> daemon checks the local job queue.</td>
<td><em>Step 17: Specify additional configuration file keywords</em> on page 410</td>
</tr>
<tr>
<td>SCHEDD_RUNS_HERE</td>
<td>Specifies whether this daemon will run on the host.</td>
<td><em>Step 3: Define LoadLeveler machine characteristics</em> on page 374</td>
</tr>
</tbody>
</table>
### Configuration file keywords

**Table 14. Configuration file keywords (continued)**

<table>
<thead>
<tr>
<th>Configuration file keyword</th>
<th>Brief description</th>
<th>For details</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEDD_SUBMIT_AFFINITY</td>
<td>Specifies whether the IlsSubmit command submits a job to the machine where the command was invoked provided the schedd daemon is running on the machine.</td>
<td>“Step 3: Define LoadLeveler machine characteristics” on page 374</td>
</tr>
<tr>
<td>SCHEDD_STREAM_PORT</td>
<td>The port number used when connecting to the daemon.</td>
<td>“Step 13: Define network characteristics” on page 391</td>
</tr>
<tr>
<td>SCHEDULE_BY_RESOURCES</td>
<td>Specifies which consumable resources are considered by the LoadLeveler schedulers.</td>
<td>“Step 4: Define consumable resources” on page 376</td>
</tr>
<tr>
<td>SCHEDULER_API</td>
<td>When YES, disables the native LoadLeveler scheduling algorithm. Note: This keyword is obsolete and should be replaced by SCHEDULER_TYPE = API.</td>
<td>“Step 2: Define LoadLeveler cluster characteristics” on page 369</td>
</tr>
<tr>
<td>SCHEDULER_TYPE</td>
<td>Specifies the LoadLeveler scheduling algorithm:</td>
<td>“Step 2: Define LoadLeveler cluster characteristics” on page 369</td>
</tr>
<tr>
<td></td>
<td>• LL_DEFAULT</td>
<td>Additional information on Gang scheduling can be found in Chapter 18.</td>
</tr>
<tr>
<td></td>
<td>• BACKFILL</td>
<td>“Preemption using the Gang scheduler,” on page 421.</td>
</tr>
<tr>
<td></td>
<td>• API</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• GANG</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Linux notes</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GANG scheduling is not supported by LoadLeveler for Linux.</td>
<td></td>
</tr>
<tr>
<td>SEC_ADMIN_GROUP</td>
<td>Sets the group name which contains the LoadLeveler administrators when security services are enabled.</td>
<td>“Configuring LoadLeveler to use Cluster Security Services” on page 407</td>
</tr>
<tr>
<td></td>
<td><strong>Linux notes</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Neither DCE nor CtSec security are supported on LoadLeveler for Linux. This keyword is ignored.</td>
<td></td>
</tr>
</tbody>
</table>
Table 14. Configuration file keywords (continued)

<table>
<thead>
<tr>
<th>Configuration file keyword</th>
<th>Brief description</th>
<th>For details</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC_ENABLEMENT</td>
<td>Specifies the security mechanism to be used:</td>
<td>&quot;Configuring LoadLeveler to use Cluster Security Services&quot; on page 407</td>
</tr>
<tr>
<td></td>
<td>• COMPAT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DCE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• CTSEC</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Linux notes</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Do not set this keyword to DCE or CtSec in the configuration file for a Linux machine. Neither DCE nor CtSec security are supported on LoadLeveler for Linux.</td>
<td></td>
</tr>
<tr>
<td>SEC_SERVICES_GROUP</td>
<td>The name of the LoadLeveler services group when security services are enabled.</td>
<td>&quot;Configuring LoadLeveler to use Cluster Security Services&quot; on page 407</td>
</tr>
<tr>
<td></td>
<td><strong>Linux notes</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Neither DCE nor CtSec security are supported on LoadLeveler for Linux. This keyword is ignored.</td>
<td></td>
</tr>
<tr>
<td>SEC_IMPOSED_MECHS</td>
<td>A list of LoadLeveler’s security mechanisms permitted when cluster security services are enabled.</td>
<td>&quot;Limiting which security mechanisms LoadLeveler can use&quot; on page 410</td>
</tr>
<tr>
<td></td>
<td><strong>Linux notes</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Neither DCE nor CtSec security are supported on LoadLeveler for Linux. This keyword is ignored.</td>
<td></td>
</tr>
<tr>
<td>SPOOL</td>
<td>The local directory where LoadLeveler keeps the local job queue and checkpoint files.</td>
<td>&quot;Step 11: Specify where files and directories are located&quot; on page 386</td>
</tr>
<tr>
<td>START</td>
<td>Start expression. Determines if a machine can run a job.</td>
<td>&quot;Step 8: Manage a job’s status using control expressions&quot; on page 383</td>
</tr>
<tr>
<td>STARTD</td>
<td>Location of the startd executable (LoadL_startd).</td>
<td>&quot;LoadLeveler daemons&quot; on page 8</td>
</tr>
<tr>
<td>STARTER</td>
<td>Location of the starter executable (LoadL_starter).</td>
<td>&quot;LoadLeveler daemons&quot; on page 8</td>
</tr>
</tbody>
</table>
### Configuration file keywords

<table>
<thead>
<tr>
<th>Configuration file keyword</th>
<th>Brief description</th>
<th>For details</th>
</tr>
</thead>
<tbody>
<tr>
<td>START_CLASS [class]</td>
<td>Specifies the job starting rule for a job class.</td>
<td>Chapter 18, “Preemption using the Gang scheduler,” on page 421</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Linux notes</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Do not use this keyword. The GANG scheduler is not supported by LoadLeveler for Linux.</td>
</tr>
<tr>
<td>STARTD_RUNS_HERE</td>
<td>Specifies whether this daemon will run on the host.</td>
<td>“Step 3: Define LoadLeveler machine characteristics” on page 374</td>
</tr>
<tr>
<td>START_DAEMONS</td>
<td>Specifies whether to start the daemons on the machine.</td>
<td>“Step 3: Define LoadLeveler machine characteristics” on page 374</td>
</tr>
<tr>
<td>STARTD_DGRAM_PORT</td>
<td>The port number used when connecting to the daemon.</td>
<td>“Step 13: Define network characteristics” on page 391</td>
</tr>
<tr>
<td>STARTD_STREAM_PORT</td>
<td>The port number used when connecting to the daemon.</td>
<td>“Step 13: Define network characteristics” on page 391</td>
</tr>
<tr>
<td>SUBMIT_FILTER</td>
<td>The program you want to run to filter a job script when the job is submitted.</td>
<td>“Filtering a job script” on page 318</td>
</tr>
<tr>
<td>SUSPEND</td>
<td>Suspend expression. Determines if a job should be suspended.</td>
<td>“Step 8: Manage a job’s status using control expressions” on page 383</td>
</tr>
<tr>
<td>SYSPRIO</td>
<td>System priority expression.</td>
<td>“Step 6: Prioritize the queue maintained by the negotiator” on page 378</td>
</tr>
<tr>
<td>TRUNC_GSMONITOR_LOG_ON_OPEN</td>
<td>When true, specifies that the log file is restarted with every invocation of the daemon.</td>
<td>“Step 12: Record and control log files” on page 387</td>
</tr>
<tr>
<td>TRUNC_KBDD_LOG_ON_OPEN</td>
<td>When true, specifies the log file is restarted with every invocation of the daemon.</td>
<td>“Step 12: Record and control log files” on page 387</td>
</tr>
<tr>
<td>TRUNC_MASTER_LOG_ON_OPEN</td>
<td>When true, specifies the log file is restarted with every invocation of the daemon.</td>
<td>“Step 12: Record and control log files” on page 387</td>
</tr>
<tr>
<td>TRUNC_NEGOTIATOR_LOG_ON_OPEN</td>
<td>When true, specifies the log file is restarted with every invocation of the daemon.</td>
<td>“Step 12: Record and control log files” on page 387</td>
</tr>
<tr>
<td>TRUNC_SCHEDD_LOG_ON_OPEN</td>
<td>When true, specifies the log file is restarted with every invocation of the daemon.</td>
<td>“Step 12: Record and control log files” on page 387</td>
</tr>
<tr>
<td>TRUNC_STARTD_LOG_ON_OPEN</td>
<td>When true, specifies the log file is restarted with every invocation of the daemon.</td>
<td>“Step 12: Record and control log files” on page 387</td>
</tr>
<tr>
<td>TRUNC_STARTER_LOG_ON_OPEN</td>
<td>When true, specifies the log file is restarted with every invocation of the daemon.</td>
<td>“Step 12: Record and control log files” on page 387</td>
</tr>
</tbody>
</table>
Table 14. Configuration file keywords (continued)

<table>
<thead>
<tr>
<th>Configuration file keyword</th>
<th>Brief description</th>
<th>For details</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATE_ON_POLL_INTERVAL_ONLY</td>
<td>When true, specifies that the startd daemons send machine update information to the Central Manager only at poll intervals.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>VACATE</td>
<td>The vacate expression. Determines whether suspended jobs should be vacated.</td>
<td>“Step 8: Manage a job’s status using control expressions” on page 383</td>
</tr>
<tr>
<td>VM_IMAGE_ALGORITHM</td>
<td>Virtual memory algorithm. Used for checking the image_size requirement.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>WALLCLOCK_ENFORCE</td>
<td>When 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.</td>
<td>“Step 17: Specify additional configuration file keywords” on page 410</td>
</tr>
<tr>
<td>X_RUNS_HERE</td>
<td>When true, specifies that you want to start the keyboard daemon.</td>
<td>“Step 3: Define LoadLeveler machine characteristics” on page 374</td>
</tr>
</tbody>
</table>

User-defined keywords

The following table serves only as a reference. These keywords are described in more detail in “User-defined variables” on page 73.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackgroundLoad</td>
<td>Defines the variable BackgroundLoad and assigns to it a floating point constant. This might be used as a noise factor indicating no activity.</td>
</tr>
<tr>
<td>CPU_Busy</td>
<td>Defines the variable CPU_Busy and reassigns to it at each evaluation the Boolean value True or False, depending on whether the Berkeley one-minute load average is equal to or greater than the saturation level of 1.5.</td>
</tr>
<tr>
<td>CPU_Idle</td>
<td>Defines the variable CPU_Idle and reassigns to it at each evaluation the Boolean value True or False, depending on whether the Berkeley one-minute load average is equal to or less than 0.7.</td>
</tr>
<tr>
<td>HighLoad</td>
<td>Is a keyword that the user can define to use as a saturation level at which no further jobs should be started.</td>
</tr>
<tr>
<td>HOUR</td>
<td>Defines the variable HOUR and assigns to it a constant integer value.</td>
</tr>
<tr>
<td>JobLoad</td>
<td>Defines the variable JobLoad which defines the load on the machine caused by running the job.</td>
</tr>
<tr>
<td>KeyboardBusy</td>
<td>Defines the variable KeyboardBusy and reassigns to it at each evaluation the Boolean value True or False, depending on whether the keyboard and mouse have been idle for fifteen minutes.</td>
</tr>
<tr>
<td>LowLoad</td>
<td>Defines the variable LowLoad and assigns to it the value of BackgroundLoad. This might be used as a restart level at which jobs can be started again and assumes only running 1 job on the machine.</td>
</tr>
<tr>
<td>mail</td>
<td>Specifies a local program you want to use in place of the LoadLeveler default mail notification method.</td>
</tr>
<tr>
<td>MINUTE</td>
<td>Defines the variable MINUTE and assigns to it a constant integer value.</td>
</tr>
<tr>
<td>StateTimer</td>
<td>Defines the variable StateTimer and reassigns to it at each evaluation the number of seconds since the current state was entered.</td>
</tr>
</tbody>
</table>
## 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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Brief Description</th>
<th>For Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arch</td>
<td>Standard architecture of the system.</td>
<td>&quot;LoadLeveler variables&quot; on page 74</td>
</tr>
<tr>
<td>ClassSysprio</td>
<td>Job priority for the class.</td>
<td>&quot;Step 6: Prioritize the queue maintained by the negotiator” on page 378</td>
</tr>
<tr>
<td>Connectivity</td>
<td>The ratio of usable switch adapters to total switch adapters on a machine.</td>
<td>&quot;LoadLeveler variables&quot; on page 74</td>
</tr>
<tr>
<td>ConsumableCpus</td>
<td>Number of ConsumableCpus currently available on the machine, if defined in SCHEDULE_BY_RESOURCES. If not, then it is the same as Cpus.</td>
<td>&quot;LoadLeveler variables&quot; on page 74</td>
</tr>
<tr>
<td>ConsumableMemory</td>
<td>Amount of ConsumableMemory currently available on the machine, if defined in SCHEDULE_BY_RESOURCES. If not, then it is the same as Memory.</td>
<td>&quot;LoadLeveler variables&quot; on page 74</td>
</tr>
<tr>
<td>ConsumableVirtualMemory</td>
<td>Amount of ConsumableVirtualMemory currently available on the machine, if defined in SCHEDULE_BY_RESOURCES. If not, then it is the same as VirtualMemory.</td>
<td>&quot;LoadLeveler variables&quot; on page 74</td>
</tr>
<tr>
<td>Cpus</td>
<td>Number of CPU’s installed.</td>
<td>&quot;LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>CurrentTime</td>
<td>The UNIX date that includes the current system time, in seconds, since January 1, 1970.</td>
<td>&quot;LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>CustomMetric</td>
<td>The relative machine priority.</td>
<td>&quot;LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>Disk</td>
<td>Free disk in megabytes on the file system where checkpoints are stored.</td>
<td>&quot;LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>domain or domainname</td>
<td>Dynamically indicates the domain name of the current host machine where the program is running.</td>
<td>&quot;LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>EnteredCurrentState</td>
<td>Value of CurrentTime when the current state was entered.</td>
<td>&quot;LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>FreeRealMemory</td>
<td>The amount of free real memory in megabytes on the machine.</td>
<td>&quot;LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>GroupQueuedJobs</td>
<td>The number of jobs either running or queued for the LoadLeveler group.</td>
<td>&quot;Step 6: Prioritize the queue maintained by the negotiator” on page 378</td>
</tr>
<tr>
<td>GroupRunningJobs</td>
<td>The number of jobs currently running for the LoadLeveler group.</td>
<td>&quot;Step 6: Prioritize the queue maintained by the negotiator” on page 378</td>
</tr>
<tr>
<td>GroupSysprio</td>
<td>The job priority for the group.</td>
<td>&quot;Step 6: Prioritize the queue maintained by the negotiator” on page 378</td>
</tr>
<tr>
<td>GroupTotalJobs</td>
<td>The total number of jobs associated with the LoadLeveler group.</td>
<td>&quot;Step 6: Prioritize the queue maintained by the negotiator” on page 378</td>
</tr>
<tr>
<td>Variable</td>
<td>Brief Description</td>
<td>For Details</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>host or hostname</td>
<td>Dynamically indicates the name of the host machine where the program is running.</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>KeyboardIdle</td>
<td>Number of seconds since the keyboard or mouse was last used.</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>LoadAvg</td>
<td>Berkeley one-minute load average.</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>Machine</td>
<td>Name of the current machine.</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>MasterMachPrio</td>
<td>A value that is 1 for master nodes and is 0 otherwise.</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>Memory</td>
<td>Physical memory installed on the machine in megabytes.</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>OpSys</td>
<td>Indicates the operating system on the host where the program is running.</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>PagesFreed</td>
<td>The number of pages freed per second.</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>PagesScanned</td>
<td>The number of pages scanned per second.</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>QDate</td>
<td>Difference in seconds between when the negotiator starts up and when the job is submitted.</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>Speed</td>
<td>The relative machine speed.</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>State</td>
<td>State of the startd. Can be None, Busy, Running, Idle, Suspend, Flush, or Drain.</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>tilde</td>
<td>Dynamically defines the pathname of the LoadLeveler home directory.</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>tm_hour</td>
<td>Number of hours since midnight (0-23).</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>tm_isdst</td>
<td>Daylight Savings Time flag: positive when in effect, zero when not in effect, negative when information is unavailable.</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>tm_mday</td>
<td>Number of the day of the month (1-31).</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>tm_min</td>
<td>Number of minutes after the hour (0-59).</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>tm_mon</td>
<td>Number of months since January (0-11).</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>tm_sec</td>
<td>Number of seconds after the minute (0-59).</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>tm_wday</td>
<td>Number of days since Sunday (0-6).</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>tm_yday</td>
<td>Number of days since January 1 (0-365).</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>tm_year</td>
<td>Number of years since 1900 (0-9999).</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>tm4_year</td>
<td>The four-digit integer representation of the current year.</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
<tr>
<td>UserPrio</td>
<td>User defined priority of a job.</td>
<td>“Step 6: Prioritize the queue maintained by the negotiator” on page 378</td>
</tr>
<tr>
<td>UserQueuedJobs</td>
<td>The number of jobs either running or queued for the user.</td>
<td>“Step 6: Prioritize the queue maintained by the negotiator” on page 378</td>
</tr>
<tr>
<td>UserRunningJobs</td>
<td>The number of jobs currently running for the user.</td>
<td>“Step 6: Prioritize the queue maintained by the negotiator” on page 378</td>
</tr>
<tr>
<td>UserSysprio</td>
<td>The priority of the user who submitted the job.</td>
<td>“Step 6: Prioritize the queue maintained by the negotiator” on page 378</td>
</tr>
</tbody>
</table>
## Configuration file keywords

<table>
<thead>
<tr>
<th>Variable</th>
<th>Brief Description</th>
<th>For Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserTotalJobs</td>
<td>The total number of jobs associated with this user.</td>
<td>“Step 6: Prioritize the queue maintained by the negotiator” on page 378</td>
</tr>
<tr>
<td>VirtualMemory</td>
<td>The size of the available swap space (free paging space) on the machine in kilobytes.</td>
<td>“LoadLeveler variables” on page 74</td>
</tr>
</tbody>
</table>
Chapter 14. LoadLeveler daemons and job states

Daemons

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

The master daemon

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

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

Linux notes

The master daemon will not start on a Linux machine if
DCE_ENABLEMENT is TRUE, SEC_ENABLEMENT is set to DCE or
CTSEC, SCHEDULER_TYPE is set to GANG, NQS_DIR is specified, or the
keyword pvm_root is specified in the machine stanza of the LoadLeveler
administration file. If the master daemon does not start, no other daemons
will start.

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

The master daemon starts and if necessary, restarts all the LoadLeveler daemons
that the machine it resides on is configured to run. As part of its startup procedure,
this daemon executes the .Ilrc 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.
Daemons

The master daemon may perform the following actions in response to an \texttt{l1ctl} command:

- Kill all daemons and exit
- Kill all daemons and execute a new master
- Re-run the .\texttt{llrc} file, reread the configuration files, stop or start daemons as appropriate for the new configuration files
- Send drain request to startd and \texttt{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 \texttt{schedd}, and send result to caller

The \texttt{schedd} daemon

The \texttt{schedd} daemon receives jobs sent by the \texttt{llsubmit} command and manages those jobs to machines selected by the negotiator daemon. The \texttt{schedd} daemon is started, restarted, signalled, and stopped by the master daemon.

The \texttt{schedd} daemon can be in any one of the following activity states:

\textbf{Available}

This machine is available to schedule jobs.

\textbf{Drained}

The \texttt{schedd} machine accepts no more jobs. There are no jobs in starting or running state. Jobs in the Idle state are drained, meaning they will not get dispatched.

\textbf{Draining}

The \texttt{schedd} daemon is being drained by the administrator but some jobs are still running. The state of the machine remains Draining until all running jobs complete. At that time, the machine status changes to Drained.

\textbf{Down}

The daemon is not running on this machine. The \texttt{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 \texttt{schedd} daemon performs the following functions:

- Assigns new job ids when requested by the job submission process (for example, by the \texttt{llsubmit} command).
- Receives new jobs from the \texttt{llsubmit} command. A new job is received as a \textit{job object} for each job step. A job object is the data structure in memory containing all the information about a job step. The \texttt{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 an interactive 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.

Linux notes

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

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

**Busy** The maximum number of jobs are running on this machine as specified by the MAX_STARTERS configuration keyword.

**Down** The daemon is not running on this machine. The startd daemon enters this state when it has not reported its status to the negotiator. This can occur when the machine is actually down, or because there is a network failure.

**Drained** The startd machine will not accept any new jobs. No jobs are running when startd is in the drained state.

**Draining** The startd daemon is being drained by the administrator, but some jobs are still running. The machine remains in the draining state until all of the running jobs have completed, at which time the machine status changes to drained. The startd daemon will not accept any new jobs while in the draining state.

**Flush** Any running jobs have been vacated (terminated and returned to the queue to be redispached). The startd daemon will not accept any new jobs.

**Idle** The machine is not running any jobs.

**None** LoadLeveler is running on this machine, but no jobs can run here.

**Running** The machine is running one or more jobs and is capable of running more.

**Suspend** All LoadLeveler jobs running on this machine are stopped (cease processing), but remain in virtual memory. The startd daemon will not accept any new jobs.

The startd daemon performs these functions:
Daemons

- 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 74.
  - 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 383.
- Receives job requests from the schedd daemon to:
  - Start a job
  - 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:

<table>
<thead>
<tr>
<th>If:</th>
<th>Then this happens:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes, it can start a new job</td>
<td>The startd forks a <code>starter</code> process.</td>
</tr>
<tr>
<td>No, it cannot start a new job</td>
<td>The startd rejects the request for one of the following reasons:</td>
</tr>
<tr>
<td></td>
<td>• Jobs have been suspended, flushed, or drained</td>
</tr>
<tr>
<td></td>
<td>• The job limit set for the <code>MAX_STARTERS</code> keyword has been reached</td>
</tr>
<tr>
<td></td>
<td>• There are not enough classes available for the designated job class</td>
</tr>
</tbody>
</table>

- Receives requests from the master (via `llctl`) 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 ilstatus and ilq 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.

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

The negotiator daemon receives status messages from each schedd and startd daemon running in the cluster. The negotiator daemon tracks:
- Which schedd daemons are running
- Which startd daemons are running, and the status of each startd machine.

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

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

The negotiator performs the following functions:
- Receives and records job status changes from the schedd daemon.
- Schedules jobs based on a variety of scheduling criteria and policy options. Once a job is selected, the negotiator contacts the schedd that originally created the job.
- Handles requests to:
  - Set priorities
  - Query about jobs
Daemons

- 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 8, “Administering and configuring LoadLeveler,” on page 67.

The gsmonitor daemon

<table>
<thead>
<tr>
<th>Linux notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>The gsmonitor daemon is not available in LoadLeveler for Linux.</td>
</tr>
</tbody>
</table>

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

If the GSMONITOR_DOMAIN keyword was not specified in the LoadLeveler configuration file then LoadLeveler will try to determine if the machine is running in a peer (cluster) domain or a PSSP domain. If the gsmonitor daemon is running in an active peer domain then it will use the RMC API to determine the node numbers and names of machines running in the cluster, otherwise it will assume it is running in a PSSP domain and attempt to use the SDR access routines to gather information. If the administrator restricted where the gsmonitor daemon can run by specifying the GSMONITOR_DOMAIN as either PSSP or PEER, then the daemon will start only if it is in a valid domain corresponding to what was specified in the configuration file.

If the administrator sets up a LoadLeveler administration file that contains OSIs spanning several PSSP or peer domains then a gsmonitor daemon must be started in each domain. A gsmonitor daemon can monitor only the OSIs contained in the domain within which it is running. The administrator specifies which OSIs run the gsmonitor daemon by specifying GSMONITOR_RUNS_HERE=TRUE in the local configuration file for that OSI. The default for GSMONITOR_RUNS_HERE is False.

The gsmonitor daemon should be run on one or two nodes in each domain (PSSP, peer, or both). By running LoadL_GSmonitor on more than one node in a domain
you will have a backup in case one of the nodes that the monitor is running on goes down. LoadL_GSmonitor subscribes to the Group Services system-defined host membership group, which is represented by the

**HA_GS_HOST_MEMBERSHIP** Group Services keyword. This group monitors every configured node in the system partition (when running in a PSSP domain) and every node in the active peer domain.

**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 PSSP domains or *RSCT Administration Guide*, SA22-7889 for PEER domains. 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 471. (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 may actually complete and therefore be removed from the schedd’s list of active jobs. When communication resumes between the two daemons, the negotiator will move such jobs to the Terminated state, where they will remain for a set period of time (specified by the 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 383.

**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.
Job states
Chapter 15. Commands
Ilacctmrg

Ilacctmrg - Collect machine history files

Purpose
Collects individual machine history files together into a single file.

Syntax
Ilacctmrg [-?] [-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 lowest level of the operating system to run this release.
-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 Ilacctmrg 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 Ilssummary command. For example, if you created the file globalhist.199808301050, you can issue Ilssummary 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:
Ilacctmrg -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)

Security
Administrators can issue this command.
Ilcancel - Cancel a submitted job

Purpose
Cancels one or more jobs from the LoadLeveler queue.

Syntax
Ilcancel [-?] [-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 lowest level of the operating system to run this release.
-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
IlsSubmit command. The jobid is required.
• stepid (delimited by dot) is the step ID assigned to the job when it was
submitted using the IlsSubmit 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 (Canceled) indicates the job was canceled. A job state of RP (Remove Pending) indicates the job is in the process of being canceled.

When cancelling a job from a submit-only machine, you must specify the machine name that scheduled the job. For example, if you submitted the job from machine A, a submit-only machine, and machine B, a scheduling machine, scheduled the job to run, you must specify machine B’s name in the cancel command. If machine A and B are in different sub-domains, you must specify the fully-qualified name of the job in the cancel command. You can use the `llq -l` command to determine the fully-qualified name of the job.

Examples

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.
```

Security

Administrators and users can issue this command.
Ilckpt

Ilckpt - 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.

Linux notes

The Ilckpt command is not supported in LoadLeveler for Linux.

Syntax

Ilckpt [ -? | -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 lowest level of the operating system to run this release.

-k Specifies that the job step is to be terminated after a successful checkpoint. The default is for the job to continue. Note that you cannot use the -k and -u flags together. If you need to restart the job on the same node do not use the -k flag.

-u Specifies that the job step is to be put on user hold after a successful checkpoint. The default is for the job to continue. Note that you cannot use the -k and -u flags together.

-r When this flag is issued, it specifies that the command is to return without waiting for the checkpoint to complete. When using this flag you should be aware that information relating to the success or failure of the checkpoint will not be available to the command. The default is for the checkpoint to complete before returning.

-q Specifies quiet mode, 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 Ilsubmit command (jobid is required)
- stepid: the step ID assigned to the job when it was submitted using the Ilsubmit command (stepid is required)

Description

The Ilckpt command should be used to save the state of the job in the event it does not complete. Use the command only with jobs that are marked as checkpointable. You can mark a job step for checkpoint by specifying checkpoint=yes or checkpoint=interval in the job command file. Use
checkpoint=yes to set checkpointing for an interactive job. Refer to "Step 14: Enable checkpointing" on page 392 for more information.

When a job is checkpointed it can later be restarted from the checkpoint file rather than the beginning of the job. To restart a job from a checkpoint file, the original job command file should be used with the value of the restart_from_ckpt keyword set to yes. The name and location of the checkpoint file should be specified by the ckpt_dir and ckpt_file keywords.

If you need to restart the job on the same nodes, do not use the -k flag. Instead, use the -u flag to place the job in a hold state. You can later release the job from the hold state by issuing the llhold -r command.

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

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

Security

Administrators and users can issue this command.
Ilclass - Query class information

Purpose
Returns information about classes.

Syntax
Ilclass [-?] [-H] [-v] [-l] [classlist]

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 lowest level of the operating system to run this release.
-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:
Ilclass -l silver gold

Results
The Standard Listing: The standard listing is generated when you do not specify -l
with the Ilclass command. The following is sample output from the Ilclass 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>MaxJobCPU</th>
<th>MaxProcCPU</th>
<th>Free</th>
<th>Max</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel</td>
<td>2+02:45:00</td>
<td>05:30:00</td>
<td>18</td>
<td>24</td>
<td>Parallel job class</td>
</tr>
</tbody>
</table>

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 106 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
Ilclass

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 llclass command. The following is sample output from the llclass -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 johntt loadl rhclark srherb wilson
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
Class_ckpt_dir:
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: 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
Max_proto_instances: 2
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.

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 is used to prevent preemption of a job step by specifying a value of 99, or to return a job step to the normal preemptable state by specifying a value of 1.

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 106 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
Ilclass

number of tasks allowed to run at any given time for job steps of the specified class in the LoadLeveler cluster.

Max_proto_instances
The maximum number of protocol instances allowed for a job step of the specified class.

Name  The name of the class

Nice  The nice value of jobs of the specified class.

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 Ilclass is a collection of these snapshots, all taken at varying times, the total picture may not be completely consistent.

Security
Administrators and users can issue this command.
Ilctl - Control LoadLeveler daemons

Purpose

Controls LoadLeveler daemons on all members of the LoadLeveler cluster.

Syntax

```
Ilctl [-?] [-H] [-v] [-q] [-g] [-h <hostname>] <keyword>
```

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 lowest level of the operating system to run this release.

-q  Specifies quiet mode: print no messages other than error messages.

-g  Indicates that the command applies globally to all machines, except submit-only machines, that are listed in the administration file.

-h host

Indicates that the command applies to only the host machine in the LoadLeveler cluster. If neither -h nor -g is specified, the default is the machine on which the Ilctl 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 84.

**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...
Ilctl

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 Appendix B. Troubleshooting of IBM LoadLeveler Using and Administering.

reconfig
 Forces all daemons to reread the administration and configuration files.

recycle
 Stops all LoadLeveler daemons and restarts them.

resume [sched | startd [classlist | allclasses]]
 When you issue `resume` with no options, job submission and job execution on this machine is resumed. When you issue `resume sched`, the schedd machine resumes the submission of jobs. When you issue `resume startd`, the startd machine resumes the execution of jobs. When you issue `resume startd classlist`, the startd machine resumes the execution of those job classes you specify which are also configured (defined on the machine). When you issue `resume startd allclasses`, the startd machine resumes the execution of all configured classes.

start[drained]
 When you issue `start` with no options it starts the LoadLeveler daemons on the machine or machines designated, either explicitly or implicitly. When you issue `start` without the `-g` or `-h` flag the LoadLeveler daemons are started on the same machine that issued the command. When you issue `start` with either the `-g` or `-h` flag, `rshell (rsh)` is used to start the LoadLeveler daemons on all machines specified in the administration file, or on the machine specified by the `-h` flag. You must have `rsh` privileges in order to use either the `-g` or `-h` flag.

When you issue `start` with the `drained` option the LoadLeveler daemons are started, but the startd daemon is started in the drained state.

LoadLeveler commands that run `rshell` include `Ilctl version` and `Ilctl start`.

stop
 Stops the LoadLeveler daemons on the specified machine.

suspend
 Suspends all jobs on this machine. This is not supported for parallel jobs.
version
Displays release number, service level, service level date, and operating system information for every LoadLeveler executable.

When you issue `llctl version` with either the `-g` or `-h` flag, `rsh` (`rsh`) is used to run the command on all machines specified in the administration file, or on the machine specified by the `-h` flag. You must have `rsh` privileges in order to use `llctl version` with either the `-g` or `-h` flag.

LoadLeveler commands that run `rsh` include `llctl version` and `llctl start`.

Description
This command sends a message to the master daemon on the target machine requesting that action be taken on the members of the LoadLeveler cluster. Note the following when using this command:

- To perform the control operations of the `llctl` command, you must be a LoadLeveler administrator. The only exception to this rule is the “start” operation.
- LoadLeveler will fail to start if any value has been set for the MALLOCTYPE environment variable.
- After you make changes to the administration and configuration files for a running cluster, be sure to issue `llctl reconfig`. This command causes the LoadLeveler daemons to reread these files, and prevents problems that can occur when the LoadLeveler commands are using a new configuration while the daemons are using an old configuration.

**Note:** Changes to SCHEDULER_TYPE will not take effect at reconfiguration. The administrator must stop and restart or recycle LoadLeveler when changing SCHEDULER_TYPE.

- The `llctl drain startd classlist` command drains classes on the startd machine, and the startd daemon remains operational. If you reconfigure the daemon, the draining of classes remains in effect. However, if the startd goes down and is brought up again (either by the master daemon or by a LoadLeveler administrator), the startd daemon is configured according to the global or local configuration file in effect, and therefore the draining of classes is lost. Draining all the classes on a startd machine is not equivalent to draining the startd machine. When you drain all the classes, the startd enters the Idle state. When you drain the startd, the startd enters the Drained state. Similarly, resuming all the classes on a startd machine is not equivalent to resuming the startd machine.

- If a job step is running on a machine that receives the `llctl recycle` command, or the `llctl stop` and `llctl start` commands, the running job step is terminated. If the restart option in the job command file was set to yes, then the job step will be restarted when LoadLeveler is restarted. If the job step is checkpointable, it will be restarted from the last valid checkpoint file when LoadLeveler is restarted.

- If you find that the `llctl -g` command (even if it is specified with additional options) is taking a long time to complete, you should consider using the AIX command `dsh` to send `llctl` commands (omitting the `-g` flag) to multiple nodes in a parallel fashion.

- When a node running a schedd daemon fails, resources that have been allocated to any of the jobs scheduled by that schedd are unavailable until the schedd is restarted. Administrators can, however, recover these resources by using the `llctl` command’s `purgeschedd` keyword to purge (remove) all of the jobs scheduled by the schedd on the down node. The purgeschedd keyword can only work in
conjunction with the `schedd_fenced` keyword, in the administration file, which causes the central manager to ignore (fence) the schedd daemon running on the target node. You must reconfigure the central manager so it can recognize this fence. To use the purgeschedd keyword:

1. Recognize that a node running a schedd daemon is down, and that the node will be down long enough to necessitate that you recover the resources allocated to jobs scheduled by that schedd.
2. Add the statement "schedd_fenced = true" to the failed node’s administration file machine stanza.
3. Reconfigure the central manager node so that the central manager recognizes the fenced schedd daemon.
4. Invoke "llctl -h host purgeschedd" to purge all of the jobs scheduled by the schedd on the failed node.
5. Once the failed node is working again, remove all of the files in the LoadLeveler spool directory. Remove the "schedd_fenced = true" statement from the administration file, then reconfigure the central manager node before starting schedd on the machine.

**Examples**

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 (with the exception of the submit-only machines), starting with the central manager, as defined in the machine stanzas of the administration file:
```
llctl -g start
```

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 -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 84.

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

Security
Administrators can issue this command.
The Ildcegrpmaint 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 Ildcegrpmaint 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 all the machines in the LoadLeveler cluster. The llextSDR command can be used to retrieve the DCE hostnames.

3. Before running the Ildcegrpmaint 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. For the purposes of the lldcegrpmaint command, the important actions are: (1) "Create dcehostnames" and (2) "Configure SP Trusted Services to use DCE Authentication."

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/Startd: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/Startd
   /.../c163.ppd.pok.ibm.com/LoadL/c163n02.ppd.pok.ibm.com/Kbdd
   /.../c163.ppd.pok.ibm.com/LoadL/c163n02.ppd.pok.ibm.com/GSmonitor
   ```
Security

DCE Administrators can issue this command.
IllexRPD - Extract data from an RSCT peer domain

Purpose

Extracts the necessary data from an RSCT peer domain (or local node if there is no active domain) to set up the administration file.

Syntax

IllexRPD [-H | -? | -v | -m | -a adapter_name]

Flags

-? Provides a short usage message.
-H Provides extended help information.
-v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.
-m Specifies that only machine stanzas are to be generated. The adapter stanzas (and the corresponding adapter_stanzas statement of the machine stanza) will be suppressed in the final output. This option is for Dynamic Adapter Configuration support for peer domains with AIX RSCT.

LoadLeveler will dynamically detect and handle adapters and adapter changes for any machine in these domains which do not specify an adapter stanza in the administration file.

-a adapter_name
Specifies that the interface name of the given adapter_name on each node is used as the label (machine stanza name) of the generated machine stanza.

If you do not specify an adapter (or if an adapter is specified but does not exist on a particular node) then the label used for that machine is the Name field from the RSCT IBM.PeerNode class for the machine in the cluster. This default naming behaves the same as the IllexSDR command does when extracting data from the SDR for a PSSP domain.

Note: If an administrator wants to configure LoadLeveler to communicate using the Switch Network Interface for High Performance Switch (HPS) adapters in a peer domain they should use the -a flag with ml0 specified as the adapter_name. ml0 is guaranteed to be present on every node that contains an HPS adapter.

It is recommended that you do not specify sn as the adapter name. If you do, the machine will be named with the IP name of the sn adapter. If that IP name becomes unavailable because the adapter changes, LoadLeveler will not be able to contact any daemons on that machine.

Description

The command extracts the data for LoadLeveler to setup the administration file. The IllexRPD command must be run on one of the nodes in an active RSCT peer domain to obtain the RSCT peer nodes and network interface data from that cluster. If you are not running the command in an active RSCT peer domain you will just get information from the local machine. Adapter stanza names for HPS adapters are not included in the machine stanza alias. If you run an application
which requires LoadLeveler to recognize a node by the interface name of a HPS adapter you must manually add the adapter stanza name for the HPS adapter as an alias in the machine stanza.

Since it is possible to have nodes defined to both the RSCT peer and the PSSP domains at the same time the system administrator must be very cautious when extracting and merging data from different domains to ensure duplicate or conflicting information does not end up in the LoadLeveler administration file. Use the llextRPD command to extract data from peer domains and the llextSDR command to extract data from PSSP domains.

Examples

The following example extracts the data from an RSCT peer domain:

```
llextRPD -a m10
```

Results

```
#llextRPD: Cluster = "llcluster" ID = "0Jt9zGF7nbDWWjTDrxjG" on Thu Jun 12 15:24:13 2003

c121san10.ppd.pok.ibm.com: type = machine
  adapter_stanzas = c121s0n10.ppd.pok.ibm.com c121s1n10.ppd.pok.ibm.com
c121san10.ppd.pok.ibm.com: type = adapter
  adapter_name = sn0
  network_type = switch
  interface_address = 192.168.0.10
  interface_name = c121s0n10.ppd.pok.ibm.com
  multilink_address = 10.10.10.10
  logical_id = 2
  adapter_type = Switch_Network_Interface_For_HPS
  device_driver_name = sni0
  network_id = 1

c121san11.ppd.pok.ibm.com: type = adapter
  adapter_name = sn1
  network_type = switch
  interface_address = 192.168.1.10
  interface_name = c121s1n10.ppd.pok.ibm.com
  multilink_address = 10.10.10.10
  logical_id = 0
  adapter_type = Switch_Network_Interface_For_HPS
  device_driver_name = sn1
  network_id = 1

c121san10.ppd.pok.ibm.com: type = adapter
  adapter_name = ml0
  network_type = multilink
  interface_address = 10.10.10.10
  interface_name = c121s0n10.ppd.pok.ibm.com
  multilink_list = sn0,sn1

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

c121san04.ppd.pok.ibm.com: type = machine
  adapter_stanzas = c121s0n04.ppd.pok.ibm.com c121s1n04.ppd.pok.ibm.com
```

llextRPD
alias = c121f1rp04.ppd.pok.ibm.com

c121s0n04.ppd.pok.ibm.com: type = adapter
   adapter_name = sn0
   network_type = switch
   interface_address = 192.168.0.4
   interface_name = c121s0n04.ppd.pok.ibm.com
   multilink_address = 10.10.10.4
   logical_id = 11
   adapter_type = Switch_Network_Interface_For_HPS
   device_driver_name = sni0
   network_id = 1

   c121s1n04.ppd.pok.ibm.com: type = adapter
   adapter_name = sn1
   network_type = switch
   interface_address = 192.168.1.4
   interface_name = c121s1n04.ppd.pok.ibm.com
   multilink_address = 10.10.10.4
   logical_id = 9
   adapter_type = Switch_Network_Interface_For_HPS
   device_driver_name = sni1
   network_id = 1

   c121san04.ppd.pok.ibm.com: type = adapter
   adapter_name = ml0
   network_type = multilink
   interface_address = 10.10.10.4
   interface_name = c121san04.ppd.pok.ibm.com
   multilink_list = sn0,sn1

   c121f1rp04.ppd.pok.ibm.com: type = adapter
   adapter_name = en0
   network_type = ethernet
   interface_address = 9.114.66.68
   interface_name = c121f1rp04.ppd.pok.ibm.com
   device_driver_name = ent0

Examples

   The following example extracts the data from an RSCT peer domain for a dynamic
   adapter configuration:
   llextrPD -m -a ml0

Results

   #llextrPD: Cluster = "acc97" ID = "28jek7RdrHdGwr5C6zQwWm" on Fri Aug 8 14:37:33 2003

   c97ml0n13.ppd.pok.ibm.com: type = machine
      alias = c97n13.ppd.pok.ibm.com

   c97ml0n09.ppd.pok.ibm.com: type = machine
      alias = c97n09.ppd.pok.ibm.com

   c97ml0n01.ppd.pok.ibm.com: type = machine
      alias = c97n01.ppd.pok.ibm.com

   c97ml0n05.ppd.pok.ibm.com: type = machine
      alias = c97n05.ppd.pok.ibm.com

Security

   Administrators and users can issue this command.
IlextSDR - 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.

<table>
<thead>
<tr>
<th>Linux notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>The IlextSDR command is not supported in LoadLeveler for Linux.</td>
</tr>
</tbody>
</table>

Syntax
IlextSDR [?] [-H] [-v] [-a adapter_name]

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 lowest level of the operating system to run this release.
-a adapter_name
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 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_excluse_enable keyword, whose value is obtained from the spacct_excluse_enable attribute in the class of the SDR. For more information on adapter stanzas, see “Step 5: Specify adapter stanzas” on page 345. For more information on machine stanzas, see “Step 1: Specify machine stanzas” on page 344.

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.

Since it is possible to have nodes defined to both the PSSP and the RSCT peer domains at the same time the system administrator must be very cautious when extracting and merging data from different domains to ensure duplicate or conflicting information does not end up in the LoadLeveler administration file. Use
the **llextSDR** command to extract data from PSSP domains and the **llextRPD** command to extract data from peer domains.

**Examples**

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

```
llextSDR
```

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

```
llextSDR -a css0
```

**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 **llextSDR** command:

```bash
#llextSDR: 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: type = machine
    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
    c97sn04.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
c97san04.ppd.pok.ibm.com: type = adapter
        adapter_name = css1
        network_type = switch
        interface_address = 9.114.59.4
        interface_name = c97san04.ppd.pok.ibm.com
        multilink_address = 9.114.59.196
        switch_node_number = 3
        css_type = SP_Switch2_Adapter
c97san04.ppd.pok.ibm.com: type = adapter
        adapter_name = css0
        network_type = switch
        interface_address = 9.114.59.132
        interface_name = c97san04.ppd.pok.ibm.com
        multilink_address = 9.114.59.196
        switch_node_number = 3
        css_type = SP_Switch2_Adapter
```

Chapter 15. Commands  181
This is sample output for the llextSDR -a css0 command:

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

Security

Administrators and users can issue this command.
Ilfavorjob - 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
Ilfavorjob [-?] [-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 lowest level of the operating system to run this release.
-q Specifies quiet mode: print no messages other than error messages.
-u Unfavors previously favored jobs, requeuing them according to their original priority levels.

joblist
Is a blank-delimited list of 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 llsubmit 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 llsubmit 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:
Ilfavorjob iron.12.4 zinc.8.2

This example unfavors job steps 12.4 on the machine iron and 8.2 on the machine zinc:
Ilfavorjob -u iron.12.4 zinc.8.2
Security

Administrators can issue this command.
Ilfavoruser

Ilfavoruser - 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
Ilfavoruser [-?] [-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 lowest level of the operating system to run this release.
-q Specifies quiet mode: print no messages other than error messages.
-u Unfavors previously favored users, reordering their 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:
Ilfavoruser ellen fred

This example unfavors all queued jobs submitted by users ellen and fred:
Ilfavoruser -u ellen fred

Security
Administrators can issue this command.
Ilhold - 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
Ilhold [-?] [-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 lowest level of the operating system to run this release.
-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 llsubmit command. jobid is required.
Ilhold

- **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.
```

**Security**

Administrators and users can issue this command.
Ilinit - Initialize machines in the LoadLeveler cluster

Purpose

Initializes a new machine as a member of the LoadLeveler cluster

Syntax

```
[-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
Ilinit 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 on AIX or the /opt/ibmll/LoadL/full directory
on Linux.

-cm machine

machine is the central manager machine, where the negotiator daemon runs.

-debug

Displays debug messages during the execution of Ilinit.

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.

Ilinit 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.

Security

Administrators can issue this command.
Ilmodify - Change attributes of a submitted job step

Purpose

Changes the attributes or characteristics of a submitted job.

Syntax

```
Ilmodify | -? | -H | -v | [-q] | -x <execution_factor> | -c <consumable_cpus> | -m <consumable_real_memory> | -W <wclimit_add_min> | -C <job_class> | -a <account_no> | <jobstep>
```

Flags

-? Provides a short usage message.

-H Provides extended help information.

-v Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release.

-q Specifies quiet mode: print no messages other than error messages.

-x <execution_factor>
For GANG scheduling only, specifies the execution factor value. Valid values are 1 and 99.
- 99 – makes a job step in RUNNING or STARTING state non-preemptable.
- All other job steps running on the same nodes are preempted until the non-preemptable job step finishes running or its execution factor is lowered.
- This is a LoadLeveler administrator only option.
- 1 – returns a job step to the normal preemptable state. This value is the default.

-c <consumable_cpus>
Specifies the consumable CPU value for an idle-like job step.

Allows the ConsumableCpus resource requirement to be reset to the specified value. This value can be any integer equal or greater than zero (0) and should follow the rules for the resources keyword in the job command file.

-m <consumable_real_memory>
Specifies the consumable real memory value for an idle-like job step.

Allows the ConsumableMemory resource requirement to be reset to the specified value. No units should be specified, as megabytes (MB) is assumed.
This value can be any integer equal or greater than zero (0).

-W <wclimit_add_min>
Specifies additional time in minutes to add to the wall clock limits of a running-like job step. This option is for preventing a job step from being killed due to the wall clock limits. It is a LoadLeveler administrator only option.

Both the hard limit and soft limit are increased by the specified value. This value can be any integer greater than 0.

The increase will only be effective if a limit was originally set and not already exceeded. If you attempt to modify the wall clock limit for a job step that is approaching its current wall clock limit, delays in inter-daemon communications may cause LoadLeveler to terminate the job step before the wall clock limit is updated.
Ilmodify

-C <job_class>
  Specifies the job class name.
  Allows the job class name to be reset to the specified value for an idle-like job step. This value can be any string without white spaces.

-a <account_no>
  Specifies the account number.
  Allows the account number to be reset to the specified value for an idle-like job step.

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

All options are for the job step owner or a LoadLeveler administrator on an idle-like job step with the following exceptions:

• -x 99 and -W are LoadLeveler administrator only options
• -x 99 is valid only for a job step in RUNNING or STARTING state
• -x 1 is valid for a job step in basically any job state
• -W is valid only for a job step in a running-like state

At the time a job step is modified, LoadLeveler does not check to make certain that the job step with the modified values can be scheduled to run.

To determine if a modification request is successful, issue the llq -x -l command and check the following field in the output.

<table>
<thead>
<tr>
<th>Options</th>
<th>Field to check</th>
</tr>
</thead>
<tbody>
<tr>
<td>-x</td>
<td>Execution Factor</td>
</tr>
<tr>
<td>-c or -m</td>
<td>Resources</td>
</tr>
<tr>
<td>-W</td>
<td>Wall Clk Hard Limit/Wall Clk Soft Limit</td>
</tr>
<tr>
<td>-C</td>
<td>Class</td>
</tr>
<tr>
<td>-a</td>
<td>Account</td>
</tr>
</tbody>
</table>

An idle-like state is one of the following job states:

• Idle
• Deferred
• User Hold
• System Hold
• User & System Hold
• Not Queued
• Vacated
• Vacate Pending
• Rejected
• Reject Pending
A running-like state is one of the following job states:
- Checkpointing
- Pending
- Preempted
- Preempt Pending
- Resume Pending
- Running
- Starting

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:
```bash
llmodify -x 99 c163n07.12.0
```
To extend the wall clock limits of job step c163n07.12.0 by 30 minutes:
```bash
llmodify -W 30 c163n07.12.0
```

Results
The following shows a sample system response for llmodify -x 1 c163n07.12.0:
```bash
llmodify: request has been sent to LoadLeveler.
```

llmodify returns the following exit values:
- 0 (Command has run successfully)
- -1 (An error occurred)

Security
Administrators and users can issue this command.
Ilpreempt

Ilpreempt - 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 -r flag. After the resume operation, the job step resumes its normal state controlled by the LoadLeveler scheduling and preemption rules.

Syntax
Ilpreempt { -? | -H | -v | [-q] [-r | -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 lowest level of the operating system to run this release.
-q Specifies quiet mode: print no messages other than error messages.
-r Resume jobs preempted using the Ilpreempt command.
-u Undo the previous preempt action. The -r option should be used to resume preempted jobs since the use of the -u option is changing in a future release.

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 Ilsubmit command. A jobid is required.
- stepid: The step ID assigned to the job when it was submitted using the Ilsubmit 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:
Ilpreempt c163n07.12.0

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

Results
The following shows a sample system response for the Ilpreempt command:
Ilpreempt: request has been sent to LoadLeveler.
Security

Administrators can issue this command.
Ilprio - 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. Ilprio 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
Ilprio [-?] [-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 lowest level of the operating system to run this release.
-q Specifies quiet mode: print no messages other than error messages.
+p 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
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 51 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.
```

**Security**

Administrators and users can issue this command.
Ilq

Ilq - Query job status

Purpose

Returns information about job steps in the LoadLeveler queues.

Syntax

```
[-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 lowest level of the operating system to run this release.

-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, Ilq connects with the central manager to obtain job information. When you specify -x, Ilq connects to the schedd machine that received the specified job to get extended job information. However, some statistics, including those corresponding to System Priority and q_sysprio, are available only from the central manager. Do not use the -x option if you need these statistics.

When specified without -l, CPU usage for active jobs is reported in the short format.

Note: Using both the -l and -x options without a joblist specification can produce a very long report and excessive network traffic.

-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, Ilq determines why the job remains in one of the given states instead of Running.

-l Specifies that a long listing be generated for each job for which status is requested. Fields included in the long listing are shown in "Results" on page 200.

If -l is not specified, then the standard listing is generated as shown in "Results" on page 200.

-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 -I flag. The output fields produced by this flag all have a fixed length. The output is displayed in the order in which you specify the categories. category_list can be one or more of the following:

%a Account number
%c Class
%cc Completion code
%d Completion date
%dd Dispatch Date
%dh Hold date
%dq Queue date
%gl LoadLeveler group
%gu UNIX group
Using and Administering LoadLeveler

llq

%h  Hostname (first hostname if more than one machine is allocated to the job step)
%id  Step ID
%is  Virtual image size
%jn  Job name
%jt  Job type
%nh  Number of hosts allocated to the job step
%o  Job owner
%p  User priority
%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:

<table>
<thead>
<tr>
<th>Id</th>
<th>Owner</th>
<th>Submitted</th>
<th>ST</th>
<th>PRI</th>
<th>Class</th>
<th>Running On</th>
</tr>
</thead>
<tbody>
<tr>
<td>mars.498.0</td>
<td>brownap</td>
<td>5/20 11:31</td>
<td>R</td>
<td>100</td>
<td>silver</td>
<td>mars</td>
</tr>
<tr>
<td>mars.499.0</td>
<td>brownap</td>
<td>5/20 11:31</td>
<td>R</td>
<td>50</td>
<td>No_Class</td>
<td>mars</td>
</tr>
<tr>
<td>mars.501.0</td>
<td>brownap</td>
<td>5/20 11:31</td>
<td>I</td>
<td>50</td>
<td>silver</td>
<td></td>
</tr>
</tbody>
</table>

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
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:

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

For a detailed explanation of job states, see “Job states” on page 153.

**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 196.

**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 %gl %h
```
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
```

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. Note that Large Page memory information is not included in the statistics since WLM does not manage Large Page memory. 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

c209f1n101.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.

The following is sample output for the **llq** -l command for the serial job step c271f2rp01.ppd.pok.ibm.com.8.0:

```
=============== Job Step c271f2rp01.ppd.pok.ibm.com.8.0===============
  Job Step Id: c271f2rp01.ppd.pok.ibm.com.8.0
  Job Name: c271f2rp01.ppd.pok.ibm.com.8
  Step Name: job_step_1
Structure Version: 10
  Owner: wilson
  Queue Date: Sat Aug 9 13:59:13 EDT 2003
  Status: Running
Execution Factor: 1
Completion Date:
  Completion Code:
    User Priority: 50
    user_sysprio: 0
    class_sysprio: 45
    group_sysprio: 0
  System Priority: -255
    q_sysprio: -255
  Notifications: Complete
Virtual Image Size: 15 kb
Large Page: N
Checkpointerable: no
  Ckpt Start Time:
Good Ckpt Time/Date: 0 seconds
Fail Ckpt Time/Date:
  Ckpt Accum Time: 0 seconds
  Checkpoint File:
  Restart From Ckpt: no
Restart Same Nodes: no
  Restart: no
  Hold Job Until:
    Cmd: /u/wilson/LL_SERIAL/c_cpu_60_sleep_60
    Args: arg_01 arg_02 arg_3
    Env:
      In: /dev/null
      Out: job1.c271f2rp01.8.0.out
      Err: job1.c271f2rp01.8.0.err
Initial Working Dir: /u/wilson/LL_SERIAL
Dependency:
Resources: ConsumableMemory(100.000 mb) ConsumableVirtualMemory(150.000 mb) ConsumableCpus(1)
Requirements: (Arch == "R6000") && ((OpSys == "AIX51") || (OpSys == "AIX52"))
Preferences: 
    (Machine == "c271f2rp01.ppd.pok.ibm.com" || Machine == "c271f2rp02.ppd.pok.ibm.com")
    & (Feature == "ESSSL")
```

Figure 19. llq -l output for a serial job step (Part 1 of 2)
The following listing is sample output for the `llq -l -x c271f2rp01.ppd.pok.ibm.com.16.0` command, where `c271f2rp01.ppd.pok.ibm.com.16.0` is a parallel, non-checkpointing job step:
Figure 20. llq -l -x output for a parallel, non-checkpointing job step (Part 1 of 4)
Figure 20. \texttt{llq -l -x} output for a parallel, non-checkpointing job step (Part 2 of 4)
<table>
<thead>
<tr>
<th>Detailed Information</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running Host</td>
<td>c271f2rp01.ppd.pok.ibm.com</td>
</tr>
<tr>
<td>Machine Speed</td>
<td>1.000000</td>
</tr>
<tr>
<td>Starter User Time</td>
<td>00:00:00.160000</td>
</tr>
<tr>
<td>Starter System Time</td>
<td>00:00:00.170000</td>
</tr>
<tr>
<td>Starter Total Time</td>
<td>00:00:00.330000</td>
</tr>
<tr>
<td>Starter maxrss</td>
<td>2640</td>
</tr>
<tr>
<td>Starter ixrss</td>
<td>5356</td>
</tr>
<tr>
<td>Starter idrss</td>
<td>18268</td>
</tr>
<tr>
<td>Starter isrss</td>
<td>0</td>
</tr>
<tr>
<td>Starter minflt</td>
<td>0</td>
</tr>
<tr>
<td>Starter majflt</td>
<td>0</td>
</tr>
<tr>
<td>Starter nswap</td>
<td>0</td>
</tr>
<tr>
<td>Starter inblock</td>
<td>0</td>
</tr>
<tr>
<td>Starter oublock</td>
<td>0</td>
</tr>
<tr>
<td>Starter msgsnd</td>
<td>0</td>
</tr>
<tr>
<td>Starter msgrcv</td>
<td>0</td>
</tr>
<tr>
<td>Starter nsignals</td>
<td>1</td>
</tr>
<tr>
<td>Starter nvcsw</td>
<td>163</td>
</tr>
<tr>
<td>Step User Time</td>
<td>00:03:30.360000</td>
</tr>
<tr>
<td>Step System Time</td>
<td>00:00:00.990000</td>
</tr>
<tr>
<td>Step Total Time</td>
<td>00:03:31.350000</td>
</tr>
<tr>
<td>Step maxrss</td>
<td>22820</td>
</tr>
<tr>
<td>Step ixrss</td>
<td>170284</td>
</tr>
<tr>
<td>Step idrss</td>
<td>60582968</td>
</tr>
<tr>
<td>Step isrss</td>
<td>0</td>
</tr>
<tr>
<td>Step minflt</td>
<td>37654</td>
</tr>
<tr>
<td>Step majflt</td>
<td>0</td>
</tr>
<tr>
<td>Step nswap</td>
<td>0</td>
</tr>
<tr>
<td>Step inblock</td>
<td>0</td>
</tr>
<tr>
<td>Step oublock</td>
<td>0</td>
</tr>
<tr>
<td>Step msgsnd</td>
<td>0</td>
</tr>
<tr>
<td>Step msgrcv</td>
<td>0</td>
</tr>
<tr>
<td>Step nsignals</td>
<td>0</td>
</tr>
<tr>
<td>Step nvcsw</td>
<td>3033</td>
</tr>
</tbody>
</table>

Figure 20. `llq -l -x` output for a parallel, non-checkpointing job step (Part 3 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

---

**Step nivcsw: 488**

<table>
<thead>
<tr>
<th>Name</th>
<th>Requirements</th>
<th>Preferences</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(Arch == &quot;R6000&quot;) &amp;&amp; ((OpSys == &quot;AIX51&quot;)</td>
<td></td>
</tr>
<tr>
<td>Node minimum</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Node maximum</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Node actual</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Allocated Hosts</td>
<td>c271f2rp01.ppd.pok.ibm.com:RUNNING:sn0(2502,LAPI,US,1M), sn1(2502,MPI,US,1M), sn0(2504,LAPI,US,1M), sn1(2504,MPI,US,1M), sn0(2506,LAPI,US,1M), sn1(2506,MPI,US,1M) + c271f2rp02.ppd.pok.ibm.com:RUNNING:sn0(2502,LAPI,US,1M), sn1(2502,MPI,US,1M), sn0(2504,LAPI,US,1M), sn1(2504,MPI,US,1M), sn0(2506,LAPI,US,1M), sn1(2506,MPI,US,1M)</td>
<td></td>
</tr>
</tbody>
</table>

**Master Task**

- **Executable**: /bin/poe
- **Exec Args**: /u/wilson/LL_PARA/ivp_cpu_110_120_sleep_130 -ilevel 6 -labelio yes
- **Num Task Inst**: 1
  - Task Instance: c271f2rp01:-1

**Task**

- **Num Task Inst**: 6
  - Task Instance: c271f2rp01:0:sn0(2502,LAPI,US,1M), sn1(2502,MPI,US,1M)
  - Task Instance: c271f2rp01:1:sn0(2504,LAPI,US,1M), sn1(2504,MPI,US,1M)
  - Task Instance: c271f2rp01:2:sn0(2506,LAPI,US,1M), sn1(2506,MPI,US,1M)
  - Task Instance: c271f2rp02:3:sn0(2502,LAPI,US,1M), sn1(2502,MPI,US,1M)
  - Task Instance: c271f2rp02:4:sn0(2504,LAPI,US,1M), sn1(2504,MPI,US,1M)
  - Task Instance: c271f2rp02:5:sn0(2506,LAPI,US,1M), sn1(2506,MPI,US,1M)

---

**Figure 20. llq -l -x output for a parallel, non-checkpointing job step (Part 4 of 4)**
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 153.

Execution Factor
Used only for Gang scheduling, Execution factor is used to prevent preemption of a job step by specifying a value of 99, or to return a job step to the normal preemptable state by specifying a value of 1.

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 51).
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.

Large Page
Indicates whether Large Page memory should be used to run this job step. Can be Y (use Large Page memory if available), N (No), or M (Mandatory).

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 stdin 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.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoadLeveler Group</td>
<td>The LoadLeveler group associated with the job step.</td>
</tr>
<tr>
<td>Class</td>
<td>The class of the job step as specified at job submission.</td>
</tr>
<tr>
<td>Ckpt Hard Limit</td>
<td>Checkpoint hard limit as specified at job step submission.</td>
</tr>
<tr>
<td>Ckpt Soft Limit</td>
<td>Checkpoint soft limit as specified at job step submission.</td>
</tr>
<tr>
<td>Cpu Hard Limit</td>
<td>CPU hard limit as specified at job submission.</td>
</tr>
<tr>
<td>Cpu Soft Limit</td>
<td>CPU soft limit as specified at job submission.</td>
</tr>
<tr>
<td>Data Hard Limit</td>
<td>Data hard limit as specified at job submission.</td>
</tr>
<tr>
<td>Data Soft Limit</td>
<td>Data soft limit as specified at job submission.</td>
</tr>
<tr>
<td>Core Hard Limit</td>
<td>Core hard limit as specified at job submission.</td>
</tr>
<tr>
<td>Core Soft Limit</td>
<td>Core soft limit as specified at job submission.</td>
</tr>
<tr>
<td>File Hard Limit</td>
<td>File hard limits as specified at job submission.</td>
</tr>
<tr>
<td>File Soft Limit</td>
<td>File soft limit as specified at job submission.</td>
</tr>
<tr>
<td>Stack Hard Limit</td>
<td>Stack hard limit as specified at job submission.</td>
</tr>
<tr>
<td>Stack Soft Limit</td>
<td>Stack soft limit as specified at job submission.</td>
</tr>
<tr>
<td>Rss Hard Limit</td>
<td>RSS hard limit as specified at job step submission.</td>
</tr>
<tr>
<td>Rss Soft Limit</td>
<td>RSS soft limit as specified at job step submission.</td>
</tr>
<tr>
<td>Step Cpu Hard Limit</td>
<td>Job step CPU hard limit as specified at job submission.</td>
</tr>
<tr>
<td>Step Cpu Soft Limit</td>
<td>Job step CPU soft limit as specified at job submission.</td>
</tr>
<tr>
<td>Wall Clk Hard Limit</td>
<td>Wall clock hard limit as specified at job submission.</td>
</tr>
<tr>
<td>Wall Clk Soft Limit</td>
<td>Wall clock soft limit as specified at job submission.</td>
</tr>
<tr>
<td>NQS Submit Queue</td>
<td>The name of the NQS pipe queue to which the NQS job will be routed.</td>
</tr>
<tr>
<td>NQS Query Queues</td>
<td>The NQS queue names you can use to monitor the job.</td>
</tr>
<tr>
<td>Comment</td>
<td>The comment specified by the comment keyword in the job command file.</td>
</tr>
</tbody>
</table>
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.

For more information on the maxrss, ixrss, idrss, isrss, minflt, majflt, nswap, inblock, oublock, msgsnd, msgrcv, nsignals, nvcsw, and nivcsaw fields refer to the AIX and Linux documentation for the rusage fields under the getrusage/getrusage64 subroutine.

Linux notes
Only minflt, majflt, and nswap are supported in LoadLeveler for Linux.

Starter maxrss/Step maxrss
Maximum resident set size utilized. Maximum value.

Starter ixrss/Step ixrss
An integral value indicating the amount of memory used by the text segment that was also shared among other processes (expressed in units of kilobytes * seconds-of-execution).

Starter idrss/Step Starter idrss
An integral value of the amount of unshared memory in the data segment of a process (expressed in units of kilobytes * seconds-of-execution).

Starter isrss/Step isrss
Depending on the Operating System, this field may contain the integral value of unshared stack size.
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 nvcsw/Step nvcsw
   Number of context switches due to voluntarily giving up processor. Cumulative total.

Starter nvcsw/Step nvcsw
   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)`

**Security**

Administrators and users can issue this command.
llstatus

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] [-a][-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 lowest level of the operating system to run this release.

-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.

-a Displays information for each virtual adapter followed by information for each physical adapter it manages.

-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
%sccs State of the schedd daemon
%sta Availability of the startd daemon
%stst State of the startd daemon
The standard listing is generated when you do not specify the -I option with the lstatus command. The following is sample output from the lstatus command, where there are two nodes in the cluster.

<table>
<thead>
<tr>
<th>Name</th>
<th>Schedd</th>
<th>InQ</th>
<th>Act</th>
<th>Startd</th>
<th>Run</th>
<th>LdAvg</th>
<th>Idle</th>
<th>Arch</th>
<th>OpSys</th>
</tr>
</thead>
<tbody>
<tr>
<td>k10n09.ppd.pok.ibm.com</td>
<td>Avail</td>
<td>3</td>
<td>1</td>
<td>Run</td>
<td>1.272</td>
<td>0</td>
<td>R6000</td>
<td>AIX51</td>
<td></td>
</tr>
<tr>
<td>k10n12.ppd.pok.ibm.com</td>
<td>Avail</td>
<td>0</td>
<td>0</td>
<td>Idle</td>
<td>0.00</td>
<td>365</td>
<td>R6000</td>
<td>AIX51</td>
<td></td>
</tr>
<tr>
<td>R6000/AIX51</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>running</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Machines</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>running</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:
Ilstatus

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 148.

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 149.

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 Ilstatus 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:

Ilstatus -R
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:

```plaintext
llstatus -F
```

<table>
<thead>
<tr>
<th>Floating Resource</th>
<th>Available</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA_licenses</td>
<td>20</td>
<td>29</td>
</tr>
<tr>
<td>Frame5</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>WorkBench6</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>XYZ_software</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

Resources with "+" appended to their names have the Total value reported from Startd.

Figure 21. Sample llstatus -R command output

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:

```plaintext
llstatus -f
```

```
Name       Schedd InQ    Memory     FreeVMemory Startd LdAvg OpSys
115.pok.ibm.com Avail 0 120  22708  Run 0.23 AIX51
116.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 115.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:

```plaintext
llstatus -r
```

```
115.pok.ibm.com!Avail!0!120!22708!Running!0.23!AIX51
116.pok.ibm.com!Avail!3!224!16732!Running!0.51!AIX51
```

Adapter Status Listing: When `llstatus` is issued with the `-a` flag information is generated about the status of the adapters associated with all the machines in the LoadLeveler cluster. The following is sample output from this command:

```plaintext
llstatus -a
```
For a switch adapter, the information format is: 
- adapter_name(network_type, interface_name, interface_address, multilink_address, switch_node_number or adapter_logical_id, available_adapter_windows/total_adapter_windows, 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)

The Long Listing: The long listing is generated when you specify the -l option with the ilstatus command. Following the sample output is an explanation of all possible fields displayed by the ilstatus command.

The following is sample output from the ilstatus -l c271f2rp02 command:

```
Name              = c271f2rp02.ppd.pok.ibm.com
Machine           = c271f2rp02.ppd.pok.ibm.com
Arch              = R6000
OpSys             = AIX52
SYSPRIO           = (0 - QDate)
MACHPRIO          = ((Memory + FreeRealMemory) - ((LoadAvg * 1000) + CustomMetric))
VirtualMemory     = 8378972 kb
Disk              = 41116 kb
KeyboardIdle      = 7
Tmp               = 41116 kb
LoadAvg           = 0.425903
ConfiguredClasses = Parallel(12) 85ba(2) misc(2) tiny(1) No_Class(7) small(14) large(1) medium(1)
AvailableClasses  = Parallel(12) 85ba(2) misc(2) tiny(1) No_Class(7) small(8) large(1) medium(1)
DrainingClasses   =
DrainedClasses    =
Pool              = 1 7
FabricConnectivity = 1:1
Adapter           = m10(multilink,c271f2san02.ppd.pok.ibm.com,10.10.10.6,)
                  networks(striped,c271f2san02.ppd.pok.ibm.com,10.10.10.6,-1,512,500M/512M,1,READY)
                  network(aggregate,,,10.10.10.6,-1,500/512,500M/512M,1,READY)
                  sn0(switch,c271f2son02.ppd.pok.ibm.com,192.168.0.6,10.10.10.6,2,250/256,250M/256M,1,READY)
                  sn1(switch,c271f2son02.ppd.pok.ibm.com,192.168.1.6,10.10.10.6,0,250/256,250M/256M,1,READY)
                  en0(ethernet,c271f2rp02.ppd.pok.ibm.com,9.114.175.82,)
Feature           = OSL ESSL
Max_Starters      = 500
Total Memory      = 7168 mb
Memory            = 6144 mb
FreeRealMemory    = 5291 mb
LargePageSize     = 16.000 mb
LargePageMemory   = 1.000 gb
FreeLargePageMemory= 816.000 mb
PagesFreed        = 0
PagesScanned      = 0
PagesPagedIn      = 0
PagesPagedOut     = 0
ConsumableResources = ConsumableCpus(4,4) ConsumableMemory(5.273 gb,5.859 gb) spice2g6(38,50)
ConfigTimeStamp   = Fri Aug 8 16:57:31 EDT 2003
```

Figure 22. Sample output from ilstatus -l c271f2rp02 (Part 1 of 2)
The fields FabricConnectivity, FreeLargePageMemory, LargePageMemory, LargePageSize, PagesFreed, PagesPagedIn, PagesPagedOut, and PagesScanned have no meaning in LoadLeveler for Linux and should be ignored by the user.

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 or adapter_logical_id, 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
Represents the current state of connectivity between the machine and the switch through the switch adapters. The format of the field is: network_id: connectivity, network_id: connectivity... where connectivity is either 1 or 0. A value of 1 indicates an active connection from the machine to a given network_id through one of the switch adapters.

Feature
Set of all features on this machine.

FreeLargePageMemory
Free Large Page memory.

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.
LargePageMemory
Configured Large Page physical memory.

LargePageSize
The size of a Large Page memory block.

LoadAvg
Berkely one-minute load average on machine.

Machine
Fully qualified name of the machine.

Machine Mode
The type of job this machine can run. This can be: batch, interactive, or general.

MACHPRIO
Actual expression that determines machine priority, defined in the configuration file.

MasterMachPriority
The machine priority for the parallel master node.

Max_Starters
Maximum number of initiators that can be used simultaneously on this machine.

Memory
Regular physical memory, in megabytes, on this machine.

Name
Hostname of the machine.

OpSys
Operating system on this machine.

PagesFreed
Pages freed per second. This value corresponds to the "fr" value of the vmstat command output.

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.
llstatus

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 149.

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.

**TotalMemory**
The sum of configured regular and Large Page memory.

**Unexpanded**
The number of job steps in this state on this schedd machine.

**VACATE**
The expression, defined following C conventions in the configuration file, that evaluates to true or false (T/F). This determines whether suspended jobs are vacated on this machine.

**Virtual Memory**
Available swap space (free paging space) in kilobytes, on this machine.

**Security**
Administrators and users can issue this command.
Ilsubmit

Ilsubmit - 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**

```
Ilsubmit [-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 lowest level of the operating system to run this release.
- `--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 `Ilsubmit` 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 318.
- The `Ilsubmit` 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 376.

**Linux notes**

The `Ilsubmit` command will display an error and fail to submit the job if it is a PVM or NQS job.
Examples

In this example, a job command file named `qtrlyrun.cmd` is submitted:

```sh
llsubmit qtrlyrun.cmd
```

Results

The following shows the results of the `llsubmit qtrlyrun.cmd` command issued from the machine `earth`:

```sh
llsubmit: The job "earth.505" has been submitted.
```

Note that 505 is the job ID generated by LoadLeveler.

Security

Administrators and users can issue this command.
Ilsummary - 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 384.

Syntax


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 lowest level of the operating system to run this release.

-x Provides extended information. Using -x can produce a very long report. This option is meaningful only when used with the -l option. You must enable the recording of accounting data in order to collect information with the -x flag. To do this, specify ACCT=A_ON A_DETAIL in your LoadL_config file.

-l Specifies that the long form of output is displayed.

-s Specifies a range for the start date (queue date) for accounting data to be included in this report. The format for entering the date is either MM/DD/YYYY (where MM is month, DD is day, and YYYY is year), MM/DD/YY (where YY is a two-digit year value), or a string of digits representing the number of seconds since 1970. If a two-digit year value is used, then 69-99 maps to 1969-1999, and 00-68 maps to 2000-2068. The default is to include all the data in the report.

-e Specifies a range for the end date (completion date) for accounting data to be included in this report. The format for entering the date is either MM/DD/YYYY (where MM is month, DD is day, and YYYY is year), MM/DD/YY (where YY is a two-digit year value), or a string of digits representing the number of seconds since 1970. The default is to include all the data in the report.

-u user

Specifies the user ID for whom accounting data is reported.

-c class

Specifies the class for which accounting data is reported. For reports of all formats (short, long and extended), Ilsummary will report information about every job which contains at least one step of the specified class. For the short format, Ilsummary 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), Ilsummary reports
information about every job which contains at least one step of the specified group. For the short format, llsummary also reports a job count and step count for each group; for these counts, a job’s group is determined by the group of its first step.

-G unixgroup
   Specifies the UNIX group for which accounting data is reported.

-a allocated
   Specifies the hostname that was allocated to run the job. You can specify the allocated host in short or long form.

-r report
   Specifies the report type. You must enable the recording of accounting data in order to collect information with the -r flag. To do this, specify ACCT=A_ON A_DETAIL in your LoadL_config file. You can choose one or more of the following reports:

   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 llsubmit command. The entire host.jobid string is required.

filelist
   Is a blank-delimited list of files containing the accounting data. If not specified, the default is the local history file on the machine from which the command was issued. You can use the llacctmrg command to combine history files on different schedd machines into a single history file.

Related Information
   In order to create an accounting report with the llsummary command, you must have read access to a history file. If a history file name is not specified as an
Ilsummary

argument, Ilsummary uses the history file in the LoadLeveler spool directory of the local machine as input. By default, the permissions of the spool directory are set by the Ilinit 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:

Ilsummary -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 Ilsummary. 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Jobs</th>
<th>Steps</th>
<th>Job Cpu</th>
<th>Starter Cpu</th>
<th>Leverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>krystal</td>
<td>15</td>
<td>36</td>
<td>00:09:50</td>
<td>00:00:10</td>
<td>59.0</td>
</tr>
<tr>
<td>lixin3</td>
<td>18</td>
<td>54</td>
<td>00:08:28</td>
<td>00:00:16</td>
<td>31.8</td>
</tr>
<tr>
<td>TOTAL</td>
<td>33</td>
<td>90</td>
<td>00:18:18</td>
<td>00:00:27</td>
<td>40.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>Jobs</th>
<th>Steps</th>
<th>Job Cpu</th>
<th>Starter Cpu</th>
<th>Leverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>small</td>
<td>9</td>
<td>21</td>
<td>00:01:03</td>
<td>00:00:06</td>
<td>10.5</td>
</tr>
<tr>
<td>large</td>
<td>12</td>
<td>36</td>
<td>00:13:45</td>
<td>00:00:11</td>
<td>75.0</td>
</tr>
<tr>
<td>osl2</td>
<td>3</td>
<td>9</td>
<td>00:00:27</td>
<td>00:00:02</td>
<td>13.5</td>
</tr>
<tr>
<td>No_class</td>
<td>9</td>
<td>24</td>
<td>00:03:01</td>
<td>00:00:06</td>
<td>30.2</td>
</tr>
<tr>
<td>TOTAL</td>
<td>33</td>
<td>90</td>
<td>00:18:18</td>
<td>00:00:27</td>
<td>40.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Jobs</th>
<th>Steps</th>
<th>Job Cpu</th>
<th>Starter Cpu</th>
<th>Leverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>No_Group</td>
<td>12</td>
<td>30</td>
<td>00:09:32</td>
<td>00:00:09</td>
<td>63.6</td>
</tr>
<tr>
<td>chemistry</td>
<td>7</td>
<td>18</td>
<td>00:04:50</td>
<td>00:00:05</td>
<td>58.0</td>
</tr>
<tr>
<td>engineering</td>
<td>14</td>
<td>42</td>
<td>00:03:56</td>
<td>00:00:12</td>
<td>19.7</td>
</tr>
<tr>
<td>TOTAL</td>
<td>33</td>
<td>90</td>
<td>00:18:18</td>
<td>00:00:27</td>
<td>40.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Account</th>
<th>Jobs</th>
<th>Steps</th>
<th>Job Cpu</th>
<th>Starter Cpu</th>
<th>Leverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>33333</td>
<td>16</td>
<td>39</td>
<td>00:05:54</td>
<td>00:00:11</td>
<td>32.2</td>
</tr>
<tr>
<td>22222</td>
<td>15</td>
<td>45</td>
<td>00:12:05</td>
<td>00:00:13</td>
<td>55.8</td>
</tr>
<tr>
<td>99999</td>
<td>2</td>
<td>6</td>
<td>00:00:18</td>
<td>00:00:01</td>
<td>18.0</td>
</tr>
<tr>
<td>TOTAL</td>
<td>33</td>
<td>90</td>
<td>00:18:18</td>
<td>00:00:27</td>
<td>40.7</td>
</tr>
</tbody>
</table>

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 ilsummary -r throughput command. Only the user output is shown; the class, group, and account lines are not shown.

<table>
<thead>
<tr>
<th>Name</th>
<th>Jobs</th>
<th>Steps</th>
<th>AvgQueueTime</th>
<th>AvgRealTime</th>
<th>AvgCPUTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadl</td>
<td>1</td>
<td>4</td>
<td>0+00:00:03</td>
<td>0+00:05:27</td>
<td>0+00:05:17</td>
</tr>
<tr>
<td>user1</td>
<td>2</td>
<td>6</td>
<td>0+00:03:05</td>
<td>0+00:03:45</td>
<td>0+00:03:04</td>
</tr>
<tr>
<td>ALL</td>
<td>3</td>
<td>10</td>
<td>0+00:01:52</td>
<td>0+00:04:26</td>
<td>0+00:03:58</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Jobs</th>
<th>Steps</th>
<th>MinQueueTime</th>
<th>MinRealTime</th>
<th>MinCPUTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadl</td>
<td>1</td>
<td>4</td>
<td>0+00:00:01</td>
<td>0+00:02:49</td>
<td>0+00:02:44</td>
</tr>
<tr>
<td>user1</td>
<td>2</td>
<td>6</td>
<td>0+00:02:02</td>
<td>0+00:03:43</td>
<td>0+00:03:02</td>
</tr>
<tr>
<td>ALL</td>
<td>3</td>
<td>10</td>
<td>0+00:00:01</td>
<td>0+00:02:49</td>
<td>0+00:02:44</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Jobs</th>
<th>Steps</th>
<th>MaxQueueTime</th>
<th>MaxRealTime</th>
<th>MaxCPUTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadl</td>
<td>1</td>
<td>4</td>
<td>0+00:00:06</td>
<td>0+00:12:58</td>
<td>0+00:12:37</td>
</tr>
<tr>
<td>user1</td>
<td>2</td>
<td>6</td>
<td>0+00:06:21</td>
<td>0+00:03:48</td>
<td>0+00:03:07</td>
</tr>
<tr>
<td>ALL</td>
<td>3</td>
<td>10</td>
<td>0+00:06:21</td>
<td>0+00:12:58</td>
<td>0+00:12:37</td>
</tr>
</tbody>
</table>

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.

MinCPUtime  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,
Ilsummary

...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 Ilsummary command, the generated output you receive will resemble the listing below. In this sample, c271f2rp01.ppd.pok.ibm.com 16 is a parallel job consisting of one job step with six tasks.

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.

Figure 23. Output generated by Ilsummary -x -l command (Part 1 of 8)
Restart From Ckpt: no
Restart Same Nodes: no
Restart: no
Hold Job Until:
  Cmd: /bin/poe
  Args: /u/wilson/LL_PARA/ivp_cpu_110_120_sleep_130 -ilevel 6 -labelio yes
  Env: LANG=en_US; LOGIN=wilson; PATH= ...
  In: /dev/null
  Out: poe5_l.c271f2rp01.16.0.out
  Err: poe5_l.c271f2rp01.16.0.err
Initial Working Dir: /u/wilson/LL_PARA
  Requirements: (Arch == "R6000") && ((OpSys == "AIX51") || (OpSys == "AIX52"))
  Preferences: (Machine == { "c271f2rp01.ppd.pok.ibm.com" "c271f2rp02.ppd.pok.ibm.com" }) && (Feature == "ESSL")
  Step Type: General Parallel
  Min Processors: 2
  Max Processors: 2
  Alloc. Host Count: 2
  Allocated Host: c271f2rp01.ppd.pok.ibm.com
                   c271f2rp02.ppd.pok.ibm.com
  Node Usage: shared
  Notify User: wilson@c271f2rp01.ppd.pok.ibm.com
  Shell: /bin/ksh
LoadLeveler Group: No_Group
  Class: large
  Ckpt Hard Limit: undefined
  Ckpt Soft Limit: undefined
  Cpu Hard Limit: 03:30:30 (12630 seconds)
  Cpu Soft Limit: 02:00:00 (7200 seconds)
  Data Hard Limit: 5.200 gb (5583457484 bytes)
  Data Soft Limit: 4.100 gb (4402341478 bytes)
  Core Hard Limit: 8.000 gb (8589934592 bytes)
  Core Soft Limit: 5.500 gb (5905580032 bytes)
  File Hard Limit: 1.500 tb (1649267441664 bytes)
  File Soft Limit: 1.200 tb (1319413953331 bytes)

Figure 23. Output generated by llsrun summary -x -l command (Part 2 of 8)
ilsummary

Stack Hard Limit: 400.000 mb (419430400 bytes)
Stack Soft Limit: 200.000 mb (209715200 bytes)
Rss Hard Limit: 3.500 pb (3904649673949184 bytes)
Rss Soft Limit: 2.500 pb (2814795671065600 bytes)
Step Cpu Hard Limit: 2+12:45:00 (218700 seconds)
Step Cpu Soft Limit: 2+02:30:00 (181800 seconds)
Wall Ck Hard Limit: 01:40:00 (6000 seconds)
Wall Ck Soft Limit: 01:40:00 (6000 seconds)
Comment: Test job 1 of Parallel test suite 8.
Account: 99999
NQS Submit Queue:
NQS Query Queues:
Job Tracking Exit:
Job Tracking Args:
  Task geometry:
    Resources: ConsumableMemory(100.000 mb) ConsumableVirtualMemory(400.000 mb) spice2g6(2)
    Blocking: UNSPECIFIED
Adapter Requirement: (css0,LAPI,not_shared,US),(css0,MPI,shared,US)
Step Cpus: 0
Step Virtual Memory: 2.344 gb
Step Real Memory: 600.000 mb
Step Adapter Memory: 12.000 mb (12582912 bytes)
Large Page: N

------------------ Detail for c271f2rp01.ppd.pok.ibm.com.16.0 ------------------
  Running Host: c271f2rp01.ppd.pok.ibm.com
  Machine Speed: 1.000000
  Event: System
  Starter User Time: 00:00:00.000000
  Starter System Time: 00:00:00.000000
  Starter Total Time: 00:00:00.000000
  Starter maxrss: 0
  ...

Figure 23. Output generated by ilsummary -x -l command (Part 3 of 8)
Figure 23. Output generated by llvmsummary -x -I command (Part 4 of 8)
Figure 23. Output generated by llsummary -x -l command (Part 5 of 8)

Figure 23. Output generated by llsummary -x -l command (Part 6 of 8)
<table>
<thead>
<tr>
<th>Step nivcsw:</th>
<th>Event: Installation Defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Name:</td>
<td>user_event_1</td>
</tr>
<tr>
<td>Starter User Time:</td>
<td>00:00:00.060000</td>
</tr>
<tr>
<td>Starter System Time:</td>
<td>00:00:00.090000</td>
</tr>
<tr>
<td>Starter Total Time:</td>
<td>00:00:00.150000</td>
</tr>
<tr>
<td>Starter maxrss:</td>
<td>2540</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Starter nivcsw:</td>
<td>1600</td>
</tr>
<tr>
<td>Step User Time:</td>
<td>00:03:48.800000</td>
</tr>
<tr>
<td>Step System Time:</td>
<td>00:03:49.400000</td>
</tr>
<tr>
<td>Step Total Time:</td>
<td>00:03:49.400000</td>
</tr>
<tr>
<td>Step maxrss:</td>
<td>22820</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Step nivcsw:</td>
<td>324</td>
</tr>
<tr>
<td>Event: Installation Defined</td>
<td></td>
</tr>
<tr>
<td>Event Name:</td>
<td>user_event_2</td>
</tr>
<tr>
<td>Starter User Time:</td>
<td>00:00:00.060000</td>
</tr>
<tr>
<td>Starter System Time:</td>
<td>00:00:00.090000</td>
</tr>
<tr>
<td>Starter Total Time:</td>
<td>00:00:00.150000</td>
</tr>
<tr>
<td>Starter maxrss:</td>
<td>2540</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Starter nivcsw:</td>
<td>2173</td>
</tr>
<tr>
<td>Step User Time:</td>
<td>00:05:13.810000</td>
</tr>
<tr>
<td>Step System Time:</td>
<td>00:05:14.520000</td>
</tr>
<tr>
<td>Step Total Time:</td>
<td>00:05:14.520000</td>
</tr>
<tr>
<td>Step maxrss:</td>
<td>22820</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 23. Output generated by llsummary -x -l command (Part 7 of 8)*
For an explanation of these fields, see the description of the output fields for the long listing of the \texttt{llq} command.

\section*{Security}

Administrators and users can issue this command.
Chapter 16. 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” on page 246
- “Error Handling API” on page 285
- “Checkpointing API” on page 242
- “Submit API” on page 295
- “Data Access API” on page 247
- “Parallel Job API” on page 286
- “Workload Management API” on page 297
- “Query API” on page 291
- “User exits” on page 315

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 ...

© Copyright IBM Corp. 2001, 2003, 2004
Parameters

program
Is the name of the program that performs the account validation. The default is llacctval. The name you specify here must match the value specified on the ACCT_VALIDATION keyword in the configuration file.

user_name
Is the name of the user whose account number you want to validate.

user_group
Is the login group name of the user.

user_acct#
Is the account number specified by the user in the job command file.

acct1 acct2 ...
Are the account numbers obtained from the user stanza in the LoadLeveler administration file.

Description
llacctval is invoked from within the llsubmit command. If the return code is 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:

```c
#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:

```c
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.

```c
char *name
```

Specifies a character string identifying the event. This can be one of the following:

- An installation generated string that uses the command l cancelled
- 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:

```bash
llpjob history_file
```
Checkpointing API

Where `history_file` is a local or global history file.

Checkpointing API

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.

Linux notes

This API is not available in LoadLeveler for Linux.

This section describes routines used for checkpointing jobs running under LoadLeveler. "Step 14: Enable checkpointing" on page 392 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**

```cpp
extern "C"{void ckpt();}
```

**C syntax**

```c
void ckpt();
```

**FORTRAN syntax**

```fortran
call ckpt()
```

**ll_init_ckpt**

**Purpose**
Initiates a checkpoint from within a serial application.

**Library**
LoadLeveler API library `libllapi.a`.

**Syntax**

```c
#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

**Purpose**
Initiates a checkpoint on a specific job step.

**Library**
LoadLeveler API library libllapi.a

**Syntax**
```c
#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 `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`.  

Using and Administering LoadLeveler
**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
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.

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, classes, and machines. 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 248 for more information.
• Call ll_set_request to filter the objects you want to query. See "ll_set_request subroutine" on page 249 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 252 for more information.
  - Call ll_get_data to retrieve specific data from an object. See "ll_get_data subroutine" on page 257 for more information.
  - Call ll_next_obj to retrieve the next object in the list. See "ll_next_obj subroutine" on page 276 for more information.
Call *ll_free_objs to free the list of objects you received. See "ll_free_objs subroutine" on page 277 for more information.

*Call ll_deallocate to end the query. See "ll_deallocate subroutine" on page 277 for more information.

To see code that uses these subroutines, refer to "Examples of using the Data Access API" on page 278. For more information on LoadLeveler objects, see "Understanding the LoadLeveler job object model" on page 254.

**Il_query subroutine**

**Purpose**
The *Il_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**

```c
#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)
- CLASSES (to query information about job classes)
- CLUSTER (to query cluster information)
- WLMSTAT (to query AIX Workload Manager)

**Description**

*query_type is the input field for this subroutine.

This subroutine is used in conjunction with other data access subroutines to query information about job and machine objects. You must call *ll_query prior to using the other data access subroutines.

**Return values**

This subroutine returns a pointer to an LL_element object. The pointer is used by subsequent data access subroutine calls.

**Error values**

NULL  The subroutine was unable to create the appropriate pointer.

**Related information**

Subroutines: *ll_get_data, ll_set_request, ll_reset_request, ll_get_objs, ll_free_objs, ll_next_obj, ll_deallocate."
Il_set_request subroutine

Purpose
The Il_set_request subroutine determines the data requested during a subsequent Il_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 Il_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 Il_query subroutine.

query_flags
When query_type (in Il_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_PROCID
  Query by process ID of a task of a job step.
- 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 Il_query) is MACHINES, query_flags can be the following:
- QUERY_ALL
  Query all machines.
- QUERY_HOST
  Query by machine names.

When query_type (in Il_query) is CLASSES, query_flags can be the following:
- QUERY_ALL
  Query all classes.
- QUERY_CLASS
  Query by LoadLeveler class.

When query_type (in Il_query) is CLUSTER, query_flags can be the following:
Data Access API

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.

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_PROCID, the object_filter must contain a list with a single process ID of a task of a job step.
- If you specify QUERY_USER, the object_filter must contain a list of user IDs.
- 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 198.

Note: If you query a history file for job information, always specify ALL_DATA.

- If you specify JOBS and query_flags QUERY_PROCID you must 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 216.

- If you specify CLASSES, 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.
The QUERY_PROCID flag should not be used in combination with any other query_flags.

You can request certain combinations 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_set_request, ll_get_objs, ll_free_objs, ll_next_obj, ll_deallocate.

ll_set_request subroutine

Purpose
The ll_set_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.

Linux notes
A NULL pointer is returned and an error code set if:

- DCE_ENABLEMENT is TRUE
- SEC_ENABLEMENT is DCE or CTSEC
- SCHEDULER_TYPE is GANG
- pvm_root or NQS_DIR is specified

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:

```c
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:

<table>
<thead>
<tr>
<th>QUERY_ALL</th>
<th>negotiator (LL_CM), schedd (LL_SCHEDD), or history file (LL_HISTORY_FILE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_JOBID</td>
<td>negotiator (LL_CM) or schedd (LL_SCHEDD)</td>
</tr>
<tr>
<td>QUERY_STEPID</td>
<td>negotiator (LL_CM)</td>
</tr>
<tr>
<td>QUERY_PROCID</td>
<td>startd (LL_STARTD)</td>
</tr>
<tr>
<td>QUERY_USER</td>
<td>negotiator (LL_CM)</td>
</tr>
<tr>
<td>QUERY_GROUP</td>
<td>negotiator (LL_CM)</td>
</tr>
<tr>
<td>QUERY_CLASS</td>
<td>negotiator (LL_CM)</td>
</tr>
<tr>
<td>QUERY_HOST</td>
<td>negotiator (LL_CM)</td>
</tr>
<tr>
<td>QUERY_STARTDATE</td>
<td>history file (LL_HISTORY_FILE)</td>
</tr>
<tr>
<td>QUERY_ENDDATE</td>
<td>history file (LL_HISTORY_FILE)</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>QUERY_ALL</th>
<th>negotiator (LL_CM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_HOST</td>
<td>negotiator (LL_CM)</td>
</tr>
</tbody>
</table>

When `query_type` (in `ll_query`) is `CLASSES`, 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:

<table>
<thead>
<tr>
<th>QUERY_ALL</th>
<th>negotiator (LL_CM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_CLASS</td>
<td>negotiator (LL_CM)</td>
</tr>
</tbody>
</table>

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) |

---

*Data Access API*

Chapter 16. Application Programming Interfaces (APIs) 253
Data Access API

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.

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)
-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
-17 This release of LoadLeveler for Linux does not support DCE.
-18 This release of LoadLeveler for Linux does not support CTSEC.
-19 This release of LoadLeveler for Linux does not support GANG scheduling.
-20 This release of LoadLeveler for Linux does not support PVM.
-21 This release of LoadLeveler for Linux does not support NQS.

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 IL_get_data subroutine to access both attributes and connected objects. See “IL_get_data subroutine” on page 257 for more information.

The root of the job model is the Job object, as shown in Figure 24 on page 256. 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.
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 258 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.

### `ll_get_data` subroutine

Before you use this subroutine, make sure you are familiar with "Understanding the LoadLeveler job object model" on page 254.

**Purpose**
The `ll_get_data` subroutine returns data from a valid `LL_element`.

**Library**
LoadLeveler API library `libllapi.a`

**Syntax**
```c
#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 the location where you want the data stored. If the call returns a non-zero value an error has occurred and the contents of the location are undefined.

**Description**
`object` and `specification` are input fields, while `resulting_data_type` is an output field.

The `ll_get_data` subroutine of the data access API allows you to access LoadLeveler objects. The parameters of `ll_get_data` are a LoadLeveler object (`LL_element`), a specification that indicates what information about the object is being requested, and a pointer to the area where the information being requested should be stored.

If the specification indicates an attribute of the element that is passed in, the result pointer must be the address of a variable of the appropriate type, and must be initialized to NULL. The type returned by each specification is found in Table 15 on page 258. If the specification queries the connection to another object, the returned value is of type `LL_element`. You can use a subsequent `ll_get_data` call to query information about the new object.

The data type `char*` and any arrays of type `int` or `char` must be freed by the caller.

`LL_element` pointers cannot be freed by the caller.
Data Access API

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

<table>
<thead>
<tr>
<th>Object</th>
<th>Specification</th>
<th>Resulting Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adapter</td>
<td>LL_AdapterAvailWindowCount</td>
<td>int*</td>
<td>A pointer to an integer indicating the number of adapter windows not in use.</td>
</tr>
<tr>
<td>Adapter</td>
<td>LL_AdapterCommInterface</td>
<td>int*</td>
<td>A pointer to a string containing the adapter’s communication interface.</td>
</tr>
<tr>
<td>Adapter</td>
<td>LL_AdapterInterfaceAddress</td>
<td>int*</td>
<td>A pointer to a string containing the adapter’s interface IP address.</td>
</tr>
<tr>
<td>Adapter</td>
<td>LL_AdapterMaxWindowSize</td>
<td>int*</td>
<td>A pointer to the integer indicating the maximum allocatable adapter window memory.</td>
</tr>
<tr>
<td>Adapter</td>
<td>LL_AdapterMemory</td>
<td>int*</td>
<td>A pointer to the integer indicating the amount of total adapter memory.</td>
</tr>
<tr>
<td>Adapter</td>
<td>LL_AdapterMinWindowSize</td>
<td>int*</td>
<td>A pointer to the integer indicating the minimum allocatable adapter window memory.</td>
</tr>
<tr>
<td>Adapter</td>
<td>LL_AdapterName</td>
<td>int*</td>
<td>A pointer to a string containing the adapter name.</td>
</tr>
<tr>
<td>Adapter</td>
<td>LL_AdapterTotalWindowCount</td>
<td>int*</td>
<td>A pointer to the integer indicating the number of windows on the adapter.</td>
</tr>
<tr>
<td>AdapterUsage</td>
<td>LL_AdapterUsageDevice</td>
<td>char**</td>
<td>A pointer to a string containing the name of the adapter device being used.</td>
</tr>
<tr>
<td>AdapterUsage</td>
<td>LL_AdapterUsageMode</td>
<td>int*</td>
<td>A pointer to a string containing the mode used for css IP or US.</td>
</tr>
<tr>
<td>AdapterUsage</td>
<td>LL_AdapterUsageProtocol</td>
<td>int*</td>
<td>A pointer to a string containing the task’s protocol.</td>
</tr>
<tr>
<td>AdapterUsage</td>
<td>LL_AdapterUsageTag</td>
<td>char**</td>
<td>A pointer to a character string that indicates which network table the adapter usage is in. Adapter usages with the same tag are in the same switch table.</td>
</tr>
<tr>
<td>AdapterUsage</td>
<td>LL_AdapterUsageWindow</td>
<td>int*</td>
<td>A pointer to a string containing the adapter window assigned to the task.</td>
</tr>
</tbody>
</table>
Table 15. Specifications for ll_get_data subroutine (continued)

<table>
<thead>
<tr>
<th>Object</th>
<th>Specification</th>
<th>Resulting Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdapterUsage</td>
<td>LL_AdapterUsageWindowMemory</td>
<td>int*</td>
<td>A pointer to the integer indicating the number of bytes used by the adapter window.</td>
</tr>
<tr>
<td>AdapterReq</td>
<td>LL_AdapterReqCommLevel</td>
<td>int*</td>
<td>A pointer to the integer indicating the adapter's communication level.</td>
</tr>
<tr>
<td>AdapterReq</td>
<td>LL_AdapterReqUsage</td>
<td>int*</td>
<td>A pointer to the integer indicating the requested adapter usage. This integer will be one of the values defined in the Usage enum.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassName</td>
<td>char**</td>
<td>A pointer to a string containing the name of the class.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassPriority</td>
<td>int*</td>
<td>The class system priority</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassExcludeUsers</td>
<td>char***</td>
<td>A pointer to an array of strings containing users not permitted to use the class. The array ends with a NULL string.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassIncludeUsers</td>
<td>char***</td>
<td>A pointer to an array of strings containing users permitted to use the class. The array ends with a NULL string.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassExcludeGroups</td>
<td>char***</td>
<td>A pointer to an array of strings containing groups not permitted to use the class. The array ends with a NULL string.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassIncludeGroups</td>
<td>char***</td>
<td>A pointer to an array of strings containing groups permitted to use the class. The array ends with a NULL string.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassAdmin</td>
<td>char***</td>
<td>A pointer to an array of strings containing administrators for the class. The array ends with a NULL string.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassNqsClass</td>
<td>int*</td>
<td>Tells whether the class is an NQS gateway</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassNqsSubmit</td>
<td>char**</td>
<td>A pointer to a string containing the NQS queue to submit jobs.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassNqsQuery</td>
<td>char***</td>
<td>A pointer to an array of strings containing NQS queues to query about job dispatch. The array ends with a NULL string.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassMaxProcessors</td>
<td>int*</td>
<td>Lists the maximum number of processors for a parallel job step.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassMaxJobs</td>
<td>int*</td>
<td>Lists the maximum number of job steps that can run at any time.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassGetFirstResourceRequirement</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the first resource requirement.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassGetNextResourceRequirement</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the next resource requirement.</td>
</tr>
</tbody>
</table>
### Table 15. Specifications for ll_get_data subroutine (continued)

<table>
<thead>
<tr>
<th>Object</th>
<th>Specification</th>
<th>Resulting Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>LL_ClassComment</td>
<td>char**</td>
<td>A pointer to a string containing the class comment.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassCkptDir</td>
<td>char**</td>
<td>A pointer to a string containing the directory for checkpoint files.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassCkptTimeHardLimit</td>
<td>int64_t*</td>
<td>Specifies the checkpoint time hard limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassCkptTimeSoftLimit</td>
<td>int64_t*</td>
<td>Specifies the checkpoint time soft limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassWallClockLimitHard</td>
<td>int64_t*</td>
<td>Specifies the wall clock hard limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassWallClockLimitSoft</td>
<td>int64_t*</td>
<td>Specifies the wall clock soft limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassCpuStepLimitHard</td>
<td>int64_t*</td>
<td>Specifies the Hard Job_cpu_limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassCpuStepLimitSoft</td>
<td>int64_t*</td>
<td>Specifies the Soft Job_cpu_limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassCpuLimitHard</td>
<td>int64_t*</td>
<td>Specifies the cpu hard limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassCpuLimitSoft</td>
<td>int64_t*</td>
<td>Specifies the cpu soft limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassDataLimitHard</td>
<td>int64_t*</td>
<td>Specifies the data hard limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassDataLimitSoft</td>
<td>int64_t*</td>
<td>Specifies the data soft limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassCoreLimitHard</td>
<td>int64_t*</td>
<td>Specifies the core file hard limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassCoreLimitSoft</td>
<td>int64_t*</td>
<td>Specifies the core file soft limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassFileLimitHard</td>
<td>int64_t*</td>
<td>Specifies the file size hard limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassFileLimitSoft</td>
<td>int64_t*</td>
<td>Specifies the file size soft limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassStackLimitHard</td>
<td>int64_t*</td>
<td>Specifies the stack size hard limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassStackLimitSoft</td>
<td>int64_t*</td>
<td>Specifies the stack size soft limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassRssLimitHard</td>
<td>int64_t*</td>
<td>Specifies the resident set size hard limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassRssLimitSoft</td>
<td>int64_t*</td>
<td>Specifies the resident set size soft limit.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassNice</td>
<td>int*</td>
<td>Specifies the nice value.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassFreeSlots</td>
<td>int*</td>
<td>Specifies the number of available initiators.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassMaximumSlots</td>
<td>int*</td>
<td>Specifies the total number of configured initiators.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassConstraints</td>
<td>int*</td>
<td>Specifies whether values of Maximum and Free Slots are constrained by MAX_STARTERS and MAXJOBS</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassExecutionFactor</td>
<td>int*</td>
<td>Specifies the execution factor.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassMaxTotalTasks</td>
<td>int*</td>
<td>Specifies the value for Max_total_tasks.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassPreemptClass</td>
<td>char**</td>
<td>A pointer to a string containing the PREEMPT_CLASS rule.</td>
</tr>
<tr>
<td>Class</td>
<td>LL_ClassStartClass</td>
<td>char**</td>
<td>A pointer to a string containing the START_CLASS rule.</td>
</tr>
</tbody>
</table>
Table 15. Specifications for *ll_get_data* subroutine (continued)

<table>
<thead>
<tr>
<th>Object</th>
<th>Specification</th>
<th>Resulting Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>LL_ClassMaxProtocolInstances</td>
<td>int*</td>
<td>Specifies the maximum number of adapter windows per protocol per task.</td>
</tr>
<tr>
<td>Cluster</td>
<td>LL_ClusterGetFirstResource</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the first resource.</td>
</tr>
<tr>
<td>Cluster</td>
<td>LL_ClusterGetNextResource</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the next resource.</td>
</tr>
<tr>
<td>Cluster</td>
<td>LL_ClusterDefinedResources</td>
<td>char***</td>
<td>A pointer to an array containing the names of consumable resources defined in the cluster. The array ends with a NULL string.</td>
</tr>
<tr>
<td>Cluster</td>
<td>LL_ClusterDefinedResourceCount</td>
<td>int*</td>
<td>A pointer to an integer indicating the number of consumable resources defined in the cluster.</td>
</tr>
<tr>
<td>Cluster</td>
<td>LL_ClusterEnforcedResources</td>
<td>char***</td>
<td>A pointer to an array of characters indicating the number of enforced resources.</td>
</tr>
<tr>
<td>Cluster</td>
<td>LL_ClusterEnforcedResourceCount</td>
<td>int*</td>
<td>A pointer to an integer indicating the number of enforced resources.</td>
</tr>
<tr>
<td>Cluster</td>
<td>LL_ClusterEnforceSubmission</td>
<td>int*</td>
<td>A pointer to a boolean integer indicating resources required at time of submission.</td>
</tr>
<tr>
<td>Cluster</td>
<td>LL_ClusterSchedulerType</td>
<td>char**</td>
<td>A pointer to a string containing the scheduler type.</td>
</tr>
<tr>
<td>Cluster</td>
<td>LL_ClusterSchedulingResources</td>
<td>char***</td>
<td>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.</td>
</tr>
<tr>
<td>Cluster</td>
<td>LL_ClusterSchedulingResourceCount</td>
<td>int*</td>
<td>A pointer to an integer indicating the number of consumable resources considered by the scheduler for the cluster.</td>
</tr>
<tr>
<td>Credential</td>
<td>LL_CredentialGid</td>
<td>int*</td>
<td>A pointer to an integer containing the UNIX gid of the user submitting the job.</td>
</tr>
<tr>
<td>Credential</td>
<td>LL_CredentialGroupName</td>
<td>char**</td>
<td>A pointer to a string containing the UNIX group name of the user submitting the job.</td>
</tr>
<tr>
<td>Credential</td>
<td>LL_CredentialUid</td>
<td>int*</td>
<td>A pointer to an integer containing the UNIX uid of the person submitting the job.</td>
</tr>
<tr>
<td>Credential</td>
<td>LL_CredentialUserName</td>
<td>char**</td>
<td>A pointer to a string containing the user ID of the user submitting the job.</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageEventUsageCount</td>
<td>int*</td>
<td>Count of Event Usages</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageGetFirstEventUsage</td>
<td>LL_element*</td>
<td>First Event Usage</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageGetNextEventUsage</td>
<td>LL_element*</td>
<td>Next Event Usage</td>
</tr>
<tr>
<td>Object</td>
<td>Specification</td>
<td>Resulting Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------</td>
<td>---------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStarterIdrss64</td>
<td>int64_t*</td>
<td>Starter idrss value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStarterInblock64</td>
<td>int64_t*</td>
<td>Starter inblock value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStarterIsrss64</td>
<td>int64_t*</td>
<td>Starter isrss value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStarterIxrss64</td>
<td>int64_t*</td>
<td>Starter ixrss value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStarterMajflt64</td>
<td>int64_t*</td>
<td>Starter majflt value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStarterMaxrss64</td>
<td>int64_t*</td>
<td>Starter maxrss value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStarterMinflt64</td>
<td>int64_t*</td>
<td>Starter minflt value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStarterMsgrcv64</td>
<td>int64_t*</td>
<td>Starter msgrcv value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStarterMsgsnd64</td>
<td>int64_t*</td>
<td>Starter msgsnd value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStarterNivcsw64</td>
<td>int64_t*</td>
<td>Starter nivcsw value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStarterNsignals64</td>
<td>int64_t*</td>
<td>Starter nsignals value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStarterNswap64</td>
<td>int64_t*</td>
<td>Starter nswap value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStarterNvcsr64</td>
<td>int64_t*</td>
<td>Starter nvcsw value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStarterOublock64</td>
<td>int64_t*</td>
<td>Starter oublock value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStarterSystemTime64</td>
<td>int64_t*</td>
<td>Starter system time of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStarterUserTime64</td>
<td>int64_t*</td>
<td>Starter user time of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStepIdrss64</td>
<td>int64_t*</td>
<td>Step idrss value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStepInblock64</td>
<td>int64_t*</td>
<td>Step inblock value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStepIsrss64</td>
<td>int64_t*</td>
<td>Step isrss value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStepIxrss64</td>
<td>int64_t*</td>
<td>Step ixrss value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStepMajflt64</td>
<td>int64_t*</td>
<td>Step majflt value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStepMaxrss64</td>
<td>int64_t*</td>
<td>Step maxrss value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStepMinflt64</td>
<td>int64_t*</td>
<td>Step minflt value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStepMsgrcv64</td>
<td>int64_t*</td>
<td>Step msgrcv value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStepMsgsnd64</td>
<td>int64_t*</td>
<td>Step msgsnd value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStepNivcsw64</td>
<td>int64_t*</td>
<td>Step nivcsw value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStepNsignals64</td>
<td>int64_t*</td>
<td>Step nsignals value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStepNswap64</td>
<td>int64_t*</td>
<td>Step nswap value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStepNvcsr64</td>
<td>int64_t*</td>
<td>Step nvcsw value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStepOublock64</td>
<td>int64_t*</td>
<td>Step oublock value of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStepSystemTime64</td>
<td>int64_t*</td>
<td>Step system time of dispatch</td>
</tr>
<tr>
<td>DispUsage</td>
<td>LL_DispUsageStepUserTime64</td>
<td>int64_t*</td>
<td>Step user time of dispatch</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageEventId</td>
<td>int*</td>
<td>Event id</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageEventName</td>
<td>char**</td>
<td>Event name</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageEventTimestamp</td>
<td>int*</td>
<td>Event timestamp</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStarterIdrss64</td>
<td>int64_t*</td>
<td>Starter idrss value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStarterInblock64</td>
<td>int64_t*</td>
<td>Starter inblock value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStarterIsrss64</td>
<td>int64_t*</td>
<td>Starter isrss value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStarterIxrss64</td>
<td>int64_t*</td>
<td>Starter ixrss value of event</td>
</tr>
</tbody>
</table>
Table 15. Specifications for \texttt{ll\_get\_data} subroutine (continued)

<table>
<thead>
<tr>
<th>Object</th>
<th>Specification</th>
<th>Resulting Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStarterMajflt64</td>
<td>int64_t*</td>
<td>Starter majflt value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStarterMaxrss64</td>
<td>int64_t*</td>
<td>Starter maxrss value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStarterMinflt64</td>
<td>int64_t*</td>
<td>Starter minflt value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStarterMsgrcv64</td>
<td>int64_t*</td>
<td>Starter msgrcv value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStarterMsgsnd64</td>
<td>int64_t*</td>
<td>Starter msgsnd value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStarterNvcsw64</td>
<td>int64_t*</td>
<td>Starter nvcsw value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStarterNswap64</td>
<td>int64_t*</td>
<td>Starter nswap value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStarterOublock64</td>
<td>int64_t*</td>
<td>Starter oublock value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStarterSystemTime64</td>
<td>int64_t*</td>
<td>Starter system time of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStarterUserTime64</td>
<td>int64_t*</td>
<td>Starter user time of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStepIdrss64</td>
<td>int64_t*</td>
<td>Step idrss value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStepInblock64</td>
<td>int64_t*</td>
<td>Step inblock value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStepIsrss64</td>
<td>int64_t*</td>
<td>Step isrss value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStepIxrss64</td>
<td>int64_t*</td>
<td>Step ixrss value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStepMajflt64</td>
<td>int64_t*</td>
<td>Step majflt value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStepMaxrss64</td>
<td>int64_t*</td>
<td>Step maxrss value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStepMinflt64</td>
<td>int64_t*</td>
<td>Step minflt value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStepMsgrcv64</td>
<td>int64_t*</td>
<td>Step msgrcv value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStepMsgsnd64</td>
<td>int64_t*</td>
<td>Step msgsnd value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStepNvcsw64</td>
<td>int64_t*</td>
<td>Step nvcsw value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStepNsignals64</td>
<td>int64_t*</td>
<td>Step nsignals value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStepNswap64</td>
<td>int64_t*</td>
<td>Step nswap value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStepNvcs6w64</td>
<td>int64_t*</td>
<td>Step nvcs6w value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStepOublock64</td>
<td>int64_t*</td>
<td>Step oublock value of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStepSystemTime64</td>
<td>int64_t*</td>
<td>Step system time of event</td>
</tr>
<tr>
<td>EventUsage</td>
<td>LL_EventUsageStepUserTime64</td>
<td>int64_t*</td>
<td>Step user time of event</td>
</tr>
<tr>
<td>Job</td>
<td>LL_JobCredential</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the job credential.</td>
</tr>
<tr>
<td>Job</td>
<td>LL_JobGetFirstStep</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the first step of the job, to be used in subsequent \texttt{ll_get_data} calls.</td>
</tr>
<tr>
<td>Job</td>
<td>LL_JobGetNextStep</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the next step.</td>
</tr>
<tr>
<td>Job</td>
<td>LL_JobName</td>
<td>char**</td>
<td>A pointer to a character string containing the job name.</td>
</tr>
<tr>
<td>Job</td>
<td>LL_JobStepCount</td>
<td>int*</td>
<td>A pointer to an integer indicating the number of steps connected to the job.</td>
</tr>
</tbody>
</table>
| Job               | LL\_JobStepType         | int*               | A pointer to an integer indicating the type of job, which can be \texttt{INTERACTIVE\_JOB} or \texttt{BATCH\_JOB}.
### Data Access API

<table>
<thead>
<tr>
<th>Object</th>
<th>Specification</th>
<th>Resulting Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job</td>
<td>LL_JobSubmitHost</td>
<td>char**</td>
<td>A pointer to a character string containing the name of the host machine from which the job was submitted.</td>
</tr>
<tr>
<td>Job</td>
<td>LL_JobSubmitTime</td>
<td>time_t*</td>
<td>A pointer to the time_t structure indicating when the job was submitted.</td>
</tr>
<tr>
<td>Job</td>
<td>LL_JobVersionNum</td>
<td>int*</td>
<td>A pointer to an integer indicating the job’s version number.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineAdapterList</td>
<td>char***</td>
<td>A pointer to an array containing the list of adapters associated with the machine. The array ends with a NULL string.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineArchitecture</td>
<td>char**</td>
<td>A pointer to a string containing the machine architecture.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineAvailableClassList</td>
<td>char***</td>
<td>A pointer to an array containing the currently available job classes defined on the machine. The array ends with a NULL string.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineConfiguredClassList</td>
<td>char***</td>
<td>A pointer to an array containing the initiators on the machine. The array ends with a NULL string.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineContinueExpr</td>
<td>char**</td>
<td>A pointer to a string containing the machine’s continue control expression.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineCPUs</td>
<td>int*</td>
<td>A pointer to an integer containing the number of CPUs on the machine.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineDisk</td>
<td>int*</td>
<td>A pointer to an integer indicating the disk space in KBs in the machine’s execute directory.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineDisk64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the disk space in KBs in the machine’s execute directory.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineDrainingClassList</td>
<td>char***</td>
<td>A pointer to an array containing the draining class list on the machine. The array ends with a NULL string.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineDrainClassList</td>
<td>char***</td>
<td>A pointer to an array containing the drain class list on the machine. The array ends with a NULL string.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineFeatureList</td>
<td>char***</td>
<td>A pointer to an array containing the features defined on the machine. The array ends with a NULL string.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineFreeRealMemory</td>
<td>int*</td>
<td>A pointer to an integer indicating the amount of free real memory in MBs on the machine.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineFreeRealMemory64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the amount of free real memory in MBs on the machine.</td>
</tr>
<tr>
<td>Object</td>
<td>Specification</td>
<td>Resulting Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineGetFirstAdapter</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the machine’s first adapter.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineGetFirstResource</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the machine’s first resource.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineGetNextAdapter</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the machine’s next adapter.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineGetNextResource</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the machine’s next resource.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineKbddIdle</td>
<td>int*</td>
<td>A pointer to an integer indicating the number of seconds since the kbdd daemon detected keyboard mouse activity.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineKillExpr</td>
<td>char**</td>
<td>A pointer to a string containing the machine’s kill control expression.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineLargePageSize64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the size of the machine’s Large Page.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineLargePageCount64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the number of Large Pages defined on the machine.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineLargePageFree64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the number of Large Pages free.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineLoadAverage</td>
<td>double*</td>
<td>A pointer to a double containing the load average on the machine.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineMaxTasks</td>
<td>int*</td>
<td>A pointer to an integer indicating the maximum number of tasks this machine can run at one time.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineMachineMode</td>
<td>char**</td>
<td>A pointer to a string containing the configured machine mode.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineName</td>
<td>char**</td>
<td>A pointer to a string containing the machine name.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineOperatingSystem</td>
<td>char**</td>
<td>A pointer to a string containing the operating system on the machine.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachinePagesFreed</td>
<td>int*</td>
<td>A pointer to an integer indicating the number of pages freed per second by the page replacement algorithm.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachinePagesFreed64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the number of pages freed per second by the page replacement algorithm.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachinePagesPagedIn</td>
<td>int*</td>
<td>A pointer to an integer indicating the number of pages paged in per second from paging space.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachinePagesPagedIn64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the number of pages paged in per second from paging space.</td>
</tr>
</tbody>
</table>
Table 15. Specifications for ll_get_data subroutine (continued)

<table>
<thead>
<tr>
<th>Object</th>
<th>Specification</th>
<th>Resulting Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine</td>
<td>LL_MachinePagesPagedOut</td>
<td>int*</td>
<td>A pointer to an integer indicating the number of pages paged out per second to paging space.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachinePagesPagedOut64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the number of pages paged out per second to paging space.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachinePagesScanned</td>
<td>int*</td>
<td>A pointer to an integer indicating the number of pages scanned per second by the page replacement algorithm.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachinePagesScanned64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the number of pages scanned per second by the page replacement algorithm.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachinePoolList</td>
<td>int**</td>
<td>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.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachinePoolListSize</td>
<td>int*</td>
<td>A pointer to an integer indicating the number of pools configured for the machine.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineRealMemory</td>
<td>int*</td>
<td>A pointer to an integer indicating the physical memory in MBs on the machine.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineRealMemory64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the physical memory in MBs on the machine.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineScheddRunningJobs</td>
<td>int*</td>
<td>A pointer to an integer indicating a list of the running jobs assigned to schedd.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineScheddState</td>
<td>int*</td>
<td>A pointer to an integer indicating the machine’s schedd state.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineScheddTotalJobs</td>
<td>int*</td>
<td>A pointer to an integer indicating the total number of jobs assigned to the schedd.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineSpeed</td>
<td>double*</td>
<td>A pointer to a double containing the configured speed of the machine.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineStartExpr</td>
<td>char**</td>
<td>A pointer to a string containing the machine’s start control expression.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineStartdRunningJobs</td>
<td>int*</td>
<td>A pointer to an integer containing the number of running jobs known by the startdd daemon.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineStartdState</td>
<td>char**</td>
<td>A pointer to a string containing the state of the startdd daemon.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineStepList</td>
<td>char***</td>
<td>A pointer to an array containing the steps running on the machine. The array ends with a NULL string.</td>
</tr>
</tbody>
</table>
Table 15. Specifications for ll_get_data subroutine (continued)

<table>
<thead>
<tr>
<th>Object</th>
<th>Specification</th>
<th>Resulting Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine</td>
<td>LL_MachineSuspendExpr</td>
<td>char**</td>
<td>A pointer to a string containing the machine’s suspend control expression.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineTimeStamp</td>
<td>time_t*</td>
<td>A pointer to a time_t structure indicating the time the machine last reported to the negotiator.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineVacateExpr</td>
<td>char**</td>
<td>A pointer to a string containing the machine’s vacate control expression.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineVirtualMemory</td>
<td>int*</td>
<td>A pointer to an integer containing the free swap space in KBs on the machine.</td>
</tr>
<tr>
<td>Machine</td>
<td>LL_MachineVirtualMemory64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the free swap space in KBs on the machine.</td>
</tr>
<tr>
<td>MachUsage</td>
<td>LL_MachUsageDispUsageCount</td>
<td>int*</td>
<td>Count of Dispatch Usages</td>
</tr>
<tr>
<td>MachUsage</td>
<td>LL_MachUsageGetFirstDispUsage</td>
<td>LL_element*</td>
<td>First Dispatch Usage</td>
</tr>
<tr>
<td>MachUsage</td>
<td>LL_MachUsageGetNextDispUsage</td>
<td>LL_element*</td>
<td>Next Dispatch Usage</td>
</tr>
<tr>
<td>MachUsage</td>
<td>LL_MachUsageMachineName</td>
<td>char**</td>
<td>Machine name</td>
</tr>
<tr>
<td>MachUsage</td>
<td>LL_MachUsageMachineSpeed</td>
<td>double*</td>
<td>Machine speed</td>
</tr>
<tr>
<td>Node</td>
<td>LL_NodeGetFirstTask</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the first task for this node.</td>
</tr>
<tr>
<td>Node</td>
<td>LL_NodeGetNextTask</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the next task for this node.</td>
</tr>
<tr>
<td>Node</td>
<td>LL_NodeInitiatorCount</td>
<td>int*</td>
<td>A pointer to an integer indicating the number of tasks running on the node.</td>
</tr>
<tr>
<td>Node</td>
<td>LL_NodeMaxInstances</td>
<td>int*</td>
<td>A pointer to an integer indicating the maximum number of machines requested.</td>
</tr>
<tr>
<td>Node</td>
<td>LL_NodeMinInstances</td>
<td>int*</td>
<td>A pointer to an integer indicating the minimum number of machines requested.</td>
</tr>
<tr>
<td>Node</td>
<td>LL_NodeRequirements</td>
<td>char**</td>
<td>A pointer to a string containing the node requirements.</td>
</tr>
<tr>
<td>Node</td>
<td>LL_NodeTaskCount</td>
<td>int*</td>
<td>A pointer to an integer indicating the different types of tasks running on the node.</td>
</tr>
<tr>
<td>Resource</td>
<td>LL_ResourceAvailableValue</td>
<td>int*</td>
<td>A pointer to an integer indicating the value of available resources.</td>
</tr>
<tr>
<td>Resource</td>
<td>LL_ResourceAvailableValue64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the value of available resources.</td>
</tr>
<tr>
<td>Resource</td>
<td>LL_ResourceName</td>
<td>char**</td>
<td>A pointer to a string containing the resource name.</td>
</tr>
<tr>
<td>Resource</td>
<td>LL_ResourceInitialValue</td>
<td>int*</td>
<td>A pointer to an integer indicating the initial resource value.</td>
</tr>
</tbody>
</table>
**Table 15. Specifications for ll_get_data subroutine (continued)**

<table>
<thead>
<tr>
<th>Object</th>
<th>Specification</th>
<th>Resulting Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource</td>
<td>LL_ResourceInitialValue64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the initial resource value.</td>
</tr>
<tr>
<td>ResourceReq</td>
<td>LL_ResourceRequirementName</td>
<td>char**</td>
<td>A pointer to a string containing the resource requirement name.</td>
</tr>
<tr>
<td>ResourceReq</td>
<td>LL_ResourceRequirementValue</td>
<td>int*</td>
<td>A pointer to an integer indicating the value of the resource requirement.</td>
</tr>
<tr>
<td>ResourceReq</td>
<td>LL_ResourceRequirementValue64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the value of the resource requirement.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepAccountNumber</td>
<td>char**</td>
<td>A pointer to a string containing the account number specified by the user submitting the job.</td>
</tr>
</tbody>
</table>
| Step        | LL_StepBlocking                          | int*                | A pointer to an integer representing blocking as specified by the user in the job command file.  
|             |                                          |                     | • Returns -1 if unlimited is specified  
|             |                                          |                     | • Returns 0 if blocking is unspecified |
| Step        | LL_StepClassSystemPriority              | int*                | A pointer to an integer indicating the class priority of the job step.      |
| Step        | LL_StepCheckpointable                    | int*                | A pointer to an integer indicating if checkpointing was enabled via the checkpoint keyword (0=disabled, 1=enabled). |
| Step        | LL_StepCheckpointing                     | Boolean             | If True, 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_StepCkptGoodElapsedTime              | 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 restart_from_ckpt keyword (0= no, 1= yes). |
### Table 15. Specifications for ll_get_data subroutine (continued)

<table>
<thead>
<tr>
<th>Object</th>
<th>Specification</th>
<th>Resulting Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>LL_StepCkptRestartSameNodes</td>
<td>int*</td>
<td>A pointer to a string indicating the value specified by the user for the <code>restart_on_same_nodes</code> keyword (0= no, 1= yes).</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCkptTimeHardLimit</td>
<td>int*</td>
<td>A pointer to an integer indicating the hard limit set by the user in the <code>ckpt_time_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCkptTimeHardLimit64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the hard limit set by the user in the <code>ckpt_time_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCkptTimeSoftLimit</td>
<td>int*</td>
<td>A pointer to an integer indicating the soft limit set by the user in <code>ckpt_time_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCkptTimeSoftLimit64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the soft limit set by the user in <code>ckpt_time_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepComment</td>
<td>char**</td>
<td>A pointer to a string indicating the comment specified by the user submitting the job.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCompletionCode</td>
<td>int*</td>
<td>A pointer to an integer indicating the completion code of the step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCompletionDate</td>
<td>time_t*</td>
<td>A pointer to a <code>time_t</code> structure indicating the completion date of the step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCoreLimitHard</td>
<td>int*</td>
<td>A pointer to an integer indicating the core hard limit set by the user in the <code>core_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCoreLimitHard64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the core hard limit set by the user in the <code>core_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCoreLimitSoft</td>
<td>int*</td>
<td>A pointer to an integer indicating the core soft limit set by the user in the <code>core_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCoreLimitSoft64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the core soft limit set by the user in the <code>core_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCpuLimitHard</td>
<td>int*</td>
<td>A pointer to an integer indicating the CPU hard limit set by the user in the <code>cpu_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCpuLimitHard64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the CPU hard limit set by the user in the <code>cpu_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCpuLimitSoft</td>
<td>int*</td>
<td>A pointer to an integer indicating the CPU soft limit set by the user in the <code>cpu_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCpuLimitSoft64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the CPU soft limit set by the user in the <code>cpu_limit</code> keyword.</td>
</tr>
<tr>
<td>Object</td>
<td>Specification</td>
<td>Resulting Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCpuStepLimitHard</td>
<td>int*</td>
<td>A pointer to an integer indicating the CPU step hard limit set by the user</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>in the <code>job_cpu_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCpuStepLimitHard64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the CPU step hard limit set by the user in the <code>job_cpu_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCpuStepLimitSoft</td>
<td>int*</td>
<td>A pointer to an integer indicating the CPU step soft limit set by the user</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>in the <code>job_cpu_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepCpuStepLimitSoft64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the CPU step soft limit set by the user in the <code>job_cpu_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepDataLimitHard</td>
<td>int*</td>
<td>A pointer to an integer indicating the data hard limit set by the user</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>in the <code>data_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepDataLimitHard64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the data hard limit set by the user in the <code>data_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepDataLimitSoft</td>
<td>int*</td>
<td>A pointer to an integer indicating the data soft limit set by the user</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>in the <code>data_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepDataLimitSoft64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the data soft limit set by the user in the <code>data_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepDispatchTime</td>
<td>time_t*</td>
<td>A pointer to a <code>time_t</code> structure indicating the time the negotiator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>dispatched the job.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepEnvironment</td>
<td>char**</td>
<td>A pointer to a string containing the environment variables set by the user</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>in the executable.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepErrorFile</td>
<td>char**</td>
<td>A pointer to a string containing the standard error file name used by the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>executable.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepExecSize</td>
<td>int*</td>
<td>A pointer to an integer indicating the executable size.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepExecutionFactor</td>
<td>int*</td>
<td>A pointer to an integer indicating the execution_factor of the job step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepFileLimitHard</td>
<td>int*</td>
<td>A pointer to an integer indicating the file hard limit set by the user in the <code>file_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepFileLimitHard64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the file hard limit set by the user in the <code>file_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepFileLimitSoft</td>
<td>int*</td>
<td>A pointer to an integer indicating the file soft limit set by the user in the <code>file_limit</code> keyword.</td>
</tr>
</tbody>
</table>
Table 15. Specifications for ll_get_data subroutine (continued)

<table>
<thead>
<tr>
<th>Object</th>
<th>Specification</th>
<th>Resulting Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>LL_StepFileLimitSoft64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the file soft limit set by the user in the file_limit keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepGetFirstAdapterReq</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the first adapter requirement.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepGetFirstMachine</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the first machine in the step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepGetFirstMachUsage</td>
<td>LL_element*</td>
<td>First Mach Usage</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepGetFirstNode</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the first node of the step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepGetFirstSwitchTable</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the first switch table for this step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepGetMasterTask</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the master task of the step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepGetNextAdapterReq</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the next adapter requirement.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepGetNextMachine</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the next machine of the step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepGetNextMachUsage</td>
<td>LL_element*</td>
<td>Next Mach Usage of step</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepGetNextNode</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the next node of the step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepGetNextSwitchTable</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the next switch table for this step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepGroupSystemPriority</td>
<td>int*</td>
<td>A pointer to an integer indicating the group priority of a job step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepHoldType</td>
<td>int*</td>
<td>A pointer to an integer indicating the hold state of the step (user, system, etc). The value returned is in the HoldType enum.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepHostList</td>
<td>char***</td>
<td>A pointer to an array containing the list of hosts in the host_list file associated with the step. The array ends with a null string.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepID</td>
<td>char**</td>
<td>A pointer to a string containing the ID of the step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepImageSize</td>
<td>int*</td>
<td>A pointer to an integer indicating the image size of the executable.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepImageSize64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the image size of the executable.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepInputFile</td>
<td>char**</td>
<td>A pointer to a string containing the standard input file name used by the executable.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepIwd</td>
<td>char**</td>
<td>A pointer to a string containing the initial working directory name used by the executable.</td>
</tr>
</tbody>
</table>
### Data Access API

<table>
<thead>
<tr>
<th>Object</th>
<th>Specification</th>
<th>Resulting Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>LL_StepJobClass</td>
<td>char**</td>
<td>A pointer to a string containing the class of the step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepLargePage</td>
<td>char**</td>
<td>A pointer to a string containing the Large Page level of support associated with the job step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepLoadLevelerGroup</td>
<td>char**</td>
<td>A pointer to a string containing the name of the LoadLeveler group specified by the step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepMachineCount</td>
<td>int*</td>
<td>A pointer to an integer indicating the number of machines assigned to the step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepMachUsageCount</td>
<td>int*</td>
<td>Count of Machine Usages</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepMessages</td>
<td>char**</td>
<td>A pointer to a string containing a list of messages from LL.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepName</td>
<td>char**</td>
<td>A pointer to a string containing the name of the step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepNodeCount</td>
<td>int*</td>
<td>A pointer to an integer indicating the number of node objects associated with the step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepNodeUsage</td>
<td>int*</td>
<td>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.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepOutputFile</td>
<td>char**</td>
<td>A pointer to a character string containing the standard output file name used by the executable.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepParallelMode</td>
<td>int*</td>
<td>A pointer to an integer indicating the mode of the step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepPriority</td>
<td>int*</td>
<td>A pointer to an integer indicating the priority of the step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepQueueSystemPriority</td>
<td>int*</td>
<td>A pointer to an integer indicating the adjusted system priority of the job step. Only the CM has the current value for LL_StepQueueSystemPriority.</td>
</tr>
</tbody>
</table>
| 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.  
• 1 indicates yes 
• 0 indicates no. |
| Step               | LL_StepRssLimitHard | int*            | A pointer to an integer indicating the RSS hard limit set by the user in the rss_limit keyword. |
| Step               | LL_StepRssLimitHard64 | int64_t*       | A pointer to a 64-bit integer indicating the RSS hard limit set by the user in the rss_limit keyword. |
Table 15. Specifications for `ll_get_data` subroutine (continued)

<table>
<thead>
<tr>
<th>Object</th>
<th>Specification</th>
<th>Resulting Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>LL_StepRssLimitSoft</td>
<td>int*</td>
<td>A pointer to an integer indicating the RSS soft limit set by the user in the <code>rss_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepRssLimitSoft64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the RSS soft limit set by the user in the <code>rss_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepShell</td>
<td>char**</td>
<td>A pointer to a character string containing the shell name used by the executable.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStackTraceHard</td>
<td>int*</td>
<td>A pointer to an integer indicating the stack hard limit set by the user in the <code>stack_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStackTraceHard64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the stack hard limit set by the user in the <code>stack_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStackTraceSoft</td>
<td>int*</td>
<td>A pointer to an integer indicating the stack soft limit set by the user in the <code>stack_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStackTraceSoft64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the stack soft limit set by the user in the <code>stack_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStartCount</td>
<td>int*</td>
<td>A pointer to an integer indicating the number of times the step has been started.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStartDate</td>
<td>time_t*</td>
<td>A pointer to a <code>time_t</code> structure indicating the value the user specified in the <code>startdate</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStarterIdrss64</td>
<td>int64_t*</td>
<td>Starter idrss value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStarterInblock64</td>
<td>int64_t*</td>
<td>Starter inblock value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStarterIrss64</td>
<td>int64_t*</td>
<td>Starter irrss value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStarterIxrss64</td>
<td>int64_t*</td>
<td>Starter ixrss value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStarterMajflt64</td>
<td>int64_t*</td>
<td>Starter majflt value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStarterMaxrss64</td>
<td>int64_t*</td>
<td>Starter maxrss value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStarterMinflt64</td>
<td>int64_t*</td>
<td>Starter minflt value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStarterMsgrcv64</td>
<td>int64_t*</td>
<td>Starter msgrcv value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStarterMsgsnd64</td>
<td>int64_t*</td>
<td>Starter msgsnd value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStarterNivcs64</td>
<td>int64_t*</td>
<td>Starter nivcs value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStarterNsignals64</td>
<td>int64_t*</td>
<td>Starter nsignals value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStarterNswap64</td>
<td>int64_t*</td>
<td>Starter nswap value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStarterNvcs64</td>
<td>int64_t*</td>
<td>Starter nvcs value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStarterOublock64</td>
<td>int64_t*</td>
<td>Starter oublock value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStarterSystemTime64</td>
<td>int64_t*</td>
<td>Starter system time</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStarterUserTime64</td>
<td>int64_t*</td>
<td>Starter user time</td>
</tr>
<tr>
<td>Object</td>
<td>Specification</td>
<td>Resulting Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepState</td>
<td>int*</td>
<td>A pointer to an integer indicating the state of the Step (Idle, Pending, Starting, etc.). The value returned is in the StepState enum.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStepIdrss64</td>
<td>int64_t*</td>
<td>Step idrss value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStepInblock64</td>
<td>int64_t*</td>
<td>Step inblock value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStepIxsrss64</td>
<td>int64_t*</td>
<td>Step ixrss value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStepMajflt64</td>
<td>int64_t*</td>
<td>Step majflt value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStepMinflt64</td>
<td>int64_t*</td>
<td>Step minflt value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStepMaxrss64</td>
<td>int64_t*</td>
<td>Step maxrss value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStepMsgrcv64</td>
<td>int64_t*</td>
<td>Step msgrcv value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStepNivcsw64</td>
<td>int64_t*</td>
<td>Step ivcsw value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStepNsignals64</td>
<td>int64_t*</td>
<td>Step signals value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStepNswap64</td>
<td>int64_t*</td>
<td>Step nswap value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStepNvcsw64</td>
<td>int64_t*</td>
<td>Step nvcsw value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStepOublock64</td>
<td>int64_t*</td>
<td>Step oublock value</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStepSystemTime64</td>
<td>int64_t*</td>
<td>Step system time</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepStepUserTime64</td>
<td>int64_t*</td>
<td>Step user time</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepSystemPriority</td>
<td>int*</td>
<td>A pointer to an integer indicating the overall system priority of the job step. Only the CM has the current value for LL_StepSystemPriority.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepTaskGeometry</td>
<td>char**</td>
<td>A pointer to a string containing the values specified in the task.geometry statement by the user in the job command file. The syntax is the same as specified in the statement , [(task id, task id, ...)(task id, task id, ...)]...]. If unspecified, a null string is returned.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepTaskInstanceCount</td>
<td>int*</td>
<td>A pointer to an integer indicating the number of task instances in the step. This is only available from the schedd daemon.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepTasksPerNodeRequested</td>
<td>int*</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 15. Specifications for `ll_get_data` subroutine (continued)

<table>
<thead>
<tr>
<th>Object</th>
<th>Specification</th>
<th>Resulting Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>LL_StepTotalNodesRequested</td>
<td>char**</td>
<td>A pointer to a string containing the values specified by the user in the job command file node statement. The syntax is the same as specified in the statement, [min],[max], where min contains the minimum number of nodes requested and max contains the maximum nodes requested. If unspecified, a null string is returned.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepTotalTasksRequested</td>
<td>int*</td>
<td>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.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepUserSystemPriority</td>
<td>int*</td>
<td>A pointer to an integer indicating the user system priority of the job step.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepWallClockLimitHard</td>
<td>int*</td>
<td>A pointer to an integer indicating the wall clock hard limit set by the user in the <code>wall_clock_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepWallClockLimitHard64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the wall clock hard limit set by the user in the <code>wall_clock_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepWallClockLimitSoft</td>
<td>int*</td>
<td>A pointer to an integer indicating the wall clock soft limit set by the user in the <code>wall_clock_limit</code> keyword.</td>
</tr>
<tr>
<td>Step</td>
<td>LL_StepWallClockLimitSoft64</td>
<td>int64_t*</td>
<td>A pointer to a 64-bit integer indicating the wall clock soft limit set by the user in the <code>wall_clock_limit</code> keyword.</td>
</tr>
<tr>
<td>Task</td>
<td>LL_TaskExecutable</td>
<td>char**</td>
<td>A pointer to a string containing the name of the executable.</td>
</tr>
<tr>
<td>Task</td>
<td>LL_TaskExecutableArguments</td>
<td>char**</td>
<td>A pointer to a string containing the arguments passed by the user in the executable.</td>
</tr>
<tr>
<td>Task</td>
<td>LL_TaskGetFirstResourceRequirement</td>
<td>LL_element</td>
<td>A pointer to the element associated with the first resource requirement.</td>
</tr>
<tr>
<td>Task</td>
<td>LL_TaskGetFirstTaskInstance</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the first task instance.</td>
</tr>
<tr>
<td>Task</td>
<td>LL_TaskGetNextResourceRequirement</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the next resource requirement.</td>
</tr>
<tr>
<td>Task</td>
<td>LL_TaskGetNextTaskInstance</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the next task instance.</td>
</tr>
<tr>
<td>Task</td>
<td>LL_TaskIsMaster</td>
<td>int*</td>
<td>A pointer to an integer, where I indicates master task.</td>
</tr>
<tr>
<td>Task</td>
<td>LL_TaskTaskInstanceCount</td>
<td>int*</td>
<td>A pointer to an integer indicating the number of task instances.</td>
</tr>
<tr>
<td>Task Instance</td>
<td>LL_TaskInstanceAdapterCount</td>
<td>int*</td>
<td>A pointer to the integer indicating the number of adapters.</td>
</tr>
</tbody>
</table>
Table 15. Specifications for ll_get_data subroutine (continued)

<table>
<thead>
<tr>
<th>Object</th>
<th>Specification</th>
<th>Resulting Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Instance</td>
<td>LL_TaskInstanceGetFirstAdapter</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the first adapter.</td>
</tr>
<tr>
<td>Task Instance</td>
<td>LL_TaskInstanceGetFirstAdapterUsage</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the first adapter usage.</td>
</tr>
<tr>
<td>Task Instance</td>
<td>LL_TaskInstanceGetNextAdapter</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the next adapter.</td>
</tr>
<tr>
<td>Task Instance</td>
<td>LL_TaskInstanceGetNextAdapterUsage</td>
<td>LL_element*</td>
<td>A pointer to the element associated with the next adapter usage.</td>
</tr>
<tr>
<td>Task Instance</td>
<td>LL_TaskInstanceMachineName</td>
<td>char**</td>
<td>A pointer to the string indicating the machine assigned to a task.</td>
</tr>
<tr>
<td>Task Instance</td>
<td>LL_TaskInstanceTaskID</td>
<td>int*</td>
<td>A pointer to the integer indicating the task ID.</td>
</tr>
<tr>
<td>WlmStat</td>
<td>LL_WlmStatCpuSnapshotUsage</td>
<td>int*</td>
<td>A pointer to CPU usage obtained from the AIX Workload Manager.</td>
</tr>
<tr>
<td>WlmStat</td>
<td>LL_WlmStatCpuTotalUsage</td>
<td>int64_t*</td>
<td>A pointer to total CPU usage obtained from the AIX Workload Manager.</td>
</tr>
<tr>
<td>WlmStat</td>
<td>LL_WlmStatMemorySnapshotUsage</td>
<td>int*</td>
<td>A pointer to real memory usage obtained from the AIX Workload Manager.</td>
</tr>
<tr>
<td>WlmStat</td>
<td>LL_WlmStatMemoryHighWater</td>
<td>int64_t*</td>
<td>A pointer to real memory high water mark obtained from the AIX Workload Manager.</td>
</tr>
</tbody>
</table>

**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.

Il_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.

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.

Il_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.
Data Access API

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
main(int argc, char *argv[]) {
    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 25. Obtaining machine, job, and cluster information with the Data Access API (Part 1 of 4)
printf("Startd State: %s\n", startd_state);
if (strncmp(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 25. 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.

```c
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 25. Obtaining machine, job, and cluster information with the Data Access API (Part 3 of 4)
The following example shows how LoadLeveler's Data Access API can be used to extract job accounting information saved in a history file.

```c
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);
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);
}
}
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(" 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(" 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 25. 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 *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.
        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("*************************************************
    Number of job objects returned = %d\n",
    obj_count);
    printf("*************************************************
    
    /* Loop through the job objects. */
    job_count = 0;
    while (job) {
        job_count++;;
        printf("Job number = %d\n", job_count);
        
    /* Loop through the job step objects. */

Figure 26. 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(" Step number = %d\n", step_count);
    ll_get_data(step, LL StepID, &str_data);
    ll_get_data(step, LL_StepID, &str_data);
    ll_get_data(step, LL StepImageSize, &int_data);
    ll_get_data(step, LL StepImageSize64, &int64_data);
    ll_get_data(step, LL_StepCpuLimitHard, &int_data);
    ll_get_data(step, LL_StepCpuLimitHard64, &int64_data);
    ll_get_data(step, LL_StepCpuLimitSoft, &int_data);
    ll_get_data(step, LL_StepCpuLimitSoft64, &int64_data);
    ll_get_data(step, LL_StepDataLimitHard64, &int64_data);
    ll_get_data(step, LL_StepDataLimitSoft64, &int64_data);
    ll_get_data(step, LL_StepStepUserTime64, &int64_data);
    ll_get_data(step, LL_StepStepSystemTime64, &int64_data);
    rc = ll_get_data(step, LL StepGetFirstMachUsage, &mach_usage);
    while (mach_usage) {
        mach_usage_count++;
    }
}

Figure 26. Extracting job accounting information from a history file (Part 2 of 3)
printf(" Machine Usage number = %dn", mach_usage_count);
ll_get_data(mach_usage, LL_MachUsageMachineName, &str_data);
printf(" Machine name = %s", 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(" Dispatch Usage number = %d", disp_usage_count);
    ll_get_data(disp_usage, LL_DispUsageStepUserTime64, &int64_data);
    ll_get_data(disp_usage, LL_DispUsageStepSystemTime64, &int64_data);
    printf(" LL_DispUsageStepUserTime64 = %lld (microsecs)", int64_data);
    printf(" LL_DispUsageStepSystemTime64 = %lld (microsecs)", 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_DispUsageGetFirstEventUsage, &event_usage);
while (event_usage) {
    event_usage_count++;
    printf(" Event Usage number = %d", event_usage_count);
    ll_get_data(event_usage, LL_EventUsageEventName, &str_data);
    ll_get_data(event_usage, LL_EventUsageStepUserTime64, &int64_data);
    ll_get_data(event_usage, LL_EventUsageStepSystemTime64, &int64_data);
    printf(" LL_EventUsageEventName = %s", STR_NULL(str_data));
    printf(" LL_EventUsageStepUserTime64 = %lld (microsecs)", int64_data);
    printf(" LL_EventUsageStepSystemTime64 = %lld (microsecs)", int64_data);
    ll_get_data(disp_usage, LL_DispUsageGetNextEventUsage, &event_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 26. 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.

Linux notes
The Parallel Job API is not supported in LoadLeveler for Linux.

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 SIGUSER1 indicates the job must terminate but will be restarted, while a SIGTERM indicates the job must terminate but will not be restarted.

Il_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
Parallel Job API

LoadLeveler executable specified in the job command file and the Parallel Slaves are the processes started by the Parallel Master through the **il_start_host** API.

**Note:** This API is obsolete and is supported for backward compatibility only.

**Library**

LoadLeveler API library **libllapi.a**

**Syntax**

```c
int ll_get_hostlist(struct JM_JOB_INFO *jobinfo);
```

**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.

```c
typedef 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;
} JM_JOB_INFO;

typedef 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;
} JM_NODE_INFO;
```

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 \texttt{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

\texttt{ll\_start\_host} subroutine

Purpose
This subroutine starts a task on a selected machine.

Library
LoadLeveler API library \texttt{libllapi.a}

Syntax

```c
int ll_start_host(char *host, char *start_cmd);
```

Parameters

- \textit{host}
  
  Is the name of the node on which you want to start the task.

- \textit{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 \texttt{ll\_get\_hostlist} subroutine once and only once by the Parallel Master. Acquiring the \textit{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 \texttt{ll\_start\_host} returns a negative value. The Parallel Master exit status is included in the job mail returned to the user.
Parallel Job API

**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.

**Examples**
A sample program called `para_api.c` is provided in the `samples/llpara` 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/llpara`. For example:
   ```c
   char *startCmd = "/home/user/para_api -s";
   ```
3. Issue `make` to create the executable `para_api`.
4. Update your job command file as follows:

   ```bash
   #!/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
   
   Submit the job command file to LoadLeveler.
   The syntax to invoke the Parallel Master is:
   ```bash
   para_api
   ```
   The syntax to invoke the Parallel Slave is:
   ```bash
   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 297 for more information. These APIs exist for backward compatibility. It is recommended that you use the Data Access API when possible.

Linux notes

The Query API is not supported in LoadLeveler for Linux.

The query API consists of the following subroutines: ll_get_jobs, ll_free_jobs, ll_get_nodes, and ll_free_nodes.

Il_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 subroutine returns a value of zero when successful. Otherwise, it returns an integer value defined in the llapi.h file.

**Error values**

-1  There is an error in the input parameter.
-2  The API cannot connect to the central manager.
-3  The API cannot allocate memory.
-4  A configuration error occurred.
-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.
The API cannot connect to the central manager.
The API cannot allocate memory.
A configuration error occurred.
DCE identity can not be determined
No DCE credentials
DCE credentials within 300 secs of expiration
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.

Il_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.

### Linux notes

- llsubmit returns an error value of −1 and writes the error messages to stderr when:
  - DCE_ENABLEMENT is TRUE
  - SEC_ENABLEMENT is DCE or CTSEC
  - SCHEDULER_TYPE is GANG
  - pvm_root or NQS_DIR is specified
- If the job_cmd_file argument is associated with a PVM job or an NQS job, llsubmit returns a value of −1 and writes the error messages to stderr.

### 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

```c
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.
Submit API

Description
LoadLeveler must be installed and configured correctly on the machine on which
the submit application is run.

The uid and gid in effect when llsubmit is invoked are the uid and gid used when
the job is run.

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 six subroutines:

- **ll_control subroutine**
- **ll_modify subroutine**
- **ll_preempt subroutine**
- **ll_start_job subroutine**
- **ll_terminate_job subroutine**
- **ll_start_job_ext subroutine**

The **ll_control** subroutine can be used to perform most of the LoadLeveler control operations and is designed for general use.

---

**Linux notes**
The **ll_preempt** subroutine is not available in LoadLeveler for Linux.

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 291](#) for more information.

**Note:** The AIX Workload Manager (WLM) and the LoadLeveler Workload Management API are two distinct and unrelated components.

### Il_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`.

<table>
<thead>
<tr>
<th>Linux notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>- <code>ll_control</code> returns an error condition when:</td>
</tr>
<tr>
<td>- <code>DCE_ENABLEMENT</code> is <code>TRUE</code></td>
</tr>
<tr>
<td>- <code>SEC_ENABLEMENT</code> is <code>DCE</code> or <code>CTSEC</code></td>
</tr>
<tr>
<td>- <code>SCHEDULER_TYPE</code> is <code>GANG</code></td>
</tr>
<tr>
<td>- <code>pvm_root</code> or <code>NQS_DIR</code> is specified</td>
</tr>
</tbody>
</table>

#### Library
LoadLeveler API library `libllapi.a`

#### Syntax

```c
#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);
```

298 Using and Administering LoadLeveler
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_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, LL_CONTROL_START_DRAINED,
};
```

char **host_list
A NULL terminated array of host names.

char **user_list
A NULL terminated array of user names.

char **job_list
A NULL terminated array of job 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 169.

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 MALLOCCTYPE environment variable.

LL_CONTROL_START_DRAINED:
Starts the LoadLeveler in the drained state.
Workload Management API

**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 reredispatch (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_COOKIE_DRAIN_STARTD, or LL_COOKIE_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, `llctl` operations allow the user to specify a list of hosts (through the `host_list` argument). To perform these operations, the calling program must have LoadLeveler administrator authority. The only exception to this rule is the LL_COOKIE_START operation.

**llfavorjob type of operations:** The `llfavorjob` type of control operations are: LL_COOKIE_FAVOR_JOB, and LL_COOKIE_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_COOKIE_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.

Ifavoruser type of operations: The Ifavoruser 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.

Ilhold type of operations: The Ilhold 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 Ilhold types of operations have no effect on jobs that do not belong to him/her.

Ilprio type of operations: The Ilprio 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. Ilprio 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 Ilprio 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 Ilprio 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 llcontrol 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 llcontrol version has been specified.
-20 A system error has occurred.
-21 The system cannot allocate memory.
A control_op operation that is not valid has been specified.

The job_list argument contains one or more errors.

The host_list argument contains one or more errors.

The user_list argument contains one or more errors.

Incompatible arguments have been specified for HOLD operation.

Incompatible arguments have been specified for PRIORITY operation.

Incompatible arguments have been specified for FAVORJOB operation.

Incompatible arguments have been specified for FAVORUSER operation.

An error occurred while ll_control tried to start a child process.

An error occurred while ll_control tried to start the LoadL_master daemon.

An error occurred while ll_control tried to execute the Illpurgeschedd command.

The class_list argument contains incompatible information.

ll_control cannot create a file in the /tmp directory.

LoadLeveler has encountered miscellaneous incompatible input specifications.

DCE identity can not be determined

No DCE credentials

DCE credentials within 300 secs of expiration

64-bit API not supported when DCE is enabled

This release of LoadLeveler for Linux does not support DCE.

This release of LoadLeveler for Linux does not support CTSEC.

This release of LoadLeveler for Linux does not support GANG.

This release of LoadLeveler for Linux does not support PVM.

This release of LoadLeveler for Linux does not support NQS.

Related information

ll modify subroutine

Purpose
The ll modify subroutine modifies the attributes of a submitted job step.

Linux notes

- ll modify returns an error condition when:
  - DCE_ENABLEMENT is TRUE
  - SEC_ENABLEMENT is DCE or CTSEC
  - SCHEDULER_TYPE is GANG
  - pvm_root or NQS_DIR is specified
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).

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.

<table>
<thead>
<tr>
<th>Linux notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>On LoadLeveler for Linux type can not be set to EXECUTION_FACTOR.</td>
</tr>
</tbody>
</table>

• 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 lmodify 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

MODIFY_NO_DCE_SUPPORT_ERROR
DCE_ENABLEMENT was set to TRUE or SEC_ENABLEMENT was set to DCE. LoadLeveler for Linux does not support DCE.

MODIFY_NO_CTEST_SUPPORT_ERROR
SEC_ENABLEMENT was set to CTSEC. LoadLeveler for Linux does not support CTSEC.

MODIFY_NO_GANG_SUPPORT_ERROR
SCHEDULER_TYPE was set to GANG. LoadLeveler for Linux does not support GANG scheduling.

MODIFY_NO_PVM_SUPPORT_ERROR
pvm_root was specified. LoadLeveler for Linux does not support PVM.

MODIFY_NO_NQS_SUPPORT_ERROR
NQS_DIR was specified. LoadLeveler for Linux does not support NQS.

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("**** 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\n", rc, errmsg);
    free(errmsg);
    exit(1);
}
return 0;
}

Il_preempt subroutine

Purpose
The Il_preempt subroutine enables you to preempt a running job step or to resume a job step that has already been preempted through the Ilpreempt command or the Il_preempt subroutine (user-initiated). The Il_preempt subroutine cannot resume a job step preempted through PREEMPT_CLASS rules (system-initiated).

| Linux notes |
The Il_preempt subroutine is not available in LoadLeveler for Linux.

Library
LoadLeveler API library, libllapi.a

Syntax
#include "llapi.h"

int ll_preempt (int version, LL_element **errObj, char *job_step, enum LL_preempt_op type) ;

Parameters
version
Input parameter that indicates the LoadLeveler API version (should have the same value as LL_API_VERSION in llapi.h).

errObj
Provides the address of a pointer to LL_element that points to an error object if this function fails.

The caller must free the error object storage before reusing the pointer. You can also use the ll_error subroutine to display error messages stored in the error
object. If you are going to do so, the pointer should be initialized to NULL to avoid a segmentation fault when the pointer is passed to the ll_error subroutine.

jobstep
A string used to specify the name of a job step.

type
• Preempts job step if type equals PREEMPT_STEP
• Resumes job step if type equals RESUME_STEP

Description
ll_preempt() is the API for the llpreempt command.
• This function is for 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.

Linux notes
• ll_start_job returns an error condition when:
  – DCE_ENABLEMENT is TRUE
  – SEC_ENABLEMENT is DCE or CTSEC
  – SCHEDULER_TYPE is GANG
  – pvm_root or NQS_DIR is specified

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

This subroutine does not allow you to specify adapter usage information. Use the
ll_start_job_ext subroutine instead.

You must set SCHEDULER_TYPE = API in the global configuration file to use this
subroutine.

Only jobs steps currently in the Idle state are started.

Only processes having the LoadLeveler administrator user ID can invoke this
subroutine.

An external scheduler uses this subroutine to start jobs that are in idle state. The
list of jobs that are currently in the system is retrieved with the ll_get_objs API
function, passing in a query element with type JOBS. The list of machines
available to run jobs on is obtained with the ll_get_objs and a query element with
type MACHINES. Additional data about both jobs and machines is obtained with
the ll_get_data function call.

When this function is used to start a step, adapter resources are assigned to the
step according to JCF network statements, if they are present. Adapter resources
are assigned in the same manner as the backfill scheduler assigns adapter
resources, except that the Communication Level on the network statement is
ignored and a value of AVERAGE is used. It is the responsibility of the external
scheduler to ensure the machines to which the step is dispatched have sufficient
adapter resources to run the step. Otherwise the step will not be started.

Return values

This subroutine returns a value of zero to indicate the start job request was
accepted by the negotiator. However, a return code of zero does not necessarily
imply the job started. You can use the llq command to verify the job started.
Otherwise, this subroutine returns an integer value defined in the llapi.h file.

Error values

-1 There is an error in the input parameter.
-2 The subroutine cannot connect to the central manager.
-4 An error occurred reading parameters from the administration or the
   configuration file.
-5 The negotiator cannot find the specified StepId in the negotiator job queue.
A data transmission failure occurred.

The subroutine cannot authorize the action because you are not a LoadLeveler administrator.

The job object version number is incorrect.

The Stepld is not in the Idle state.

One of the nodes specified is not available to run the job.

One of the nodes specified does not have an available initiator for the class of the job.

For one of the nodes specified, the requirements statement does not satisfy the job requirements.

The number of nodes specified was less than the minimum or more than the maximum requested by the job.

The LoadLeveler default scheduler is enabled.

The same node was specified twice in ll_start_job nodeList.

DCE identity can not be determined

No DCE credentials

DCE credentials within 300 secs of expiration

64-bit not supported when DCE enabled

This release of LoadLeveler for Linux does not support DCE

This release of LoadLeveler for Linux does not support CTSEC

This release of LoadLeveler for Linux does not support GANG scheduling

This release of LoadLeveler for Linux does not support PVM.

This release of LoadLeveler for Linux does not support NQS.

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_start_jobs_ext.

Il_start_job_ext subroutine

Purpose
This subroutine tells the LoadLeveler negotiator to start a job on the specified nodes, indicating which adapter and adapter resources to use.
Linux notes

- _ll_start_job_ext_ returns an error condition when:
  - `DCE_ENABLEMENT` is `TRUE`
  - `SEC_ENABLEMENT` is `DCE` or `CTSEC`
  - `SCHEDULER_TYPE` is `GANG`
  - `pvm_root` or `NQS_DIR` is specified

An external scheduler uses this subroutine to start jobs that are in idle state. The list of jobs that are currently in the system is retrieved with the `ll_get_objs` API function, passing in a query element with type `JOBS`. The list of machines available to run jobs on is obtained with the `ll_get_objs` and a query element with type `MACHINES`. Additional data about both jobs and machines is obtained with the `ll_get_data` function call.

When this function is used to start a step, the external scheduler specifies the adapter resources that are assigned to the step and network statements in the JCF, if they are present, are ignored. It is the responsibility of the external scheduler to manage the availability of adapter resources and LoadLeveler does not prevent or detect the over commitment of adapter resources.

**Library**

LoadLeveler API library `libllapi.a`

**Syntax**

```c
#include "llapi.h"

int ll_start_job_ext(LL_start_job_info_ext *ptr);
```

**Parameters**

- `ptr` Specifies the pointer to the `LL_start_job_info_ext` structure that was allocated and populated by the caller. The `LL_start_job_info_ext` members are:
  - `int version_num`
    - Represents the version number of the `LL_start_job_info_ext` structure.
    - Should be set to `LL_PROC_VERSION`
  - `LL_STEP_ID StepId`
    - Represents the step ID of the job step to be started.
  - `char ** nodelist`
    - A pointer to an array of node names where the job will be started. The first member of the array is the parallel master node. The array must be ended with a NULL.
  - `int adapterUsageCount`
    - This is the size of the adapterUsage list. To determine what this number should be, add all the adapter usages for all protocols needed by one task and multiply the result by the number of tasks in the job.
  - `LL_ADAPTER_USAGE * adapterUsage`
    - This is a list of adapter information. The size of this list is given by `adapterUsageCount`. The members of this structure are:
      - `char * dev_name`
        - The device name of the adapter to be used.`
Workload Management API

char * protocol
A character string representing the communication protocol this usage supports. Valid values are MPI, LAPI, and MPI_LAPI.

char * subsystem
The communication subsystem this usage supports. Valid values are IP or US.

int wid
For US subsystem usages, this indicates which adapter window ID to use. For IP subsystem usages, this field is ignored.

uint64_t mem
For US subsystem usages, this is the amount of adapter memory to dedicate to the adapter usage. For IP subsystem usages, this field is ignored.

Each element in the adapterUsage list represents one communication channel for a task. If the subsystem is US (User Space), a communication channel will require a switch adapter window. Adapter windows, and User Space usages, must be specified on actual switch adapters that are only accessible if AGGREGATE_ADAPTERS=False is specified in the configuration file.

Description
You must set SCHEDULER_TYPE = API in the global configuration file to use this subroutine. In order to have access to the physical switch adapters in the LoadLeveler cluster (as opposed to virtual adapters representing all of the adapters on a network or adapters striping across multiple networks) you must specify AGGREGATE_ADAPTERS = False in the global configuration file.

Only jobs steps currently in the Idle state are started.

Only processes having the LoadLeveler administrator user ID can invoke this subroutine.

An external scheduler uses this subroutine in conjunction with the **ll_query** and **ll_get_data** subroutines of the query API. The query API returns information about which machines are available for scheduling and which jobs are currently in the job queue waiting to be scheduled.

The node list that is passed to the external scheduler API specifies the node on which each task of the job being started is to run. The distribution of tasks to nodes in the node list must be consistent with the node allocation specified by the job command file of the job being started. If it is not, the results are undefined and the job may fail to start or may start with incorrect node assignments. Except when **BLOCKING** is specified, the entries for tasks that are running on the same node must all be specified sequentially in the node list. The table below describes how nodes should be arranged in the node list for the possible combinations of node and task specification in the job command file. In the table `/` denotes integer division and (N mod M) is the remainder of the integer division of N by M.

<table>
<thead>
<tr>
<th>Job command file specification</th>
<th>Required nodelist structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>node=N</td>
<td>There must be N different machine names specified, each specified T times.</td>
</tr>
<tr>
<td>tasks_per_node=T</td>
<td></td>
</tr>
<tr>
<td>node=N,M</td>
<td>There must be between N and M different machine names specified, each specified T times.</td>
</tr>
<tr>
<td>tasks_per_node=T</td>
<td></td>
</tr>
</tbody>
</table>
### Return values

This routine returns a non-zero value to indicate the start request was not delivered to the negotiator. These values are defined in the `llapi.h` file and explained in "Error values." A return code of zero indicates the request was successfully delivered to the negotiator, but constraints on the negotiator may stop the job from starting. You can use the `llq` command to verify the job started.

#### Error values

- **-1** There is an error in the input parameter.
- **-2** The subroutine cannot connect to the central manager.
- **-4** An error occurred reading parameters from the administration or the configuration file.
- **-5** The negotiator cannot find the specified StepId in the negotiator job queue.
- **-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.

---

<table>
<thead>
<tr>
<th>node=N</th>
<th>total_tasks=TT</th>
<th>There must be N different machine names specified, each specified TT/N times.</th>
</tr>
</thead>
<tbody>
<tr>
<td>node=N</td>
<td>total_tasks=TT</td>
<td>There must be N different machine names specified. The first (TT mod N) unique</td>
</tr>
<tr>
<td></td>
<td>TT not evenly</td>
<td>machine names must each be specified (TT/N +1) times and the remaining machine</td>
</tr>
<tr>
<td></td>
<td>divisible by N</td>
<td>names are each specified TT/N times.</td>
</tr>
<tr>
<td>total_tasks=TT</td>
<td>BLOCKING=B</td>
<td>There must be TT/B sets of machine names specified, each set specifies a machine name B times. It is permissible for a machine name to be specified in more than one set.</td>
</tr>
<tr>
<td></td>
<td>TT evenly divisible by B</td>
<td>There must be TT/B sets of machine names specified. The first TT/B sets specify a machine name B times. The last set specifies a machine name (TT mod B) times. It is permissible for a machine name to be specified in more than one set.</td>
</tr>
<tr>
<td>total_tasks=TT</td>
<td>BLOCKING=UNLIMITED</td>
<td>There must be TT entries in the node list. It is permissible for a machine name to be specified in the list more than once and it is not required that the specifications be contiguous.</td>
</tr>
<tr>
<td>task_geometry</td>
<td></td>
<td>There is a 1:1 correspondence between entries in the nodelist and task ids specified in the task geometry statement. Entries in the node list that correspond to task IDs in the same set must specify the same machine. Entries in the node list that correspond to task IDs in different sets must specify different machines.</td>
</tr>
</tbody>
</table>
One of the nodes specified does not have an available initiator for the class of the job.

For one of the nodes specified, the requirements statement does not satisfy the job requirements.

The number of nodes specified was less than the minimum or more than the maximum requested by the job.

The LoadLeveler default scheduler is enabled.

The same node was specified twice in the ll_start_job nodeList.

DCE identity cannot be determined.

DCE credentials

DCE credentials within 300 secs of expiration

64-bit API not supported when DCE is enabled

Adapter usage information does not match job structure.

Adapter usage requested an adapter not on the machine.

Wrong number of entries on adapter usage list.

The adapter usage information did not specify the same protocol usage on each task.

An invalid protocol string was specified on an adapter usage.

The adapter usages specified incompatible protocols (such as PVM and MPI)

An adapter usage specified a communication subsystem that was not IP or US

This release of LoadLeveler for Linux does not support DCE.

This release of LoadLeveler for Linux does not support CTSEC.

This release of LoadLeveler for Linux does not support GANG scheduling.

This release of LoadLeveler for Linux does not support PVM.

This release of LoadLeveler for Linux does not support NQS.

Examples
Makefiles and examples which use this subroutine are located in the samples/llsch subdirectory of the release directory. The examples include the executable sch_api_ext, which invokes the query API and the job control API to start the first job in the list received from ll_query on one node and to cancel the second job in the list. To compile sch_api_ext, copy the sample to a writeable directory and update the RELEASE_DIR field to represent the current LoadLeveler release directory.

Related information
Subroutines: ll_start_job, ll_query, ll_get_data, ll_terminate_job, ll_start_job.

Il_terminate_job subroutine

Purpose
This subroutine tells the negotiator to cancel the specified job step.
Linux notes

- `ll_terminate_job` returns an error condition when:
  - `DCE_ENABLEMENT` is `TRUE`
  - `SEC_ENABLEMENT` is `DCE` or `CTSEC`
  - `SCHEDULER_TYPE` is `GANG`
  - `pvm_root` or `NQS_DIR` is specified

Library

LoadLeveler API library `libllapi.a`

Syntax

```c
#include "llapi.h"

int ll_terminate_job(LL_terminate_job_info *ptr);
```

Parameters

- `ptr` Specifies the pointer to the `LL_terminate_job_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 Stepld`
    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.
Workload Management API

-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

-27 This release of LoadLeveler for Linux does not support DCE.
-28 This release of LoadLeveler for Linux does not support CTSEC.
-29 This release of LoadLeveler for Linux does not support GANG scheduling.
-30 This release of LoadLeveler for Linux does not support PVM.
-31 This release of LoadLeveler for Linux does not support NQS.

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_start_job 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.

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 51). 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

| Linux notes |
| This keyword is not supported in LoadLeveler for Linux. |
User exits

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 files set of the PSSP install image. These command file sets 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 levels than 3.1.0.3, DCE credentials forwarded by this program pair are not capable of being subsequently forwarded by rsh commands.

**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.
```
User exits

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. A submit filter can only make changes to LoadLeveler job command file keyword statements.

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

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.
User exits

Syntax:

\texttt{prolog\_program}

Where \texttt{prolog\_program} is the name of the prolog program as defined in the \texttt{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 53.

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:

\begin{itemize}
\item \texttt{0} The job will begin.
\item \texttt{1} The job does not begin and a message is written to the starter log.
\end{itemize}

Sample prolog programs:

\textit{Sample of a prolog program for korn shell:}

\begin{verbatim}
#!/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
  OSL
    mount_osl_files >> $LOG
    if [ status = 0 ]
      then EXIT_CODE=1
    else
      EXIT_CODE=0
    fi
  ::
  sim
    copy_sim_data >> $LOG
    if [ status = 0 ]
      then EXIT_CODE=0
    else
      EXIT_CODE=1
    fi
  ::
  *
# All other job will require free space in /tmp, make sure
# enough space is available.
\end{verbatim}
User exits

```bash
*)
  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.
User exits

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 53. 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
  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
    breakw

  # 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 [ $2 = 0 ];
      then copy_sim_results >> $LOG
    endif
    breakw

User exits

# Clean up /tmp
default:
clean_tmp >> $LOG
breaksw
eendsw
Chapter 17. 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

<table>
<thead>
<tr>
<th>Task</th>
<th>For more information see page:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building jobs</td>
<td>“Step 1: Building jobs” on page 326</td>
</tr>
<tr>
<td>Editing job command files</td>
<td>“Step 2: Edit the job command file” on page 335</td>
</tr>
<tr>
<td>Submitting job command files</td>
<td>“Step 3: Submit a job command file” on page 336</td>
</tr>
<tr>
<td>Obtaining job status and updating the display</td>
<td>“Step 4: Display, refresh, and obtain job status” on page 336</td>
</tr>
<tr>
<td>Sorting the Jobs window</td>
<td>“Step 5: Sort the Jobs window” on page 337</td>
</tr>
<tr>
<td>Change job priority</td>
<td>“Step 6: Change priorities of jobs in a queue” on page 338</td>
</tr>
<tr>
<td>Placing a job on hold</td>
<td>“Step 7: Hold a job” on page 338</td>
</tr>
<tr>
<td>Releasing a job on hold</td>
<td>“Step 8: Release a hold on a job” on page 339</td>
</tr>
<tr>
<td>Cancelling a job</td>
<td>“Step 9: Cancel a job” on page 339</td>
</tr>
<tr>
<td>Display and refresh machine status</td>
<td>“Step 12: Display and refresh machine status” on page 340</td>
</tr>
<tr>
<td>Sorting the Machines window</td>
<td>“Step 13: Sort the Machines window” on page 341</td>
</tr>
<tr>
<td>Finding the central manager location</td>
<td>“Step 14: Find the location of the central manager” on page 341</td>
</tr>
<tr>
<td>Finding the location of the public scheduling machines</td>
<td>“Step 15: Find the location of the public scheduling machines” on page 341</td>
</tr>
<tr>
<td>Specifying which jobs appear in the Jobs window</td>
<td>“Step 17: Specify which jobs appear in the Jobs window” on page 342</td>
</tr>
<tr>
<td>Specifying which machines appear in Machines window</td>
<td>“Step 18: Specify which machines appear in Machines window” on page 343</td>
</tr>
<tr>
<td>Saving LoadLeveler messages in a file</td>
<td>“Step 19: Save LoadLeveler messages in a file” on page 343</td>
</tr>
</tbody>
</table>
Using the GUI

**Step 1: Building jobs**

From the Jobs window:

SELECT File → Build a Job

<> The dialog box shown in Figure 27 on page 327 appears:
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/lpp/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`.

<table>
<thead>
<tr>
<th>Field</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable</td>
<td>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.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Parameters to pass to the program. Required only if the executable requires them.</td>
</tr>
<tr>
<td>Stdin</td>
<td>Filename to use as standard input (stdin) by the program. Optional. The default is <code>/dev/null</code>.</td>
</tr>
<tr>
<td>Stdout</td>
<td>Filename to use as standard output (stdout) by the program. Optional. The default is <code>/dev/null</code>.</td>
</tr>
<tr>
<td>Stderr</td>
<td>Filename to use as standard error (stderr) by the program. Optional. The default is <code>/dev/null</code>.</td>
</tr>
<tr>
<td>Initialdir</td>
<td>Initial directory. LoadLeveler changes to this directory before running the job. Optional. The default is your current working directory.</td>
</tr>
<tr>
<td>Notify User</td>
<td>User id of person to notify regarding status of submitted job. Optional. The default is your userid.</td>
</tr>
<tr>
<td>StartDate</td>
<td>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.</td>
</tr>
<tr>
<td>StartTime</td>
<td>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 <code>StartTime</code> but not <code>StartDate</code>, the default <code>StartDate</code> is the current day. If you specify <code>StartDate</code> but not <code>StartTime</code>, the default <code>StartTime</code> is 00:00:00. This means that the job will start as soon as possible on the specified date.</td>
</tr>
<tr>
<td>Priority</td>
<td>Number between 0 and 100, inclusive. Optional. The default is 50. This is the user priority. For more information on this priority, refer to &quot;Setting and changing the priority of a job&quot; on page 51.</td>
</tr>
<tr>
<td>Image size</td>
<td>Number in kilobytes that reflects the maximum size you expect your program to grow to as it runs. Optional.</td>
</tr>
<tr>
<td>Class</td>
<td>Class name. The job will only run on machines that support the specified class name. Your system administrator defines the class names. Optional: • Press the Choices button to get a list of available classes. • Press the Details button under the class list to obtain long listing information about classes.</td>
</tr>
<tr>
<td>Field</td>
<td>Input</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
</tr>
</tbody>
</table>
| Hold  | Hold status of the submitted job. Permitted values are:  
  *user* | User hold  
  *system* | System hold (only valid for LoadLeveler administrators)  
  *usersys* | User and system hold (only valid for LoadLeveler administrators)  
  **Note:** The default is a no-hold state. |
| Account Number | Number associated with the job. For use with the llaacctmrg and llssummary commands for acquiring job accounting data.  
Optional. Required only if the ACCT keyword is set to A VALIDATE 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, /bin/sh 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:  
  *shared* | The node can be shared with other tasks of other job steps. This is the default.  
  *not shared* | The node cannot be shared.  
  *slice not shared* | Job will not share nodes during its time-slice. |
| Dependency | A Boolean expression defining the relationship between the job steps.  
Optional. |
| Large Page | Whether or not the job step requires Large Page memory.  
  *yes* | Use Large Page memory if available, otherwise use regular memory.  
  *mandatory* | Use of Large Page memory is mandatory.  
  *no* | Do not use Large Page memory. |
| Comments | Comments associated with the job. These comments help to distinguish one job from another job.  
Optional. |

**Note:** The fields that appear in this table are what you see when viewing the Build a Job window. The text in these fields does not necessarily correspond with the keywords listed in Chapter 12, “Job command file keywords,” on page 97.

See Chapter 12, “Job command file keywords,” on page 97 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 98 for more information.
- **Interval** Yes, checkpoint the job at intervals determined by LoadLeveler. See “checkpoint” on page 98 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
fields will be available; any unavailable fields will be greyed out. LoadLeveler will assign defaults for any fields that you leave blank.

<table>
<thead>
<tr>
<th>Field</th>
<th>Available in:</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min # of Nodes</td>
<td>Tasks Per Node Model and Tasks with Uniform Blocking Model</td>
<td>Minimum number of nodes required for running the parallel job. For more information, see &quot;node&quot; on page 111. Optional. The default is one.</td>
</tr>
<tr>
<td>Max # of Nodes</td>
<td>Tasks Per Node Model</td>
<td>Maximum number of nodes required for running the parallel job. For more information, see &quot;node&quot; on page 111. Optional. The default is the minimum number of nodes.</td>
</tr>
<tr>
<td>Tasks per Node</td>
<td>Tasks Per Node Model</td>
<td>The number of tasks of the parallel job you want to run per node. For more information, see &quot;tasks_per_node&quot; on page 121. Optional.</td>
</tr>
<tr>
<td>Total Tasks</td>
<td>Tasks with Uniform Blocking Model, and Custom Blocking Model</td>
<td>The total number of tasks of the parallel job you want to run on all available nodes. For more information, see &quot;total_tasks&quot; on page 122. Optional for Uniform, required for Custom Blocking. The default is one.</td>
</tr>
<tr>
<td>Blocking</td>
<td>Custom Blocking Model</td>
<td>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 &quot;blocking&quot; on page 97.</td>
</tr>
<tr>
<td>Task Geometry</td>
<td>Custom Geometry Model</td>
<td>The task ids of each task that you want to run on each node. You can use the &quot;Set Geometry&quot; button for step-by-step directions. For more information, see &quot;task_geometry&quot; on page 121.</td>
</tr>
</tbody>
</table>

**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 109.

<table>
<thead>
<tr>
<th>Field</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI/LAPI</td>
<td>Choose one or none of these boxes to specify the MPI (Message Passing Interface) protocol, the LAPI (Low-level Application Programming Interface) protocol or neither protocol. When you choose MPI protocol, the share windows between MPI and LAPI checkbox lets you use MPI_LAPI as the protocol. Optional.</td>
</tr>
<tr>
<td>MPI_LAPI</td>
<td>Share windows between MPI and LAPI.</td>
</tr>
<tr>
<td>Adapter/Network</td>
<td>Select an adapter name or a network type from the list. Required for each protocol you select.</td>
</tr>
<tr>
<td>Adapter Usage</td>
<td>Specifies that the adapter is either shared or not shared. Optional. The default is shared.</td>
</tr>
<tr>
<td>Communication Mode</td>
<td>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.</td>
</tr>
</tbody>
</table>
**Using the GUI**

<table>
<thead>
<tr>
<th>Field</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication Level</td>
<td>Implies the amount of memory to be allocated to each window for User Space mode. Allocation can be Low, Average, or High.</td>
</tr>
<tr>
<td>Instances</td>
<td>Specifies the number of windows or IP addresses the communication subsystem should allocate to this protocol. Optional. The default is 1 unless sn_all or csss is specified for network and then the default is max.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Machine type. The job will not run on any other machine type. Optional. The default is the architecture of your current machine.</td>
</tr>
<tr>
<td>Operating System</td>
<td>Operating system. The job will not run on any other operating system. Optional. The default is the operating system of your current machine.</td>
</tr>
<tr>
<td>Disk</td>
<td>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.</td>
</tr>
<tr>
<td>Memory</td>
<td>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.</td>
</tr>
<tr>
<td>Large Page Memory</td>
<td>Amount of Large Page Memory, in megabytes. The job step requires at least this much Large Page Memory to run. Optional.</td>
</tr>
<tr>
<td>Machines</td>
<td>Machine names. The job will only run on the specified machines. Optional.</td>
</tr>
<tr>
<td>Total Memory</td>
<td>Amount of total (regular and Large Page Memory) in megabytes needed to run the job step. Optional.</td>
</tr>
<tr>
<td>Features</td>
<td>Features. The job will only run on machines with specified features. Optional.</td>
</tr>
<tr>
<td>LoadLeveler Version</td>
<td>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.</td>
</tr>
<tr>
<td>Pool</td>
<td>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.</td>
</tr>
<tr>
<td>Requirement</td>
<td>Requirements. The job will only run if these requirements are met.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Input</th>
</tr>
</thead>
</table>
| 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 |
### Using the GUI

#### Field | Input
--- | ---
Data Limit | Maximum amount of the data segment that the submitted job can use. Express the amount as: `integer[, fraction][units]`<br>Optional
Core Limit | Maximum size of a core file.<br>Optional
RSS Limit | Maximum size of the resident set size. It is the largest amount of physical memory a user’s process can allocate.<br>Optional
File Limit | Maximum size of a file that is created.<br>Optional
Stack Limit | Maximum size of the stack.<br>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.<br>Optional
Wall Clock Limit | Maximum amount of elapsed time for which a job can run.<br>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.<br>Optional. The default is one.
Max # of Processors | Maximum number of processors required for running the PVM job.<br>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.<br>Required.
Adapter Usage | Specifies that the adapter is either shared or not shared.<br>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 12, “Job command file keywords,” on page 97.

<table>
<thead>
<tr>
<th>Field</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ckpt Directory</td>
<td>Specifies a checkpoint directory name.</td>
</tr>
<tr>
<td>Ckpt Time Limits</td>
<td>Sets the limits for the elapsed time a job can take checkpointing.</td>
</tr>
</tbody>
</table>

**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.  
     **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**  
     Add a step to the job command file  
     Delete a step from the job command file  
     Clear the fields in the Build a Job window  
     Select defaults to use in the fields

     **Select**  
     Add a Step  
     Delete a Step  
     Clear Fields  
     Set Field Defaults

**Note:** Other options include Go to Next Step, Go to Previous Step, and Go to Last Step that allow you to edit various steps in the job command file.

Using the Tools pull-down menu, you can modify the job command file. Your choices appear in the following table:

**To**  
Name the job  
Open a window where you can enter a script file  
Fill in the fields using another file  
View the job command file in a window

**Select**  
Set Job Name  
Append Script  
Restore from File  
View Entire Job
To Select
Determine which step you are viewing What is step #
Start a new job command file Start a new job

<table>
<thead>
<tr>
<th>To</th>
<th>Do This</th>
</tr>
</thead>
</table>
| Save the information you entered into a file which you can submit later | SELECT Save
> A window appears prompting you to enter a job filename.
ENTER a job filename in the text entry field.
SELECT OK
> The window closes and the information you entered is saved in the file you specified. |
| Submit the program immediately and discard the information you entered | SELECT Submit
GO TO Step 4 |

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:

**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 198.

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

<table>
<thead>
<tr>
<th>To:</th>
<th>Select Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort jobs by the machine from which they were submitted</td>
<td>Sort by Submitting Machine</td>
</tr>
<tr>
<td>Sort by owner</td>
<td>Sort by Owner</td>
</tr>
</tbody>
</table>
### 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 51. 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
<> LoadLeveler cancels the job and the job information disappears from the Jobs window.

Go to the next step.

Step 10: Modify consumable CPUs, consumable memory, class, and account number
Modifies the consumable CPUs or memory requirements of a non-running job.

SELECT
Modify → Consumable CPUs
or
Modify → Consumable Memory
or
Modify → Class
or
Modify → Account number
<> A dialog box appears prompting you to enter a number representing the new value for consumable CPUs or consumable memory, or a new class or account number.

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
Using the GUI

- 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 \texttt{llstatus -l} command.

SELECT Actions → Adapter Details

This displays virtual and physical adapter information for each selected machine. Selecting this option has the same effect as typing the \texttt{llstatus -a} command.

SELECT Actions → Floating Resources

This displays consumable resources for the LoadLeveler cluster. Selecting this option has the same effect as typing the \texttt{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 \texttt{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

<table>
<thead>
<tr>
<th>To:</th>
<th>Select Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort by machine name</td>
<td>Sort by Name</td>
</tr>
<tr>
<td>Sort by sched state</td>
<td>Sort by Sched</td>
</tr>
<tr>
<td>Sort by total number of jobs scheduled</td>
<td>Sort by InQ</td>
</tr>
<tr>
<td>Sort by number of running jobs scheduled by this machine</td>
<td>Sort by Act</td>
</tr>
<tr>
<td>Sort by startd state</td>
<td>Sort by Startd</td>
</tr>
<tr>
<td>Sort by the number of jobs running on this machine</td>
<td>Sort by Run</td>
</tr>
<tr>
<td>Sort by load average</td>
<td>Sort by LdAvg</td>
</tr>
<tr>
<td>Sort by keyboard idle time</td>
<td>Sort by Idle</td>
</tr>
<tr>
<td>Sort by hardware architecture</td>
<td>Sort by Arch</td>
</tr>
<tr>
<td>Sort by operating system type</td>
<td>Sort by OpSys</td>
</tr>
<tr>
<td>Not specify a sort</td>
<td>No Sort</td>
</tr>
</tbody>
</table>

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.

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.

<table>
<thead>
<tr>
<th>To Display</th>
<th>Select Select +</th>
</tr>
</thead>
<tbody>
<tr>
<td>All jobs in the queue</td>
<td>All</td>
</tr>
<tr>
<td>All jobs belonging to a specific user (or users)</td>
<td>By User</td>
</tr>
<tr>
<td></td>
<td>&lt;&gt; A window appears prompting you to enter the user IDs whose jobs you want to view.</td>
</tr>
<tr>
<td>All jobs submitted to a specific machine (or machines)</td>
<td>By Machine</td>
</tr>
<tr>
<td></td>
<td>&lt;&gt; A window appears prompting you to enter the machine names on which the jobs you want to view are running.</td>
</tr>
<tr>
<td>All jobs belonging to a specific group (or groups)</td>
<td>By Group</td>
</tr>
<tr>
<td></td>
<td>&lt;&gt; A window appears prompting you to enter the LoadLeveler group names to which the jobs you want to view belong.</td>
</tr>
<tr>
<td>All jobs having a particular ID</td>
<td>By Job Id</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
To Display | Select Select *
---|---
Note: When you choose By User, By Machines, or By Group, you can use a UNIX regular expression enclosed in parenthesis. For example, you can enter (^k10) to display all machines beginning with the characters “k10”.

**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 | All
View machines by operating system | by OpSys
< A window appears prompting you to enter the operating system of those machines you want to view.
View machines by hardware architecture | by Arch
< A window appears prompting you to enter the hardware architecture of those machines you want to view.
View machines by state | by State
< A cascading pull-down menu appears prompting you to select the state of the machines that you want to view.

**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
Using the GUI

<> 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 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, see “llextSDR - Extract adapter information from the SDR” on page 180 or “llextRPD - Extract data from an RSCT peer domain” on page 177.
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_jobs_scheduled = 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 28. 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. To take advantage of dynamic adapter configuration you must exclude this keyword from the machine stanza. LoadLeveler will then dynamically obtain the adapter configuration for this machine from the RSCT.

**Note:** The dynamic adapter configuration feature cannot be used to determine adapter characteristics for the following switch adapters:
- SP_Switch_Adapter
- SP_Switch_MX_Adapter
- SP_Switch_MX2_Adapter
- RS/6000_SP_System_Attachment_Adapter
- SP_Switch2_Adapter
- SP_Switch2_PCIAttachment_Adapter
- SP_Switch2_MX2_Adapter
- SP_Switch2_PCI-X_Attachment_Adapter

All adapter stanzas you define must be specified on this keyword. If the keyword is specified without defining any adapter stanza names no adapter will be configured for the machine.

`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.

**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.
Customizing the administration file

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
   ```

   - 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 466.

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 385.

`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 417.

dce_host_name = dce hostname
Where dce hostname is the DCE hostname of this machine. Execute either "SDRGetObjects Node dcehostname," or "llextSDR" 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.
genral 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_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.

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.

resources = name(count) name(count) ... name(count)
Customizing the administration file

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. 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 lctcl -h host purgesched command when you want to attempt to recover resources lost when a node running the schedd daemon fails. A true setting prevents conflicts from arising when a schedd machine is restarted while a purge is taking place. For more information, see “How Do I Recover Resources Allocated by a schedd Machine?” on page 468.

schedd_host = true | false

When true this keyword specifies that if a schedd is running on a machine that it will serve as a public scheduling machine. A public scheduling machine accepts job submissions from other machines in the LoadLeveler cluster. Jobs are submitted to a public scheduling machine if:

1. The submission occurs on a machine which does not run the schedd daemon. These include submit-only machines and machines which are configured to run other LoadLeveler daemons but not the schedd daemon.
2. The submission occurs on a machine which runs the schedd daemon but is configured to submit jobs to a public scheduling machine by having the SCHEDD_SUBMIT_AFFINITY keyword set to false in the global or local configuration file.
Customizing the administration file

This keyword does not configure LoadLeveler to run the schedd daemon on a node. Use the configuration keyword **SCHEDD_RUNS_HERE** to run the schedd daemon on a node. Refer to 376 for more information.

**spacct_excluse_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 380. 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.

In order to distinguish speed among different machines you must include this value in the local configuration file.

For information on how the **speed** keyword can be used to schedule machines, refer to Step 7: Prioritize the order of executing machines maintained by the negotiator on page 380.

**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**.

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/loadl/1.2.0/aix32/pvm3
```
Customizing the administration file

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:

```
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 29. Format of a user stanza

You can specify the following keywords in 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 current UNIX group.
**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 352 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 471.

**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, 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 471.

**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 471.

**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
Customizing the administration file

-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 378 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
```
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.

```plaintext
# 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.
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 393.

- **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)
  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. The allowable units are those normally used with LoadLeveler data limits:

- b bytes
- w words
- kb kilobytes \((2^{10} \text{ bytes})\)
- kw kilowords \((2^{12} \text{ bytes})\)
- mb megabytes \((2^{20} \text{ bytes})\)
- mw megawords \((2^{22} \text{ bytes})\)
- gb gigabytes \((2^{30} \text{ bytes})\)
- gw gigawords \((2^{32} \text{ bytes})\)
- tb terabytes \((2^{40} \text{ bytes})\)
- tw terawords \((2^{42} \text{ bytes})\)
- pb petabytes \((2^{50} \text{ bytes})\)
- pw petawords \((2^{52} \text{ bytes})\)
- eb exabytes \((2^{60} \text{ bytes})\)
- ew exawords \((2^{62} \text{ bytes})\)

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 whether a job step of this class is preemptable or not. The default value is 1, indicating it is preemptable. The values 2 and 3 are allowed for compatibility reasons and mean the same as 1.

**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
Customizing the administration file

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_protocol_instances = number**
Where number specifies the maximum value allowed on the instances keyword on the network statement for jobs submitted on this class. The default is 2.

**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 87.

```
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 378 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**

<table>
<thead>
<tr>
<th>Limit</th>
<th>How It Is Enforced</th>
</tr>
</thead>
<tbody>
<tr>
<td>ckpt_time_limit</td>
<td>Per job step</td>
</tr>
<tr>
<td>core_limit</td>
<td>Per process</td>
</tr>
<tr>
<td>cpu_limit</td>
<td>Per process</td>
</tr>
<tr>
<td>data_limit</td>
<td>Per process</td>
</tr>
<tr>
<td>file_limit</td>
<td>Per process</td>
</tr>
<tr>
<td>job_cpu_limit</td>
<td>Per job step</td>
</tr>
<tr>
<td>rss_limit</td>
<td>Per process</td>
</tr>
<tr>
<td>stack_limit</td>
<td>Per process</td>
</tr>
<tr>
<td>wall_clock_limit</td>
<td>Per job step</td>
</tr>
</tbody>
</table>

Individual keywords are described in “Specifying limits in the class stanza” on page 360. The following section gives you a general overview of limits.
Customizing the administration file

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 "Enforcing 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 106.

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.

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

Exceeding job step limits: When a hard limit is exceeded LoadLeveler sends a non-trappable signal 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. Enforcing job step limits

<table>
<thead>
<tr>
<th>Type of Job</th>
<th>When a Soft Limit is Exceeded</th>
<th>When a Hard Limit is Exceeded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial</td>
<td>SIGXCPU or SIGKILL issued</td>
<td>SIGKILL issued</td>
</tr>
<tr>
<td>Parallel (non-PVM)</td>
<td>SIGXCPU issued to both the user program and to the parallel daemon</td>
<td>SIGTERM issued</td>
</tr>
<tr>
<td>PVM</td>
<td>SIGXCPU issued to the user program</td>
<td><strong>pvm_halt</strong> invoked to shut down PVM</td>
</tr>
</tbody>
</table>

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

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

\[\text{limit\_type} = \text{hardlimit,softlimit}\]

For example:

\[\text{core\_limit} = 120\text{kb}, 100\text{kb}\]

To specify only a hard limit, you can enter, for example:

\[\text{core\_limit} = 120\text{kb}\]

To specify only a soft limit, you can enter, for example:

\[\text{core\_limit} = , 100\text{kb}\]

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 \text{data\_limit}, \text{core\_limit}, \text{file\_limit}, \text{stack\_limit}, and \text{rss\_limit} — the hard limit and the soft limit are expressed as:

\[\text{integer[,fraction]}[\text{units}]\]

The allowable units for these limits are:

- b bytes
- w words
- kb kilobytes (\(2^{10}\) bytes)
- kw kilowords (\(2^{12}\) bytes)
- mb megabytes (\(2^{20}\) bytes)
- mw megawords (\(2^{22}\) bytes)
- gb gigabytes (\(2^{30}\) bytes)
- gw gigawords (\(2^{32}\) bytes)
- tb terabytes (\(2^{40}\) bytes)
- tw terawords (\(2^{42}\) bytes)
- pb petabytes (\(2^{50}\) bytes)
- pw petawords (\(2^{52}\) bytes)
- eb exabytes (\(2^{60}\) bytes)
- ew exawords (\(2^{62}\) bytes)

If no units are specified for data limits, then bytes are assumed.

For limit keywords that refer to a time limit — such as \text{ckpt\_time\_limit}, \text{cpu\_limit}, \text{job\_cpu\_limit}, and \text{wall\_clock\_limit} — the hard limit and the soft limit are expressed as:

\[\text{[hours:]minutes:seconds[.fraction]}\]

Fractions are rounded to seconds.

You can use the following character strings with all limit keywords except the \text{copy} keyword for \text{wall\_clock\_limit}, \text{job\_cpu\_limit} and \text{ckpt\_time\_limit}:
Customizing the administration file

\textbf{rlim\_infinity}

Represents the largest positive number.

\textbf{unlimited}

Has same effect as \texttt{rlim\_infinity}.

\textbf{copy}

Uses the limit currently active when the job is submitted.

See \texttt{Table 19} for more information on specifying limits.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
If the hard limit: & Then the: \\
\hline
Is set in both the class stanza and the job & Smaller of the two limits is taken into consideration. If the smaller 
command file & 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. \\
\hline
Is not set in either the class stanza or the job & User per process limit set on the machine that runs the job is used. 
command file & \\
\hline
Is set in the job command file and is less than & The job is not submitted. 
its respective job soft limit & \\
\hline
Is set in the class stanza and is less than its & Soft limit is adjusted downward to equal the hard limit. 
respective class stanza soft limit & \\
\hline
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. 
& \textbf{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. \\
\hline
\end{tabular}
\caption{Setting limits}
\end{table}

\textbf{Specifying limits in the class stanza:} You can specify the following limit keywords:

\texttt{ckpt\_time\_limit} = \texttt{hardlimit,softlimit}

Where \texttt{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 Starld daemon receives status from the starter that a checkpoint has started.

Examples:

\begin{itemize}
  \item \texttt{ckpt\_time\_limit} = 30:45 \hspace{1cm} \texttt{#}hardlimit = 30 minutes 45 seconds
  \item \texttt{ckpt\_time\_limit} = 30:45,25:00 \hspace{1cm} \texttt{#}hardlimit = 30 minutes 44 seconds
  \hspace{1cm} \texttt{#}softlimit = 25 minutes
\end{itemize}

\texttt{core\_limit} = \texttt{hardlimit,softlimit}

Specifies the hard limit, soft limit, or both for the size of a core file.

Examples:

\begin{itemize}
  \item \texttt{core\_limit} = \texttt{unlimited}
  \item \texttt{core\_limit} = \texttt{30mb}
\end{itemize}

For more information, see \textit{"Overview of limits"} on page \pageref{overview_of_limits}.

\texttt{cpu\_limit} = \texttt{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 \texttt{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:

- \texttt{cpu\_limit = 12:56:21} \quad # \text{hardlimit = 12 hours 56 minutes 21 seconds}
- \texttt{cpu\_limit = 56:00,50:00} \quad # \text{hardlimit = 56 minutes 0 seconds}
- \texttt{cpu\_limit = 1:03} \quad # \text{hardlimit = 1 minute 3 seconds}
- \texttt{cpu\_limit = unlimited} \quad # \text{hardlimit = 2,147,483,647 seconds} \quad # \text{(X'7FFFFFFF')}
- \texttt{cpu\_limit = rlim\_infinity} \quad # \text{hardlimit = 2,147,483,647 seconds} \quad # \text{(X'7FFFFFFF')}
- \texttt{cpu\_limit = copy} \quad # \text{current CPU hardlimit value on the submitting machine.}

For more information, see “Overview of limits” on page 358.

\texttt{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:

- \texttt{data\_limit = 125621} \quad # \text{hardlimit = 125621 bytes}
- \texttt{data\_limit = 5621kb} \quad # \text{hardlimit = 5621 kilobytes}
- \texttt{data\_limit = 2mb} \quad # \text{hardlimit = 2 megabytes}
- \texttt{data\_limit = unlimited} \quad # \text{hardlimit = 9,223,372,036,854,775,807 bytes} \quad # \text{(X'7FFFFFFFFFFFFFFF')}
- \texttt{data\_limit = rlim\_infinity} \quad # \text{hardlimit = 9,223,372,036,854,775,807 bytes} \quad # \text{(X'7FFFFFFFFFFFFFFF')}
- \texttt{data\_limit = copy} \quad # \text{copy data hardlimit value from submitting machine.}

For more information, see “Overview of limits” on page 358.

\texttt{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 358.

\texttt{job\_cpu\_limit = hardlimit,softlimit}

Specifies the maximum total CPU time to be used by all processes of a job step.

For example:

- \texttt{job\_cpu\_limit = 10000}

For more information on this keyword, see:
- “job\_cpu\_limit” on page 106
- \texttt{JOB\_LIMIT\_POLICY} on page 83
- For general information on limits, see “Overview of limits” on page 358.

\texttt{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 358.

\texttt{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 358.

\texttt{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
Customizing the administration file

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 370.

For more general information on limits, see “Overview of limits” on page 358.

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

362 Using and Administering LoadLeveler
Customizing the administration 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

Example 8: Creating a class for MPICH-GM jobs:
MPICHGM: type=class  # class for MPICH-GM jobs
   default_resources = gmports(1)  # one gmports resource is consumed by each
   # task, if resources are not explicitly
   # specified in the job command file

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:

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 31. Format of a group stanza
Customizing the administration file

You can specify the following keywords in a group stanza:

\textbf{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.

\textbf{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.

\textbf{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.

\textbf{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 \textbf{maxidle} or if there is no group stanza at all, the maximum number of jobs that can be simultaneously idle 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 471.

\textbf{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 \textbf{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 471.

\textbf{maxqueued = number}

Where number is the maximum number of jobs allowed in the queue for this group. This prevents groups from flooding the job queue. Jobs above this maximum are placed in the NotQueued state. If no \textbf{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 471.

\textbf{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 378 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:

# 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. If you want LoadLeveler jobs to be able to request specific adapters, you must either specify adapter stanzas or configure dynamic adapters in the administration file. For more information on configuring dynamic adapters refer to 68.

Note the following when using an adapter stanza:
Customizing the administration file

- 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 “llextSDR - Extract adapter information from the SDR” on page 180 for PSSP domains or “llextRPD - Extract data from an RSCT peer domain” on page 177 for peer domains.

An adapter stanza has the following format:

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

Figure 32. Format of an adapter stanza

You can specify the following keywords in 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.

**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.

**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.

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/stswitch_node_number file of the Parallel System Support Programs (PSSP).

logical_id = id
Where id specifies the logical id assigned to the adapter. This keyword is for peer domain switch adapters.

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, RS6000_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 llextSDR to obtain all of the adapter information from the SDR.

Note: css_type cannot be RS6000_SP_System_Attachment_Adapter or SP_Switch_Adapter if you are using Gang scheduling. For more information see “Restrictions for Gang scheduling preemption” on page 428.

adapter_type = type
Where type is the designation for the type of switch adapter to be used. This keyword is used for the High Performance Switch in a peer domain. The llextRPD command will not generate an adapter_type statement if no AdapterType is found in the cluster.

device_driver_name = name
Where name specifies the device driver interface needed for user space function. A device_driver_name will be present for all adapter stanzas whose name begins with sn This keyword is for peer domain switch adapters.

network_id = id
Where id specifies the network id to which the adapter is connected. A network_id is required on all adapter stanzas whose name begins with sn. This keyword is for peer domain switch adapters.

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.

```plaintext
yugo: type=machine
    adapter_stanzas = sp01sw.ibm.com
    ...
```

```plaintext
sp01sw.ibm.com: type = adapter
    adapter_name = css0
    interface_address = 12.148.44.218
    interface_name = sp01sw.ibm.com
```
Customizing the administration file

Example 2: Specifying an HPS adapter: In the following example, the adapter stanza called “c121s0n10.ppd.pok.ibm.com” specifies an HPS adapter. Note that c121s0n10.ppd.pok.ibm.com is also specified on the adapter_stanzas keyword of the machine stanza for the “yugo” machine.

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

c121s0n10.ppd.pok.ibm.com: type = adapter
    adapter_name = sn0
    network_type = switch
    interface_address = 192.168.0.10
    interface_name = c121s0n10.ppd.pok.ibm.com
    multilink_address = 10.10.10.10
    logical_id = 2
    adapter_type = Switch_Network_Interface_For_HPS
    device_driver_name = sn0
    network_id = 1

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

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:

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 31.

LoadLeveler administrators on this list also receive mail describing problems that are encountered by the master daemon. When DCE or CtSec are enabled, the LOADL_ADMIN list is used only as a mailing list. For more information, see “Step 16: Configuring LoadLeveler to use a security service” on page 401.

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:
Customizing 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 380.

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 (POLLS_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 preemption” on page 428.

PREEMPTION_SUPPORT = full | no_adapter | none
For Gang or API schedulers, specifies the level of preemption support for a cluster. When set to full, preemption is fully supported. When set to no_adapter, preemption is supported but the adapter resources are not released by preemption. When set to none, preemption is not supported. The default value for the GANG scheduler is full. The default value for API schedulers is none. If the value of this keyword is set to any value other than none for the Default and Backfill schedulers, LoadLeveler will not start. When set to full or no_adapter, LoadLeveler checks that MACHINE_AUTHENTICATE = TRUE and PROCESS_TRACKING = TRUE were specified in the configuration file and that only supported adapters are defined in the administration file. If this keyword is set to none preemption requests will be rejected.

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” on page 370.
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 lctl.

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 383.

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 297.

GANG
Specifies that you will use the LoadLeveler Gang scheduling algorithm. For more information, see Chapter 18, “Preemption using the Gang scheduler,” on page 421.

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 297.

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:
Customizing the configuration file

- **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 55 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 383. 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.
Customizing the configuration file

- 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 380.

See “Keyword considerations for parallel jobs” on page 55 for information on which keywords associated with parallel jobs are supported by the Backfill scheduler.

- **GANG** Similar to the Backfill scheduler. This scheduler provides preemption support.

  For more information on setting up Gang scheduling, see Chapter 18, “Preemption using the Gang scheduler,” on page 421.

- **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 291). Also, some LoadLeveler functions are not available when you use API (for more information, see “Workload Management API” on page 297 and “Usage notes” on page 314).

  When you enable a scheduler type of API, you must specify `AGGREGATE_ADAPTERS=NO` to make the individual switch adapters available to the external scheduler. This means the external scheduler receives each individual adapter connected to the network, instead of collectively grouping them together. You’ll see each adapter listed individually in the `Ilstatus -1` command output. When this keyword is set to YES, the `Ilstatus -1` command will show an aggregate adapter which contains information on all switch adapters on the same network. For detailed information about individual switch adapters, issue the `Ilstatus -a` command.

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 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
Customizing the configuration file

**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 or is not specified, it will be defaulted to No_Class (1)
  - The number of instances for a class specified inside the parenthesis () must be an unsigned integer. If the number specified is 0, it is correct syntactically, but the class will not be defined in LoadLeveler
  - If the number of instances for all classes in the CLASS statement are 0, the default No_Class(1) will be used

- The format, CLASS = [ "class1" "class2" "class3" "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.

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.

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 473 and “Step 3: Specify class stanzas” on page 353 for more information on classes.

Defining classes – examples

Example 1: This example specifies multiple classes:
Class = No_Class(2)

or
Class = { "No_Class" "No_Class" }

The machine will only run jobs that have either defaulted to or explicitly requested class No_Class. A maximum of two LoadLeveler jobs are permitted to run simultaneously on the machine if the MAX_STARTERS keyword is not specified. See “Step 5: Specify how many jobs a machine can run” on page 378 for more information on MAX_STARTERS.

Example 2: This example specifies multiple classes:
Class = No_Class(1) Small(1) Medium(1) Large(1)

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

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

Example 4: This example specifies multiple classes:
Class = B(2) D(1)

or
Class = { "B" "B" "D" }

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")
Customizing the configuration file

**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 page 349.

**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.

This keyword does not designate a machine as a public scheduling machine. Unless configured as a public scheduling machine, a machine configured to run the schedd daemon will only accept job submissions from the same machine running the schedd daemon. A public scheduling machine accepts job submissions from other machines in the LoadLeveler cluster. To configure a machine as a public scheduling machine, refer to page 348 for more information.

**SCHEDD_SUBMIT_AFFINITY = true | false**
Specifies whether job submissions are directed to a locally running schedd daemon. When the keyword is set to true, job submissions are directed to a schedd daemon running on the same machine where the submission takes place, provided there is a schedd daemon running on that machine. In this case the submission is said to have "affinity" for the local schedd daemon. If there is no schedd daemon running on the machine where the submission takes place, or if this keyword is set to false, the job submission will only be directed to a schedd daemon serving as a public scheduling machine. In this case, if there are no public scheduling machines configured the job cannot be submitted. A public scheduling machine accepts job submissions from other machines in the LoadLeveler cluster. To configure a machine as a public scheduling machine refer to page 343 for further details.

Installations with a large number of nodes should consider setting this keyword to false in order to more evenly distribute dispatching of jobs among the schedd daemons. For more information, see "Scaling considerations" on page 470.

**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:

**SCHEDULE_BY_RESOURCES = name name ... name**
Specifies which consumable resources are considered by the LoadLeveler
Customizing the configuration file

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_POLICY = hard | soft | shares
Specifies what type of resource entitlements will be assigned to the AIX Workload Manager classes. If the value specified is shares, it means a share value is assigned to the class based on the job step’s requested resources (one unit of resource equals one share). This is the default policy. If the value specified is soft, it means a percentage value is assigned to the class based on the job step’s requested resources and the total machine resources. This percentage can be exceeded if there is no contention for the resource. If the value specified is hard, it means a percentage value is assigned to the class based on the job step’s requested resources and the total machine resources. This percentage cannot be exceeded regardless of the contention for the resource. If desired, this keyword can be used in the LoadL_config.local file to set up a different policy for each machine. The ENFORCE_RESOURCE_USAGE keyword must be set for this keyword to be valid.

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.
Customizing the configuration file

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 374.

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**).

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.

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 51 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 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 Ilq 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 Ilq 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.
Customizing the configuration file

- You can use the UserRunningJobs, GroupRunningJobs, 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:
SYSPRIO : (ClassSysprio * 100) + (UserSysprio * 10) + (GroupSysprio * 1) – (QDate)

Example 3: This example orders the queue based on the number of jobs a user is currently running. The user who has the fewest jobs running is first in the queue. You should set NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL in conjunction with this SYSPRIO expression.
SYSPRIO : 0 – UserRunningJobs

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 411.

You can use the following LoadLeveler variables in the MACHPRIO expression:

LoadAvg
The Berkeley one-minute load average of the machine, reported by startd.

Connectivity
The ratio of switch adapters with connectivity to the total number of switch adapters on the node expressed as a floating point value. The Connectivity value ranges from 0.0 to 1.0 where 0.0 means a node has no switch adapters and 1.0 means all switch adapters have connectivity.

Cpus The number of processors of the machine, reported by startd.
Customizing the configuration file

**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 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)
Example 6: This example gives nodes the with highest percentage of switch adapters with connectivity the highest priority:

MACHPRIO : Connectivity

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" (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)

Determine 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 startId machine’s load average every time the negotiator assigns a job. For more information, see “the NEGOTIATOR_INTERVAL keyword” on page 411.

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.
**Customizing the configuration file**

**KILL: expression that evaluates to T or F (true or false)**
Determine 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 33 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 33. How control expressions affect jobs](image)

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 10, “Gathering job accounting data,” on page 83.

The following keywords allow you to control accounting functions:
Customizing the configuration file

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 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 llacctmrg 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:

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

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

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

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

- **CENTRAL_MANAGER_HEARTBEAT_INTERVAL = 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 466.

**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.

<table>
<thead>
<tr>
<th>To specify the location of the:</th>
<th>Specify these keywords:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administration File</td>
<td>ADMIN_FILE = pathname (required)</td>
</tr>
<tr>
<td></td>
<td>Points to the administration file containing user, class, group, machine, and adapter stanzas. For example,</td>
</tr>
<tr>
<td></td>
<td>ADMIN_FILE = ${tilde}/admin_file</td>
</tr>
</tbody>
</table>
Customizing the configuration file

<table>
<thead>
<tr>
<th>To specify the location of the:</th>
<th>Specify these keywords:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Configuration File</td>
<td>LOCAL_CONFIG = pathname</td>
</tr>
<tr>
<td></td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>LOCAL_CONFIG = $(tilde)/$host.LoadL_config.local</td>
</tr>
<tr>
<td></td>
<td>LOCAL_CONFIG = $(tilde)/LoadL_config.$host.$(domain)</td>
</tr>
<tr>
<td></td>
<td>LOCAL_CONFIG = $(tilde)/LoadL_config.local.$(hostname)</td>
</tr>
<tr>
<td></td>
<td>If you are using a local file system, an example is:</td>
</tr>
<tr>
<td></td>
<td>LOCAL_CONFIG = /var/LoadL/LoadL_config.local</td>
</tr>
<tr>
<td></td>
<td>See &quot;LoadLeveler variables&quot; on page 74 for information about the tilde, host, and domain variables.</td>
</tr>
<tr>
<td>Local Directory</td>
<td>The following subdirectories reside in the local directory. It is possible that the local directory and LoadLeveler's home directory are the same.</td>
</tr>
<tr>
<td></td>
<td>COMM = local directory</td>
</tr>
<tr>
<td></td>
<td>Specifies the local directory where LoadLeveler keeps special files for communicating among LoadLeveler daemons running on the same machine. If not specified, the default location for the files is /tmp. If you change the COMM option you must stop and then restart LoadLeveler using the lctl command.</td>
</tr>
<tr>
<td></td>
<td>EXECUTE = local directory/execute (required)</td>
</tr>
<tr>
<td></td>
<td>Defines the local directory to store the executables of jobs submitted by other machines.</td>
</tr>
<tr>
<td></td>
<td>LOG = local directory/log (required)</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>SPOOL = local directory/spool (required)</td>
</tr>
<tr>
<td></td>
<td>Defines the local directory where LoadLeveler keeps the local job queue and checkpoint files, as well as:</td>
</tr>
<tr>
<td></td>
<td>HISTORY = $(SPOOL)/history (required)</td>
</tr>
<tr>
<td></td>
<td>Defines the pathname where a file containing the history of local LoadLeveler jobs is kept.</td>
</tr>
<tr>
<td>Release Directory</td>
<td>RELEASEDIR = release directory (required)</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>BIN = $(RELEASEDIR)/bin (required)</td>
</tr>
<tr>
<td></td>
<td>Defines the directory where LoadLeveler binaries are kept.</td>
</tr>
<tr>
<td></td>
<td>LIB = $(RELEASEDIR)/lib (required)</td>
</tr>
<tr>
<td></td>
<td>Defines the directory where LoadLeveler libraries are kept.</td>
</tr>
<tr>
<td></td>
<td>NQS_DIR = NQS directory (optional)</td>
</tr>
<tr>
<td></td>
<td>Defines the directory where NQS commands qsub, qstat, and qdel reside. The default is /usr/bin.</td>
</tr>
</tbody>
</table>

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...
Customizing the configuration file

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 389 describes the events that can be reported through logging controls.

“Setting up file system monitoring” on page 372 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

<table>
<thead>
<tr>
<th>Daemon/ Process</th>
<th>Log File (required)</th>
<th>Max Length (required)</th>
<th>Debug Control (required)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master</td>
<td>MASTER_LOG = path</td>
<td>MAX_MASTER_LOG = bytes</td>
<td>MASTER_DEBUG = flags</td>
</tr>
<tr>
<td>Schedd</td>
<td>SCHEDD_LOG = path</td>
<td>MAX_SCHEDD_LOG = bytes</td>
<td>SCHEDD_DEBUG = flags</td>
</tr>
<tr>
<td>Startd</td>
<td>STARTD_LOG = path</td>
<td>MAX_STARTD_LOG = bytes</td>
<td>STARTD_DEBUG = flags</td>
</tr>
<tr>
<td>Starter</td>
<td>STARTER_LOG = path</td>
<td>MAX_STARTER_LOG = bytes</td>
<td>STARTER_DEBUG = flags</td>
</tr>
<tr>
<td>Negotiator</td>
<td>NEGOTIATOR_LOG = path</td>
<td>MAX_NEGOTIATOR_LOG = bytes</td>
<td>NEGOTIATOR_DEBUG = flags</td>
</tr>
<tr>
<td>Kbdd</td>
<td>KBDD_LOG = path</td>
<td>MAX_KBDD_LOG = bytes</td>
<td>KBDD_DEBUG = flags</td>
</tr>
<tr>
<td>GSmonitor</td>
<td>GSMONITOR_LOG = path</td>
<td>MAX_GSMONITOR_LOG = bytes</td>
<td>GSMONITOR_DEBUG = flags</td>
</tr>
</tbody>
</table>

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 390 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:

\[
\text{TRUNC_MASTER_LOG_ON_OPEN} = \text{true} | \text{false} \\
\text{TRUNC_STARTD_LOG_ON_OPEN} = \text{true} | \text{false} \\
\text{TRUNC_SCHEDD_LOG_ON_OPEN} = \text{true} | \text{false} \\
\text{TRUNC_KBDD_LOG_ON_OPEN} = \text{true} | \text{false} \\
\text{TRUNC_NEGOTIATOR_LOG_ON_OPEN} = \text{true} | \text{false} \\
\text{TRUNC_GSMONITOR_LOG_ON_OPEN} = \text{true} | \text{false} \\
\]

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” on page 389.
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_ADAPTER**
Logs messages related to adapters.

**D_AFS**
Logs information related to AFS credentials.

**D_CKPT**
Logs information related to checkpoint and restart

**D_DAEMON**
Logs information regarding basic daemon set up and operation, including information on the communication between daemons.

**D_DBX**
Bypasses certain signal settings to permit debugging of the processes as they execute in certain critical regions.

**D_DCE**
Logs information related to DCE credentials and Cluster Security Services identities. This flag is the same as the new **D_SECURITY** flag and is provided for compatibility.

**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.
Customizing the configuration file

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_SCHEDD
Displays how the schedd works internally.

D_SECURITY
Logs information related to DCE credentials and Cluster Security Services identities. This flag replaces the D_DCE flag.

D_STANZAS
Displays internal information about the parsing of the administration file.

D_STARTD
Displays how the startd works internally.

D_STARTER
Displays how the starter works internally.

D_STREAM
Displays messages detailing socket I/O.

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

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

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

For example,

SCHEDD_DEBUG = D_CKPT  D_XDR

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

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:
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.

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.
Customizing the configuration file

**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.

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>
<thead>
<tr>
<th>Keyword</th>
<th>Stanza</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ckpt_dir</td>
<td>Class</td>
<td>Initial working directory</td>
<td>The location to be used for checkpoint files</td>
</tr>
<tr>
<td>ckpt_time_limit</td>
<td>Class</td>
<td>Unlimited</td>
<td>Amount of time a job step can take to checkpoint</td>
</tr>
</tbody>
</table>

**Note:** For more information on these keywords see "Administration file keywords" on page 125.
Configuration file keyword summary:

Table 22. Configuration file keyword summary

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKPT_CLEANUP_INTERVAL</td>
<td>-1</td>
<td>How frequently, in seconds, the CKPT_CLEANUP_PROGRAM should be run</td>
</tr>
<tr>
<td>CKPT_CLEANUP_PROGRAM</td>
<td>No default</td>
<td>Identify an administrator provided program to be run at the interval specified by CKPT_CLEANUP_INTERVAL</td>
</tr>
<tr>
<td>MAX_CKPT_INTERVAL</td>
<td>7200</td>
<td>Maximum interval, in seconds, LoadLeveler will use for checkpointing running job steps</td>
</tr>
<tr>
<td>MIN_CKPT_INTERVAL</td>
<td>900 (15 minutes)</td>
<td>Initial (and minimum) interval, in seconds, LoadLeveler will use for checkpointing running job steps</td>
</tr>
</tbody>
</table>

Note: For more information on these keywords see “Configuration file keywords and LoadLeveler variables” on page 130.

Job command file keyword summary:

Table 23. Job command file keyword summary

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkpoint</td>
<td>No</td>
<td>Indicates if a job step should be enabled for checkpoint</td>
</tr>
<tr>
<td>ckpt_dir</td>
<td>The value of the ckpt_dir keyword in the class stanza of the administration file</td>
<td>The location to be used for checkpoint files</td>
</tr>
<tr>
<td>ckpt_file</td>
<td>[jobname.]job_step_id.ckpt</td>
<td>The base name to be used for checkpoint file</td>
</tr>
<tr>
<td>ckpt_time_limit</td>
<td>The value of the ckpt_time_limit keyword in the class stanza of the administration file</td>
<td>Amount of time a job step can take to checkpoint</td>
</tr>
<tr>
<td>restart_from_ckpt</td>
<td>No</td>
<td>Indicates if a job step is to be restarted from an existing checkpoint file</td>
</tr>
</tbody>
</table>

Note: For more information on these keywords see Chapter 12, “Job command file keywords,” on page 97.

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
Customizing the configuration file

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 397 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> specifies the job name in the Job Command File. If job_name is not specified, it is omitted from the default file name
- <job_step_id> Identifies the job step that is being checkpointed

- Checkpoint Directory Name
  1. The ckpt_file keyword in the job command file, if it contains a "/" as the first character
  2. The ckpt_dir keyword in the job command file
  3. The ckpt_dir keyword specified in the class stanza of the LoadLeveler admin file
  4. The default directory is the initial working directory

Note that two or more job steps running at the same time cannot both write to the same checkpoint file, since the file will be corrupted.

Naming checkpointing files for interactive parallel jobs: The following describes the order in which keywords and variables are checked to construct the full path name for the checkpoint file for an interactive parallel job.

- Checkpoint File Name
  1. The value of the MP_CKPTFILE environment variable within the POE process
  2. The default file name, poe.ckpt.<pid>

- Checkpoint Directory Name
  1. The value of the MP_CKPTFILE environment variable within the POE process, if it contains a full path name.
  2. The value of the MP_CKPTDIR environment variable within the POE process.
  3. The initial working directory.

Note: The keywords ckpt_dir and ckpt_file are not allowed in the command file for an interactive session. If they are present, they will be ignored and the job will be submitted.
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.

**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 393.

**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, mpxl_r, mpxl90_r, or mpxl95_r.
  - Processes that use:
    - Extended shmat support
    - Pinned shared memory segments.
Customizing the configuration file

- Sets of processes in which any process is running a **setuid** program when a checkpoint occurs.
- Sets of processes if any process is running a **setgid** program when a checkpoint occurs.
- Interactive parallel jobs for which POE input or output is a pipe.
- Interactive parallel jobs for which POE input or output is redirected, unless the job is submitted from a shell that had the CHECKPOINT environment variable set to yes before the shell was started. If POE is run from inside a shell script and is run in the background, the script must be started from a shell started in the same manner for the job to be checkpointable.
- Interactive POE jobs for which the **su** command was used prior to checkpointing or restarting the job.

  - The node on which a process is restarted must have:
    - 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.

- If the LoadLeveler cluster contains nodes running a mix of 32-bit and 64-bit kernels then applications must be checkpointed and restarted on the same set of nodes. See "lkckpt - Checkpoint a running job step" on page 162 and "restart_on_same_nodes" on page 119 for more information.

• For a parallel job, the number of tasks and the task geometry (the tasks that are common within a node) must be the same on a restart as it was when the job was checkpointed.

• Any regular file open in a process when it is checkpointed must be present on the node where that process is restarted, including the executable and any dynamically loaded libraries or objects.

• If any process uses sockets or pipes, user callbacks should be registered to save data that may be "in flight" when a checkpoint occurs, and to restore the data when the process is resumed after a checkpoint or restart. Similarly, any user shared memory in a parallel task should be saved and restored.

• A checkpoint operation will not begin on a process until each user thread in that process has released all pthread locks, if held. This can potentially cause a significant delay from the time a checkpoint is issued until the checkpoint actually occurs. Also, any thread of a process that is being checkpointed that does not hold any pthread locks and tries to acquire one will be stopped immediately. There are no similar actions performed for atomic locks (**_check_lock** and **_clear_lock**, for example).

• Atomic locks must be used in such a way that they do not prevent the releasing of pthread locks during a checkpoint. For example, if a checkpoint occurs and thread 1 holds a pthread lock and is waiting for an atomic lock, and thread 2 tries to acquire a different pthread lock (and does not hold any other pthread locks) before releasing the atomic lock that is being waited for in thread 1, the checkpoint will hang.

• A process must not hold a pthread lock when creating a new process (either implicitly using **popen**, for example, or explicitly using **fork**) if releasing the lock is contingent on some action of the new process. Otherwise, a checkpoint could occur which would cause the child process to be stopped before the parent could release the pthread lock causing the checkpoint operation to hang.

• The checkpoint operation will hang if any user pthread locks are held across:
Customizing the configuration file

- 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.
• SP Switch Communications Adapter Type 6–9 (Microchannel TB3 adapters) are not supported.
• A process can only checkpoint another process if it can send a signal to the process. In other words, the privilege checking for checkpointing processes is identical to the privilege checking for sending a signal to the process. A privileged process (the effective user ID is 0) can checkpoint any process. A set of processes can only be checkpointed if each process in the set can be checkpointed.
• A process can only restart another process if it can change its entire privilege state (real, saved, and effective versions of user ID, group ID, and group list) to match that of the restarted process. A set of processes can only be restarted if each process in the set can be restarted.
• The only DCE function supported is DCE credential forwarding by LoadLeveler using the DCE_AUTHENTICATION_PAIR configuration keyword. DCE credential forwarding is for the sole purpose of DFS™ access by the application.

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
Customizing the configuration file

For more information see “Ilckpt - Checkpoint a running job step” on page 162.

- Checkpoint from a program which invokes the ll_ckpt API to initiate checkpoint of a specific job step
  For more information see “Il_ckpt” on page 244.
- 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

<table>
<thead>
<tr>
<th>To specify that:</th>
<th>Do this:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your job is checkpointable</td>
<td>• Add either one of the following two options to your job command file:</td>
</tr>
<tr>
<td></td>
<td>1. checkpoint = yes</td>
</tr>
<tr>
<td></td>
<td>This enables your job to checkpoint in any of the following ways:</td>
</tr>
<tr>
<td></td>
<td>- The application can initiate the checkpoint</td>
</tr>
<tr>
<td></td>
<td>- Checkpoint from a program which invokes the ll_ckpt API</td>
</tr>
<tr>
<td></td>
<td>- Checkpoint using the llckpt command</td>
</tr>
<tr>
<td></td>
<td>- As the result of a flush command</td>
</tr>
<tr>
<td></td>
<td>OR</td>
</tr>
<tr>
<td></td>
<td>2. checkpoint = interval</td>
</tr>
<tr>
<td></td>
<td>This enables your job to checkpoint in any of the following ways:</td>
</tr>
<tr>
<td></td>
<td>- The application can initiate the checkpoint</td>
</tr>
<tr>
<td></td>
<td>- Checkpoint from a program which invokes the ll_ckpt API</td>
</tr>
<tr>
<td></td>
<td>- Checkpoint using the llckpt command</td>
</tr>
<tr>
<td></td>
<td>- Checkpoint automatically taken by LoadLeveler</td>
</tr>
<tr>
<td></td>
<td>- As the result of a flush command</td>
</tr>
</tbody>
</table>
|                        | • If you would like your job to checkpoint itself, use the API ll_init_ckpt in your serial application, or mpc_init_ckpt for parallel jobs to cause the checkpoint to occur.
Table 24. Checkpoint configurations (continued)

<table>
<thead>
<tr>
<th>To specify that:</th>
<th>Do this:</th>
</tr>
</thead>
</table>
| LoadLeveler automatically checkpoints your job at preset intervals | 1. Add the following option to your job command file: 
   ```
   checkpoint = interval
   ```
   This enables your job to checkpoint in any of the following ways:
   - Checkpoint automatically at preset intervals
   - Checkpoint initiated from user application
   - Checkpoint from a program which invokes the `ll_ckpt` API
   - Checkpoint using the `llckpt` command
   - As the result of a flush command

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:
   ```
   MIN_CKPT_INTERVAL = number
   MAX_CKPT_INTERVAL = number
   ```
   Where `number` specifies the initial period, in seconds, between checkpoints taken for running jobs.
   - `MIN_CKPT_INTERVAL` 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:
- The first checkpoint is taken after a period of time equal to the `MIN_CKPT_INTERVAL` has passed.
- The second checkpoint is taken after LoadLeveler waits `twice as long (MIN_CKPT_INTERVAL X 2)`.
- The third checkpoint is taken after LoadLeveler waits twice as long again (`MIN_CKPT_INTERVAL X 4`) before taking the third checkpoint.

LoadLeveler continues to double this period until the value of `MAX_CKPT_INTERVAL` has been reached, where it stays for the remainder of the job.

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.

<table>
<thead>
<tr>
<th>Your job will not be checkpointed</th>
<th>Add the following option to your job command file:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>checkpoint = no</code></td>
</tr>
<tr>
<td></td>
<td>This will disable checkpoint.</td>
</tr>
</tbody>
</table>
Customizing the configuration file

Table 24. Checkpoint configurations  (continued)

<table>
<thead>
<tr>
<th>To specify that</th>
<th>Do this:</th>
</tr>
</thead>
</table>
| 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. | 1. Add the following option to your job command file:  
   • restart_from_ckpt = yes  
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:  
   • ckpt_dir  
   • ckpt_file  
   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.  
   The job command file which was used to submit the original job should be used to restart from checkpoint. The only modifications to this file should be the addition of restart_from_ckpt = yes and ensuring ckpt_dir and ckpt_file point to the appropriate checkpoint file. |

| Your job has successfully checkpointed. The job has been vacated and remains on the LoadLeveler job queue. | When the job restarts, if a checkpoint file is available, the job will be restarted from that file.  
   If a checkpoint file is not available upon restart, the job will be started from the beginning. |

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/llckpt/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**.
Step 16: Configuring LoadLeveler to use a security service

LoadLeveler can be configured in one of two ways to control authentication and authorization of LoadLeveler functions:

1. DCE security services, which uses DCE as the underlying security mechanism.
2. Cluster Security Services (CtSec), a sub-component of Reliable Scalable Cluster Technology (RSCT) which uses, at the present time, the Host Based Authentication (HBA) as the underlying security mechanism.

LoadLeveler permits only one security service to be configured at a time. Cluster Security Services and DCE cannot both be configured as the security service for LoadLeveler. You can skip this section if you do not plan to use these security features or if you plan to use the DCE credential forwarding provided by the llgetdce and llsetdce program pair. Refer to "llgetdce and llsetdce" on page 316 for more information.

Customizing the configuration file

Note: PROCESS_TRACKING must be set as "true" for preemption to work.
For more information see "Restrictions for Gang scheduling preemption" on page 428.

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.

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

Linux notes
LoadLeveler for Linux does not support DCE or CtSec security.

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.

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 403, 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.

**Using SMIT and the lldcegrpmaint 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. FSSP security services use certain fields in the SDR (System Data Repository) to determine the current software configuration. Use the command "splstda -p" to verify that the field ts_auth_methods is set to either dce or dcecompat. If
**Customizing the configuration file**

`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]. 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 `lldcegrpmaint` command to automate this process. For more information on populating the `LoadL-services` group, see step [8]. 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.

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 "`llextSDR" to obtain a listing of DCE hostnames of nodes on an SP system.

8. Execute the command:

```
lldcegrpmaint 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 `lldcegrpmaint` 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 `lldcegrpmaint` command, see "`lldcegrpmaint` LoadLeveler DCE group maintenance utility" on page 174.

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
Customizing the configuration file

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 cds3l /.:/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 to 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.

   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/Startr
   dcecp -c principal create LoadL/c163n01.pok.ibm.com/GMonitor

   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 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
Customizing the configuration file

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 404 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 404. 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 -mpwd 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. The continuation character ”\” is not 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 404

   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 404. 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
Customizing the configuration file

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 405.

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 402, 6 on page 403, and 7 on page 403 of “Using SMIT and the ldcegrpmaint command” on page 402.

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 404 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

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, `lddelegate` and `llimpersonate` support credentials forwarding. See pages 411 and 315 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 `llgetdce` and `llsetdce` 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.
Customizing the configuration file

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.

Configuring LoadLeveler to use Cluster Security Services
Cluster Security Services allows a software component to authenticate and authorize the identity of one of its peers or clients.

When configured to use Cluster Security Services, LoadLeveler will:

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

In order to use Cluster Security Services, all nodes running LoadLeveler must first be configured as part of a cluster running Reliable Scalable Cluster Technology (RSCT) version 2.3.1.0 or greater. For details on Cluster Security Services administration, see IBM Reliable Scalable Cluster Technology for AIX 5L: RSCT Administration Guide, SA22-7889.

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

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

LoadLeveler uses Cluster Security Services to authorize owners of jobs, administrators and LoadLeveler daemons to perform certain actions. Cluster Security Services uses it’s own identity mapping file to map the clients’ network identity to a local identity when performing authorizations. A typical local identity is the user name without a hostname. The local identities of the LoadLeveler administrators must be added as members of the group specified by the keyword SEC_ADMIN_GROUP. The local identity of the LoadLeveler user name must be added as the sole member of the group specified by the keyword SEC_SERVICES_GROUP. The LoadLeveler Services and Administrative groups, those identified by the keywords SEC_SERVICES_GROUP and SEC_ADMIN_GROUP, must be the same across all nodes in the LoadLeveler cluster. To ensure consistency in performing tasks which require owner, administrative or daemon privileges across all nodes in the LoadLeveler cluster, user network identities must be mapped identically across all nodes in the LoadLeveler cluster. If this is not the case, LoadLeveler authorizations may fail.

Enabling Cluster Security Services: To enable LoadLeveler to use Cluster Security Services take the following steps:
1. Include, in the Trusted Host List, the host names of all hosts with which communications may take place. If LoadLeveler tries to communicate with a host not on the Trusted Host List the message: The host identified in the credentials is not a trusted host on this system will occur. Additionally, the system administrator should ensure that public keys are manually exchanged between all hosts in the LoadLeveler cluster. Refer to IBM Reliable Scalable Cluster Technology for AIX 5L: RSCT Administration Guide, SA22-7889 for information on setting up Trusted Host Lists and manually transferring public keys.

2. Create user IDs. Each LoadLeveler administrator and the LoadLeveler user ID need to be created, if they don’t already exist. You can do this through SMIT or the mkuser command.

3. The unix.map file must contain the correct value for the service name cloadl which specifies the LoadLeveler user name. If you have configured LoadLeveler to use loadl as the LoadLeveler user name, either by default or by specifying loadl in the LoadLUserid keyword of the /etc/LoadL.cfg file, nothing needs to be done. The default map file will contain the cloadl service name already assigned to loadl. If you have configured a different user name in the LoadLUserid keyword of the /etc/LoadL.cfg file, you will need to make sure that the /var/ct/cfg/unix.map file exists and that it assigns the same user name to the cloadl service name. If the /var/ct/cfg/unix.map file does not exist, create one by copying the default map file /usr/sbin/rsct/cfg/unix.map. Do not modify the default map file.

If the value of the LoadLUserid and the value associated with cloadl are not the same a security services error which indicates a Unix identity mis-match will occur.

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

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

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

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

```
unix:brad@==bradleyf
unix:marsha@==marshab
unix:loadl@==loadl
```
Customizing the configuration file

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

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

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

   ```
   mkgroup *users=sam,betty,loadl* lladmin
   ```

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

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

   ```
   mkgroup users=loadl llsvcs
   ```

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

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

   ```
   SEC_ENABLEMENT=CTSEC
   SEC_ADMIN_GROUP=**name of lladmin group**
   SEC_SERVICES_GROUP=**group name which contains the identities of the LoadLeveler daemons**
   ```

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

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

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

   ```
   HostNameField [UserNameField]
   ```

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

**Tips for configuring LoadLeveler to use Cluster Security Services**

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

If unexpected authentication or authorization problems occur in a LoadLeveler cluster with CtSec enabled, check that the steps in “Enabling Cluster Security Services” on page 407 were correctly followed for each machine in the LoadLeveler cluster.

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

• Users can not perform user tasks (such as cancel) for jobs they submitted.
  Either the machine the job was submitted from or the machine the user operation was submitted from (or both) do not contain mapping files for the user that specify the same security services identity. The user should attempt the operation from the same machine the job was submitted from and record the results. If the user still cannot perform a user task on a job they submitted, then they should contact the LoadLeveler administrator who should review the steps in “Enabling Cluster Security Services” on page 407.

• LoadLeveler daemons fail to communicate.
  When LoadLeveler daemons communicate they must first authenticate each other. If the daemons cannot authenticate a message will be put in the daemon log indicating an authentication failure. Ensure the Trusted Hosts List on all LoadLeveler nodes contains the correct entries for all of the nodes in the LoadLeveler cluster. Also, make sure that the LoadLeveler Services list on all nodes of the LoadLeveler cluster contains the local identity for the LoadLeveler user name. The ctsec_map.global must contain mapping rules to map the LoadLeveler user name from every machine in the LoadLeveler cluster to the local identity for the LoadLeveler user name. An example of what may happen when daemons fail to communicate is that an Alternate Central Manager may take over while the Primary Central Manager is still active. This can occur when the Alternate Central Manager does not trust the Primary Central Manager.

Limiting which security mechanisms LoadLeveler can use

As more security mechanisms become available, they must be configured for LoadLeveler’s use. If there are security mechanisms configured for CtSec which are not configured for LoadLeveler’s use, then the LoadLeveler configuration file keyword SEC_IMPOSED_MECHS must specify the mechanisms configured for LoadLeveler. Specify a blank delimited list containing combinations of the following values:

none If this is the only value specified, then users will run unauthenticated and, if authorization is necessary, the job will fail. If this is not the only value specified, then users may run unauthenticated and, if authorization is necessary, the job will fail.

unix If this is the only value specified, then UNIX host-based authentication will be used, otherwise, other mechanisms may be used.

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 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 317.

**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 315.

**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. For related information see 411 and 414.

**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**
Determines how many times a job can be rejected before it is canceled or put on hold. The default value is 0, which indicates a rejected job will immediately be canceled or placed on hold. **MAX_JOB_REJECT** may be set to unlimited rejects by specifying a value of −1. For related information, see the keyword 412.

**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_CYCLE_TIME_LIMIT** = *number*
Where *number* specifies the maximum time, in seconds, that LoadLeveler will allow the negotiator cycle to continue. The negotiator cycle will end, after the specified number of seconds, even if there are additional jobs waiting for dispatch. Jobs waiting for dispatch will be considered at the next negotiator cycle. The **NEGOTIATOR_CYCLE_TIME_LIMIT** keyword applies only to the BACKFILL and GANG schedulers. The number specified must be a positive integer or zero. If the keyword value is not specified or a value of zero is used, the negotiator cycle will be unlimited.

**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 378.

**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 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.

**REJECT_ON_RESTRICTED_LOGIN = true | false**

Where `true` specifies that the users account status will be checked on every node where the job will be run by calling the AIX `loginrestrictions` function with the `S_DIST_CLNT` flag. These checks include:

- Does the account still exist?
- Is the account locked?
- Has the account expired?
- Do failed login attempts exceed the limit for this account?
- Is login disabled via `/etc/nologin`?

If the AIX `loginrestrictions` function indicates a failure then the user’s job will be rejected and will be processed according to the LoadLeveler configuration parameters `MAX_JOB_REJECT` and `ACTION_ON_MAX_REJECT`.

**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
Customizing the configuration file

$(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.

**RESUME_ON_SWITCH_TABLE_ERROR_CLEAR = true | false**
If set to true it specifies that a startd which was drained when the switch table failed to unload will automatically resume once the unload errors are cleared. The unload error is considered cleared after LoadLeveler can successfully unload the switch table. The DRAIN_ON_SWITCH_TABLE_ERROR option in the configuration file must be turned on and not disabled in order for this keyword to work. Flushing, suspending, or draining of a startd manually or automatically will disable this option until the startd is manually resumed.

**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.

**UPDATE_ON_POLL_INTERVAL_ONLY = false | true**
This keyword specifies when the LoadLeveler startd daemons will send machine update transactions to the Central Manager. Normally the LoadLeveler startd daemons running on executing nodes will send transactions to the Central Manager to provide updates of machine information at various times. An update is sent every polling interval. The polling interval is calculated by multiplying the values for the two keywords, POLLING_FREQUENCY and POLLS_PER_UPDATE, specified in the LoadLeveler configuration file. In addition, updates are sent at other times such as when new jobs are started and when jobs terminate on the executing node. If you have a large and highly active cluster (the workload consists of a large number of short running jobs), the normal method for updating the central manager can add excessive network traffic. **UPDATE_ON_POLL_INTERVAL_ONLY** can help reduce this source of network traffic. When true is specified, the LoadLeveler startd daemon will only send machine updates to the Central Manager at every polling interval and not at other times.

**VM_IMAGE_ALGORITHM = FREE_PAGING_SPACE | FREE_PAGING_SPACE_PLUS_FREE_REAL_MEMORY**
This keyword is used together with the large_page job command file keyword to specify which algorithm the Central Manager uses to decide whether a machine has enough virtual memory to run a job step. When **FREE_PAGING_SPACE** is specified, LoadLeveler considers only free paging space when determining if a machine has enough virtual memory to run a job step. When **FREE_PAGING_SPACE_PLUS_FREE_REAL_MEMORY** is specified and the job step specifies large_page=N (does not use Large Page memory), LoadLeveler considers free paging space and free regular memory when determining if a machine has enough virtual memory to run a job step. When **FREE_PAGING_SPACE_PLUS_FREE_REAL_MEMORY** is specified and the job step specifies large_page=Y (uses Large Page memory, if available), LoadLeveler considers free paging space, free regular memory, and free Large Page memory when determining if a machine has enough virtual memory to run a job step, although Large Page memory is only considered for machines configured to exploit the Large Page feature. When **FREE_PAGING_SPACE_PLUS_FREE_REAL_MEMORY** is specified and the job step specifies large_page=M (must use Large Page memory), LoadLeveler considers only Large Page memory when determining if a machine has enough
virtual memory to run a job step. Only machines configured to exploit the Large Page feature are considered. The default setting for this keyword is `FREE_PAGING_SPACE`.

It is recommended that this keyword be set to the value `FREE_PAGING_SPACE_PLUS_FREE_REAL_MEMORY` since more types of virtual memory are considered, increasing the chances of finding a machine with enough virtual memory to run the job step.

This keyword is critical for job steps that must use Large Page memory (specified by the job command file keyword `large_page=M`). If this keyword is set to `FREE_PAGING_SPACE` the Large Page job step will never be scheduled to run. This keyword must be set to `FREE_PAGING_SPACE_PLUS_FREE_REAL_MEMORY` to run Large Page jobs.

`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:

<table>
<thead>
<tr>
<th>Edit this keyword:</th>
<th>To:</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBAL_HISTORY</td>
<td>Specify a directory in which to place the global history files.</td>
</tr>
<tr>
<td>ACCT</td>
<td>Turn accounting and account validation on and off and specify detailed accounting.</td>
</tr>
<tr>
<td>ACCTVALIDATION</td>
<td>Specify the account validation routine.</td>
</tr>
</tbody>
</table>

**Note:** See “Step 9: Define job accounting” on page 384 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 158 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.
Sample job accounting scenario

Task 3: Report job information on all the jobs in the history file

You can accomplish this step using either the Ilsummary command or the graphical user interface:

- **Using Ilsummary:** see “Ilsummary - Return job resource information for accounting” on page 228 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 12, “Job command file keywords,” on page 97 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.

Assume Figure 34 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 34. Environment illustrating jobs being routed to NQS machines.](image-url)
Routing jobs 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:

```plaintext
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:

```plaintext
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:

```plaintext
class = NQS
```

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
Routing jobs to NQS machines

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

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.
Routing jobs to NQS machines

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, `llsubmit` 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`. `llsubmit` 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.
Chapter 18. Preemption using the Gang scheduler

Overview

In the previous release, time-sharing was included in gang scheduling. For the current release, gang scheduling is used exclusively for preemption. Keywords relating to time-sharing are obsolete.

Gang scheduling enables preemption, which allows for a running job step to be suspended so that another job step can run. Some of the resources are released from preempted job steps. These resources are made available to the preemiting job steps. When preemiting job steps end, the preempted job can resume.

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 will 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.

   For more information see "llmodify - Change attributes of a submitted job step" on page 191.

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 if PREEMPTION_TYPE is set to full in configuration file
- Scheduling slots
- Real memory
- ConsumableCpus and ConsumableMemory
Preemption using the Gang scheduler

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.

Keywords specific to Gang scheduling

Configuration file keywords for Gang scheduling preemption
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 428.

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

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Default value</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>GANG_MATRIX_TIME_SLICE</td>
<td>obsolete keyword—do not use</td>
<td>NA</td>
</tr>
<tr>
<td>GANG_MATRIX_NODE_SUBSET_SIZE</td>
<td>obsolete keyword—do not use</td>
<td>NA</td>
</tr>
<tr>
<td>GANG_MATRIX_REORG_CYCLE</td>
<td>obsolete keyword—do not use</td>
<td>NA</td>
</tr>
<tr>
<td>GANG_MATRIX_BROADCAST_CYCLE</td>
<td>obsolete keyword—do not use</td>
<td>NA</td>
</tr>
<tr>
<td>PREEMPT_CLASS [classname]</td>
<td>No preemption</td>
<td>NA</td>
</tr>
<tr>
<td>START_CLASS [classname]</td>
<td>No restriction</td>
<td>NA</td>
</tr>
<tr>
<td>PREEMPTION_SUPPORT</td>
<td>FULL</td>
<td>FULL, NONE, NO_ADAPTER</td>
</tr>
</tbody>
</table>

Configuration file keyword details

GANG_MATRIX_TIME_SLICE
Obsolete keyword.
Preemption using the Gang scheduler

GANG_MATRIX_NODE_SUBSET_SIZE
Obsolete keyword.

GANG_MATRIX_REORG_CYCLE
Obsolete keyword.

GANG_MATRIX_BROADCAST_CYCLE
Obsolete keyword.

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 428 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 preemptible 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}
Preemption using the Gang scheduler

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.

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.

PREEMPTION_SUPPORT= full | no_adapter | none
For Gang or API schedulers, specifies the level of preemption support for a cluster. When set to full, preemption is fully supported. When set to no_adapter, preemption is supported but the adapter resources are not released by preemption. When set to none, preemption is not supported. The default value for GANG schedulers is full, but the default value for
Preemption using the Gang scheduler

all other schedulers is none. When set to full or no_adapter, LoadLeveler checks that other conditions such as MACHINE_AUTHENTICATE = TRUE and PROCESS_TRACKING = TRUE were set at start time before preemption requests are accepted by the negotiator. If this keyword is set to none preemption requests are rejected by the negotiator.

Sample configuration file

The following sample illustrates the configuration file (LoadL_config) for using the preemption function with the Gang scheduler:

```
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}

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
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 428.
Preemption using the Gang scheduler

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
   See "Supported hardware" on page 422 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

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Default value</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>execution_factor</td>
<td>1</td>
<td>1,2,3</td>
</tr>
<tr>
<td>max_smp_tasks</td>
<td>—</td>
<td>Obsolete keyword</td>
</tr>
<tr>
<td>max_total_tasks</td>
<td>-1</td>
<td>-1, 0, or any positive integer</td>
</tr>
</tbody>
</table>

Administration keyword details

**execution_factor**
This keyword appears in the Class stanza. A value of 1 means the job step is preemptable. The values 2 and 3 are allowed for compatibility reasons and mean the same as 1. The value 99 is used to prevent a job from being preempted and is set on an individual job step, not in the Class stanza. Refer to "llmodify - Change attributes of a submitted job step" on page 191 for more information.
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**
Obsolete keyword.

**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 preemption:
```
default:  type = machine
          pool_list = 1
```

426 Using and Administering LoadLeveler
Preemption using the Gang scheduler

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

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
    alias = c163sn02.ppd.pok.ibm.com
    central_manager = true

c163n03.ppd.pok.ibm.com: type = machine
    adapter_stanzas = c163sn03.ppd.pok.ibm.com
    alias = c163sn03.ppd.pok.ibm.com

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

Chapter 18. Preemption using the Gang scheduler  427
Gang scheduling interactions and restrictions

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

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 preemption
The following conditions are not supported with Gang scheduling 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. \( \text{START\_CLASS}[\text{Class\_B}] = (\text{Class\_A} < 1) \)
2. \( \text{START\_CLASS}[\text{Class\_C}] = (\text{Class\_A} < 1) \)

LoadLeveler adds all implied \( \text{START\_CLASS} \) expressions to the \( \text{START\_CLASS} \) expressions specified in the configuration file. This overrides any existing values for \( \text{START\_CLASS} \).

For example, if the configuration file contains the following statements:

\[
\begin{align*}
\text{PREEMPT\_CLASS}[\text{Class}\_A] &= \text{ALL} \ {\text{Class}\_B, \text{Class}\_C} \\
\text{START\_CLASS}[\text{Class}\_B] &= (\text{Class}\_A < 5) \\
\text{START\_CLASS}[\text{Class}\_C] &= (\text{Class}\_C < 3)
\end{align*}
\]

When LoadLeveler runs through the configuration process, the \( \text{PREEMPT\_CLASS} \) statement on the first line generates the two implied \( \text{START\_CLASS} \) statements. When the implied \( \text{START\_CLASS} \) statements are overridden and the resulting \( \text{START\_CLASS} \) statements are effectively equivalent to:

\[
\begin{align*}
\text{START\_CLASS}[\text{Class}\_B] &= (\text{Class}\_A < 1) \\
\text{START\_CLASS}[\text{Class}\_C] &= (\text{Class}\_C < 3) \&\& (\text{Class}\_A < 1)
\end{align*}
\]

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 \( \text{START\_CLASS} \) 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 \( \text{START\_CLASS} \):

\[
\begin{align*}
\text{START\_CLASS}[\text{Class}\_B] &= (\text{Class}\_A < 1) \\
\text{START\_CLASS}[\text{Class}\_B] &= (\text{Class}\_B < 1) \\
\text{START\_CLASS}[\text{Class}\_B] &= (\text{Class}\_C < 1)
\end{align*}
\]

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).

**Job command file and Gang scheduling**

The `preferences` keyword is ignored when using Gang scheduling.

### LoadLeveler commands for Gang

<table>
<thead>
<tr>
<th>Command</th>
<th>Detailed on page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>llclass</code></td>
<td>165</td>
</tr>
<tr>
<td><code>llmatrix</code></td>
<td>The Gang matrix is no longer in use.</td>
</tr>
<tr>
<td><code>llmodify</code></td>
<td>191</td>
</tr>
<tr>
<td><code>lpreempt</code></td>
<td>194</td>
</tr>
<tr>
<td><code>llq</code></td>
<td>198</td>
</tr>
<tr>
<td><code>llsubmit</code></td>
<td>226</td>
</tr>
</tbody>
</table>
Preemption using the Gang scheduler

APIs used with Gang scheduling

The following APIs have been created or modified for Gang scheduling preemption:

- "Data Access API" on page 247
- "Error Handling API" on page 285
- "ll_preempt subroutine" on page 305
- "ll_modify subroutine" on page 302
Chapter 19. 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. Refer to the allowable units for these limits listed under “Enforcing limits” on page 358.

**Note:** In LoadLeveler, the hard and soft time limits associated with the keywords **cpu_limit, job_cpu_limit, wall_clock_limit, ckpt_time_limit** are 32-bit integers. If a value that cannot be contained in a 32-bit integer is assigned to one of these limits it will be truncated to either 2147483647 or -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:**

```
requirements = (Arch == "R6000") && (Disk > 5000000000) && (Memory > 6000000000)
preferences = (Disk > 6000000000) && (Memory > 9000000000)
```

**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.
64-bit applications

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. They are 64-bit integers and 64-bit arithmetic is used to evaluate them.

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, lllstatus -l, and llq -l commands on a 64-bit LoadLeveler cluster.

llclass -l
Sample output from llclass -l command is illustrated in Figure 35.

```plaintext
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, 164926741664 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, 619249487634432 bytes)
```

Figure 35. Sample output from llclass -l command

llstatus -l
Sample output from llstatus -l command is illustrated in Figure 36.

```plaintext
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 36. Sample output from llstatus -l command
64-bit applications

Ilq -l
Sample output from llstatus -l command is illustrated in Figure 37.

Job Step Id: c209f1n05.ppd.pok.ibm.com.2.0
... 
Owner: load1
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 (13835058055216384 bytes)
File Soft Limit: 1.100 eb (126821365506731680 bytes)
Stack Hard Limit: 40.000 mb (41943040 bytes)
Stack Soft Limit: 30.000 mb (31457280 bytes)
Rss Hard Limit: 1.200 eb (13835058055216384 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 37. Sample output from ilq -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 3.2, 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 or SEC_ENABLEMENT=DCE), 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.

64-bit support for Accounting functions

LoadLeveler 64-bit support for accounting functions includes:
• Statistics of jobs such as usage, limits, consumable resources, and other 64-bit integer data are preserved in the history file as rusage64, rlimit64 structures and as data items of type int64_t.

• The LL_job_step structure defined in llapi.h allows access to the 64-bit data items either as data of type int64_t or as data of type int32_t. In the latter case, the returned values may be truncated.

• The llsummary command displays 64-bit information where appropriate.

• The data access API supports both 64–bit and 32–bit access to accounting and usage information in a history file. Please refer to the code fragment on page 282 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 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 441.

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 38:

```
# 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 38. 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 226.

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
```
Using commands

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 198.

**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 51.

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 196.

**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 187.

**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 187.

**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 216.

**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 160.

**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 216.

**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 46 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 40 on page 443. 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 39 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 39 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.

```bash
# @ executable = longjob
# @ input = longjob.in.$(stepid)
# @ output = longjob.out.$(jobid).$(stepid)
# @ error = longjob.err.$(jobid).$(stepid)
# @ queue
# @ queue
# @ queue
# @ queue
```

Figure 39. Job command file with varying input statements

To submit the job, type the command:

```bash
llsubmit longjob.cmd
```

LoadLeveler responds by issuing the following:

```
submit: The job "ll6.23" 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:

<table>
<thead>
<tr>
<th>Job Step</th>
<th>Standard Input</th>
<th>Standard Output</th>
<th>Standard Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>ll6.23.0</td>
<td>longjob.in.0</td>
<td>longjob.out.23.0</td>
<td>longjob.err.23.0</td>
</tr>
<tr>
<td>ll6.23.1</td>
<td>longjob.in.1</td>
<td>longjob.out.23.1</td>
<td>longjob.err.23.1</td>
</tr>
<tr>
<td>ll6.23.2</td>
<td>longjob.in.2</td>
<td>longjob.out.23.2</td>
<td>longjob.err.23.2</td>
</tr>
</tbody>
</table>
### Additional examples

<table>
<thead>
<tr>
<th>Job Step</th>
<th>Standard Input</th>
<th>Standard Output</th>
<th>Standard Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>ll6.23.3</td>
<td>longjob.in.3</td>
<td>longjob.out.23.3</td>
<td>longjob.err.23.3</td>
</tr>
<tr>
<td>ll6.23.4</td>
<td>longjob.in.4</td>
<td>longjob.out.23.4</td>
<td>longjob.err.23.4</td>
</tr>
</tbody>
</table>

**Example 2: Using LoadLeveler variables in a job command file**

Figure 40 on page 443 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.
Additional examples

Figure 40. 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 41 on page 445 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 that 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 `ll5.pok.ibm.com` for execution because of the explicit requirement for this machine in the requirements statement.
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+ (SP2MP/ architecture)
- MPICH
- MPICH-GM

Figure 42 on page 446 is a sample job command file for POE.
Additional examples

```plaintext
# @ job_type = parallel
# @ environment = COPY_ALL
# @ output = poe.out
# @ error = poe.error
# @ node = 8,10
# @ tasks_per_node = 2
# @ network.LAPI = sn_all,US,,instances=1
# @ network.MPI = sn_all,US,,instances=1
# @ wall_clock_limit = 60
# @ executable = /usr/bin/poe
# @ arguments = /u/richc/My_POE_program -euilib "us"
# @ class = POE
# @ queue
```

**Figure 42. POE job command file – multiple tasks per node**

Figure 42 shows the following:
- The total number of nodes requested is a minimum of eight and a maximum of 10 (node=8,10). Two tasks run on each node (tasks_per_node=2). Thus the total number of tasks can range from 16 to 20.
- Each task of the job will run using the LAPI protocol in US mode with a switch adapter (network.LAPI=sn_all,US,,instances=1), and using the MPI protocol in US mode with a switch adapter (network.MPI=sn_all,US,,instances=1).
- The maximum run time allowed for the job is 60 seconds (wall_clock_limit=60).

```plaintext
# @ job_type = parallel
# @ input = poe.in.1
# @ output = poe.out.1
# @ error = poe.err
# @ node = 2,8
# @ network.MPI = sn_single,shared,IP
# @ wall_clock_limit = 60
# @ class = POE
# @ queue
/usr/bin/poe /u/richc/my_POE_setup_program -infolevel 2
/usr/bin/poe /u/richc/my_POE_main_program -infolevel 2
```

**Figure 43. POE sample job command file – invoking POE twice**

Figure 43 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 a switch adapter in IP mode (network.MPI=sn_single,shared,IP).
- The maximum run time allowed for the job is 60 seconds (wall_clock_limit=60).
PVM 3.3 (non-SP)

Figure 44 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.

```plaintext
# @ 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 44. 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 45 on page 448 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.
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 then you actually need to run the parallel job. PVM spawns an additional task to relay messages to and from the PVM daemons. 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 64.
- 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.

```bash
#!/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 45. Sample PVM 3.3.11+ (SP2MPI Architecture) job command file
• 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 45 on page 448.

Figure 46 on page 450 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

MPICH

Figure 47 on page 451 is a sample job command file for MPICH.

Figure 46. Sequence of events in a PVM 3.3.11+ job
Figure 47 shows the following:

- The operation associated with the statement
  
  `/common/NFS/ll_bin/ll_get_machine_list > /tmp/machinelist.$LOADL_STEP_ID`
  
  creates a temporary file that contains the list of machines that have been
  assigned by LoadLeveler to this parallel job step. `$LOADL_STEP_ID` is an
  environment variable set by LoadLeveler. The source file `ll_get_machine_list.c`
  and the script `mpich_cmp.sh` that can be used to compile this file are located in
  the `/opt/ibmll/LoadL/full/samples/linux/llmpich` directory. `ll_get_machine_list`
  uses the LoadLeveler Data Access API to query the appropriate `LoadL_schedd`
  daemon for the machine list information.

- The statement `machine_count=`'cat /tmp/machinelist.$LOADL_STEP_ID | wc -l`
  counts the number of entries in the machine list. The number of entries in the
  machine list is equal to the number of parallel tasks.

- In the job command file statement
  
  `/opt/mpich/bin/mpirun -np $machine_count -machinefile /tmp/machinelist.$LOADL_STEP_ID`
  
  `/common/NFS/ll_bin/mpich_test`
  
  – `-np` specifies the number of parallel processes.
  – `-machinefile` specifies the machine list file.

- `rm /tmp/machinelist.$LOADL_STEP_ID` removes the temporary machine list
  file once the application completes.

Cancelling an MPICH job step with the `llcancel` command terminates only the
`mpirun` script associated with the job step. It does not terminate all the tasks
started by this `mpirun` script. This is also true for cancellation operations started
by the `llctl flush` and `llctl stop` commands.
MPICH-GM

Figure 48 is a sample job command file for MPICH-GM.

```bash
# LoadLeveler JCF file for running an MPICH-GM job
# @ job_type = parallel
# @ resources = gmports(1)
# @ node = 4
# @ tasks_per_node = 2
# @ output = mpich_gm_test.$(cluster).$(process).out
# @ error = mpich_gm_test.$(cluster).$(process).err
# @ queue
echo "------------------------------------------------------------"
echo LOADL_STEP_ID=$LOADL_STEP_ID
echo "------------------------------------------------------------"

# Make sure that the ll_get_machine_list binary is accessible on all machines
# in the LoadLeveler cluster.
/common/NFS/ll_bin/ll_get_machine_list > /tmp/machinelist.$LOADL_STEP_ID

machine_count=`cat /tmp/machinelist.$LOADL_STEP_ID | wc -l`
echo $machine_count
echo MachineList:
cat /tmp/machinelist.$LOADL_STEP_ID
echo "------------------------------------------------------------"
/opt/mpich/bin/mpirun.ch_gm --gm-kill 0 -np $machine_count -machinefile /
/tmp/machinelist.$LOADL_STEP_ID /common/NFS/ll_bin/mpich_gm_test
rm /tmp/machinelist.$LOADL_STEP_ID
```

Figure 48. MPICH-GM job command file

- The statement `# @ resources = gmports(1)` specifies that each task consumes one GM port. This is how LoadLeveler limits the number of GM ports simultaneously in use on any machine. This resource name is the name the LoadLeveler administrator specified in `schedule_by_resources` in the configuration file and each machine stanza in the administration file must define GM ports and specify the quantity of GM ports available on each machine. Use `llstatus -R` to confirm the names and values of the configured and available consumable resources.

- The operation associated with the statement
  
  `/common/NFS/ll_bin/llmpich/ll_get_machine_list>/tmp/machinelist.$LOADL_STEP_ID`
  
  creates a temporary file that contains the list of machines that have been assigned by LoadLeveler to this parallel job step. `LOADL_STEP_ID` is an environment variable set by LoadLeveler. The source file `ll_get_machine_list.c` and the script `mpich_gm_cmp.sh` that can be used to compile this file are located in the `/opt/ibml/LoadL/full/samples/linux/llmpich` directory.

  `ll_get_machine_list` uses the LoadLeveler Data Access API to query the appropriate `LoadL_schedd` daemon for the machine list information.

- The statement `machine_count=` shows the following:
  
  `cat /tmp/machinelist.$LOADL_STEP_ID | wc -l` counts the number of entries in the machine list. The number of entries in the machine list is equal to the number of parallel tasks.

- In the job command file statement `/opt/mpich/bin/mpirun.ch_gm --gm-kill 0 -np $machine_count -machinefile /tmp/machinelist.$LOADL_STEP_ID /common/NFS/ll_bin/mpich_gm_test:`

452 Using and Administering LoadLeveler
Additional examples

- `/opt/mpich/bin/mpirun.ch_gm` specifies the location of the `mpirun.ch_gm` script shipped with the MPICH-GM implementation that runs the MPICH-GM application.

- `--gm-kill 0` indicates that all parallel tasks are stopped when any one task ends. This option is necessary for MPICH-GM jobs to respond properly to an `llcancel` command.

- `-np` specifies the number of parallel processes.

- `-machinefile` specifies the machine list file.

  - `rm /tmp/machinelist.$LOADL_STEP_ID` removes the temporary machine list file once the application completes.

Administrator tasks: using an external scheduler

The LoadLeveler API provides interfaces that allow an external scheduler to manage the assignment of resources to jobs and dispatching those jobs. The primary interfaces for the tasks of an external scheduler are:

- `ll_query` to obtain information about the LoadLeveler cluster, the machines of the cluster, jobs and AIX Workload Manager

- `ll_get_data` to obtain information about specific objects such as jobs, machines and adapters

There are two interfaces for starting a LoadLeveler job: `ll_start_job` and `ll_start_job_ext`. Both support starting serial and parallel jobs but the capabilities for starting parallel jobs with `ll_start_job` are limited and `ll_start_job_ext` is preferred for starting parallel jobs. Either function can be used to start a serial job.

The steps for dispatching jobs with an external scheduler are:

1. Gather information about the LoadLeveler cluster (`ll_query(CLUSTER)`).
2. Gather information about the machines in the LoadLeveler cluster (`ll_query(MACHINES)`).
3. Gather information about the jobs in the cluster (`ll_query(JOBS)`).
4. For each job that has completed, restore the resources it used to available.
5. Determine which job(s) to start. Assign resources to job(s) to be started and dispatch (`ll_start_job_ext(LL_start_job_info_ext*)`).
6. Repeat

For more detailed information to help with steps 2 and 3, refer to “Getting information.” For help with step 5, refer to “Assign resources and dispatch” on page 457.

Getting information

Getting information about the LoadLeveler cluster, machines, jobs and AIX Workload Manager is a four-step process. First, you create a query object for the kind of information you want. For example, to query machine information, you would:

```
LL_element * query_element = ll_query(MACHINES);
```

Next, you customize the query to filter the specific information you want. You can filter the list of objects for which you want information. For some queries, you can also filter how much information you want. For example, the following lines customize the query for just hosts node01.ibm.com and node02.ibm.com and to return the information contained in the `llstatus -f` command:
Additional examples

```c
char * hostlist[] = { "node01.ibm.com","node02.ibm.com",NULL };    
ll_set_request(query_element,QUERY_HOST,hostlist,STATUS_LINE);
```

Once the query has been customized, it is submitted with `ll_get_objs` which returns the first object that matches the query. This object is interrogated with the `ll_get_data` command to retrieve specific attributes. Depending on the information being queried for, the query may be directed to a specific node and a specific daemon on that node. For example, a `JOBS` query for all data may be directed to the negotiator, schedd or the history file. If it is directed to schedd, you must specify the host of the schedd you are interested in. The following demonstrates retrieving the name of the first machine returned by the query constructed previously:

```c
int machine_count;    
int rc;    
LL_element * element =ll_get_objs(query_element,LL_CM,NULL,&machine_count,&rc)    
char * mname;    
ll_get_data(element,LL_MachineName,&mname);
```

Because there is only one negotiator in a LoadLeveler cluster, the host does not have to be specified. The third parameter is the address of an integer that will receive the count of objects returned and the fourth parameter is the address of an integer that will receive the completion code of the call. If the call fails, NULL is returned and the location pointed to by the fourth parameter is set to a reason code. If the call succeeds, the value returned is used as the first parameter to a call to `ll_get_data`. The second parameter to `ll_get_data` is a specification that indicates what attribute of the object is being interrogated. The third parameter to `ll_get_data` is the address of the location into which to store the result. `ll_get_data` returns zero if it is successful and non-zero if an error occurs. It is important that the specification (the second parameter to `ll_get_data`) be valid for the object passed in (the first parameter) and that the address passed in as the third parameter point to the correct type for the specification. Undefined, potentially dangerous behavior will occur if either of these conditions is not met.

The following example demonstrates printing out the name and adapter list of all machines in the LoadLeveler cluster:

```c
int i, w, rc;    
int machine_count;    
LL_element * query_elem;    
LL_element * machine;    
LL_element * adapter;    
char * machine_name;    
char * adapter_name;    
int * window_list;    
int window_count;    
    
/* First we need to obtain a query element which is used to pass */    
/* parameters in to the machine query */    
if ((query_elem = ll_query(MACHINES)) == NULL)    
  {    
    fprintf(stderr,"Unable to obtain query element\n");    
    /* without the query object we will not be able to do anything */    
    exit(-1);    
  }    
    
/* Get information relating to machines in the LoadLeveler cluster. */    
/* QUERY_ALL: we are querying all machines */    
/* NULL: since we are querying all machines we do not need to */    
/* specify a filter to indicate which machines */    
/* ALL_DATA: we want all the information available about the machine */    
rc=ll_set_request(query_elem,QUERY_ALL,NULL,ALL_DATA);
```
if(rc<0) {
  /* A real application would map the return code to a message */
  printf("%d returned from ll_set_request\n\n",rc);
  /* Without customizing the query we cannot proceed */
  exit(rc);
}

/* If successful, ll_get_objs() returns the first object that */
/* satisfies the criteria that are set in the query element and */
/* the parameters. In this case those criteris are: */
/* A machine (from the type of query object) */
/* LL CM: that the negotiator knows about */
/* NULL: since there is only one negotiator we don't have to */
/* specify which host it is on */
/* The number of machines is returned in machine_count and the */
/* return code is returned in rc */
machine = ll_get_objs(query_elem,LL_CM,NULL,&machine_count,&rc);
if(rc<0) {
  /* A real application would map the return code to a message */
  printf("%d returned from ll_get_objs\n\n",rc);

  /* query was not successful -- we cannot proceed but we need to */
  /* release the query element */
  if(ll_deallocate(query_elem) == -1) {
    fprintf(stderr,"Attempt to deallocate invalid query element\n\n");
  }
  exit(rc);
}

printf("Number of Machines = \%d\n", machine_count);
i = 0;
while(machine!=NULL) {
  printf("------------------------------------------\n\n");
  printf("Machine \%d\n", i);

  int rc = ll_get_data(machine,LL_MachineName,&machine_name);
  if(0==rc) {
    printf("Machine name = \%s\n",machine_name);
  } else {
    printf("Error \%d returned attempting to get machine name\n\n",rc);
  }

  printf("Adapters\n");
  ll_get_data(machine,LL_MachineGetFirstAdapter,&adapter);
  while(adapter != NULL) {
    rc = ll_get_data(adapter,LL_AdapterName,&adapter_name);
    if(0!=rc) {
      printf("Error \%d returned attempting to get adapter name\n\n",rc);
    } else {
      /* Because the list of windows on an adapter is returned */
      /* as an array of integers, we also need to know how big */
      /* the list is. First we query the window count, */
      /* storing the result in an integer, then we query for */
      /* the list itself, storing the result in a pointer to */
      /* an integer. The window list is allocated for us so */
      /* we need to free it when we are done */
  
  
  

Appendix A. Examples 455
printf("%s
",adapter_name);
ll_get_data(adapter,LL_AdapterTotalWindowCount,&window_count);
ll_get_data(adapter,LL_AdapterWindowList,&window_list);
for (w = 0; w < iBuffer; w++)
{
    printf("%d
",window_list[w]);
}
free(window_list);
/* After the first object has been gotten, GetNext returns */
/* the next until the list is exhausted */
ll_get_data(machine,LL_MachineGetNextAdapter,&adapter);
printf("\n");
i++;
machine = ll_next_obj(query_elem);
}
/* First we need to release the individual objects that were */
/* obtained by the query */
if(ll_free_objs(query_elem) == -1)
{
    fprintf(stderr,"Attempt to free invalid query element\n");
}
/* Then we need to release the query itself */
if(ll_deallocate(query_elem) == -1)
{
    fprintf(stderr,"Attempt to deallocate invalid query element\n");
}

The example above could be extended to retrieve all of the information available about the machines in the cluster such as memory, disk space, pool list, features, supported classes, and architecture, among other things. A similar process would be used to retrieve information about the cluster overall. The next step would be to retrieve information about the jobs in the cluster. The following example demonstrates retrieving information about jobs up to the point of starting a job:

int i, rc;
int job_count;
LL_element * query_elem;
LL_element * job;
LL_element * step;
int step_state;

/* First we need to obtain a query element which is used to pass */
/* parameters in to the jobs query */
if ((query_elem = ll_query(JOBS)) == NULL)
{
    fprintf(stderr,"Unable to obtain query element\n");
    /* without the query object we will not be able to do anything */
    exit(-1);
}
/* Get information relating to Jobs in the LoadLeveler cluster. */
printf("Jobs Information ==============================================================\n");
/* QUERY_ALL: we are querying all jobs */
/* NULL: since we are querying all jobs we do not need to */
/* specify a filter to indicate which jobs */
/* ALL_DATA: we want all the information available about the job */
rc = ll_set_request(query_elem, QUERY_ALL, NULL, ALL_DATA);
if (rc < 0)
{
    Additional examples

Using and Administering LoadLeveler

456
Assign resources and dispatch

In the previous example, we reached the point where a step to start was identified. In a real external scheduler, the decision would be reached after consideration of all the idle jobs and constructing a priority value based on attributes such as class and submit time, all of which are accessible through `ll_get_data`. Next, the list of
Additional examples

available machines would be examined to determine whether a set exists with sufficient resources to run the job. This process also involves determining the size of that set of machines using attributes of the step such as number of nodes, instances of each node and tasks per node. The LoadLeveler data query API allows access to that information about each job but the interface for starting the job does not require that the machine and adapter resource match the specifications when the job was submitted. For example, a job could be submitted specifying node=4 but could be started by an external scheduler on a single node only. Similarly, the job could specify the LAPI protocol with network.lapi=... but be started and told to use the MPI protocol. This is not considered an error since it is up to the scheduler to interpret (and enforce, if necessary), the specifications in the job command file.

In allocating adapter resources for a step, it is important that the order of the adapter usages be consistent with the structure of the step. In environments where a task can use multiple adapter windows for a protocol, LoadLeveler expects those usages to be contiguous in the list of adapter usages passed in to the ll_start_job_ext call. Similarly, all the usages for a single task must be contiguous. In other words, if a task is expected to use three adapter windows for MPI and two for LAPI, there must be three MPI usages together followed immediately by two LAPI usages. Each task must be allocated the same adapter resources so that five element pattern must be repeated for each task.

All of the job command file keywords for specifying job structure such as total_tasks, tasks_per_node, node=min,max and blocking are supported by the ll_start_job_ext() interface but users should ensure that they understand the LoadLeveler model that is created for each combination when constructing the adapter usage list for ll_start_job_ext. Jobs that are submitted with node=number and tasks_per_node result in more regular LoadLeveler models and are easier to create adapter usage lists for.

In the following example, it is assumed that the step found to be dispatched will run on one machine with two tasks, each task using one switch adapter window for MPI communication. The name of the machine to run on is contained in the variable use_machine (char *), the names of the switch adapters are contained in use_adapter_1 (char *) and use_adapter_2 (char *) and the adapter windows on those adapters in use_window_1 int and use_window_2 (int), respectively. Further more, each adapter will be allocated 1M of memory.

```c
LL_start_job_info_ext *start_info;
char *pChar;
LL_element *step;
LL_element *job;
int rc;
char *submit_host;
char *step_id;

start_info = (LL_start_job_info_ext *)malloc(sizeof(LL_start_job_info_ext));
if(start_info == NULL) {
    fprintf(stderr, "Out of memory.
");
    return;
}

/* Create a NULL terminated list of target machines. Each task */
/* must have an entry in this list and the entries for tasks on the */
/* same machine must be sequential. For example, if a job is to run */
/* on two machines, A and B, and three tasks are to run on each */
/* machine, the list would be: AAABBB */
/* Any specifications on the job when it was submitted such as */
```
Additional examples

/* nodes, total_tasks or tasks_per_node must be explicitly queried */
/* and honored by the external scheduler in order to take effect. */
/* They are not automatically enforced by LoadLeveler when an */
/* external scheduler is used. */
/* */
/* In this example, the job will only be run on one machine */
/* with only one task so the machine list consists of only 1 machine */
/* (plus the terminating NULL entry) */
start_info->nodeList = (char**)malloc(2*sizeof(char *));
if (!start_info->nodeList)
{
    fprintf(stderr, "Out of memory.\n");
    return;
}

start_info->nodeList[0] = strdup(use_machine);
start_info->nodeList[1] = NULL;

/* Retrieve information from the job to populate the start_info */
/* structure */
/* In the interest of brevity, the success of the ll_get_data() */
/* is not tested. In a real application it should be */

/* The version number is set from the header that is included when */
/* the application using the API is compiled. This allows for */
/* checking that the application was compiled with a version of the */
/* API that is compatible with the version in the library when the */
/* application is run. */
start_info->version_num = LL_PROC_VERSION;

/* Get the first step of the job to start */
ll_get_data(job, LL_JobGetFirstStep, &step);
if (step == NULL)
{
    printf("No step to start\n");
    return;
}

/* In order to set the submitting host, cluster number and proc */
/* number in the start_info structure, we need to parse it out of */
/* the step id */

/* First get the submitting host and save it */
ll_get_data(job, LL_JobSubmitHost, &submit_host);
start_info->StepId.from_host = strdup(submit_host);
free(submit_host);

rc = ll_get_data(step, LL_StepID, &step_id);

/* The step id format is submit_host.jobno.stepno. Because the */
/* submit host is a dotted string of indeterminant length, the */
/* simplest way to detect where the job number starts is to retrieve */
/* the submit host from the job and skip forward its length in the */
/* step id. */

pChar = step_id+strlen(start_info->StepId.from_host)+1;
/* The next segment is the cluster or job number */
pChar = strtok(pChar, ".");
start_info->StepId.cluster = atoi(pChar);
/* The last token is the proc or step number */
pChar = strtok(NULL, ".");
start_info->StepId.proc = atoi(pChar);
free(step_id);

/* For each protocol (eg. MPI or LAPI) on each task, we need to */
/* specify which adapter to use, whether a window is being used */
/* (subsystem = "US") or not (subsystem="IP"). If a window is used, */
Additional examples

/* the window ID and window buffer size must be specified. */
/* */
/* */
/* The adapter usage entries for the protocols of a task must be */
/* sequential and the set of entries for tasks on the same node must */
/* be sequential. For example the twelve entries for a job where */
/* each task uses one window for MPI and one for LAPI with three */
/* tasks per node and running on two nodes would be laid out as: */
/* */
/* 1: MPI window for 1st task running on 1st node */
/* 2: LAPI window for 1st task running on 1st node */
/* 3: MPI window for 2nd task running on 1st node */
/* 4: LAPI window for 2nd task running on 1st node */
/* 5: MPI window for 3rd task running on 1st node */
/* 6: LAPI window for 3rd task running on 1st node */
/* 7: MPI window for 1st task running on 2nd node */
/* 8: LAPI window for 1st task running on 2nd node */
/* 9: MPI window for 2nd task running on 2nd node */
/* 10: LAPI window for 2nd task running on 2nd node */
/* 11: MPI window for 3rd task running on 2nd node */
/* 12: LAPI window for 3rd task running on 2nd node */
/* */
/* An improperly ordered adapter usage list may cause the job not to */
/* be started or, if started, incorrect execution of the job */
/* */
/* */
/* This example starts the job with two tasks on one machine, using */
/* one switch adapter window on each task. The protocol is forced */
/* to MPI and a fixed window size of 1M is used. An actual external */
/* scheduler application would check the steps requirements and its */
/* */
/* */
start_info->adapterUsageCount = 2;
start_info->adapterUsage =
   (LL_ADAPTER_USAGE *)malloc((start_info->adapterUsageCount)
        * sizeof(LL_ADAPTER_USAGE));

start_info->adapterUsage[0].dev_name = use_adapter_1;
start_info->adapterUsage[0].protocol = "MPI";
start_info->adapterUsage[0].subsystem = "US";
start_info->adapterUsage[0].wid = use_window_1;
start_info->adapterUsage[0].mem = 1048577;

start_info->adapterUsage[1].dev_name = use_adapter_2;
start_info->adapterUsage[1].protocol = "MPI";
start_info->adapterUsage[1].subsystem = "US";
start_info->adapterUsage[1].wid = use_window_2;
start_info->adapterUsage[1].mem = 1048577;

if ((rc = ll_start_job_ext(start_info)) != API_OK)
{
    printf("Error %d returned attempting to start Job Step %s.%d.%d on %s\n",
           rc,
           start_info->StepId.from_host,
           start_info->StepId.cluster,
           start_info->StepId.proc,
           start_info->nodeList[0]);
}
else
{
    printf("ll_start_job_ext() invoked to start job step: ",
           "%s.%d.%d on machine: %s\n\n",
           start_info->StepId.from_host,
           start_info->StepId.cluster,
           start_info->StepId.proc,
           start_info->nodeList[0]);
}
free(start_info->nodeList[0]);
free(start_info);

Finally, when the step and job element are no longer in use, ll_free_objs() and ll_deallocate() should be called on the query element.
Appendix B. 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 465 for more information.
  - **The central manager is not operating.** See "What happens if the central manager isn’t operating?" on page 466 for more information.
  - **Miscellaneous questions.** See "Other questions" on page 469 for more information.

- **"Helpful hints"** on page 470, which contains tips on running LoadLeveler, including some productivity aids.

- **"Getting help from IBM"** on page 474, 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 387 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:

<table>
<thead>
<tr>
<th>Why Your Job May Not Be Running:</th>
<th>Possible Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job requires specific machine, operating system, or other resource.</td>
<td>• Does the resource exist in the LoadLeveler cluster? If yes, wait until it becomes available. Check the GUI to compare the job requirements to the machine details, especially Arch, OpSys, and Class. Ensure that the spelling and capitalization matches.</td>
</tr>
<tr>
<td>Job requires specific job class</td>
<td>• Is the class defined in the administration file? Use <code>llclass</code> to determine this. If yes, Is there a machine in the cluster that supports that class? If yes, you need to wait until the machine becomes available to run your job.</td>
</tr>
<tr>
<td>The maximum number of jobs are already running on all the eligible machines</td>
<td>Wait until one of the machines finishes a job before scheduling your job.</td>
</tr>
</tbody>
</table>
## Troubleshooting

<table>
<thead>
<tr>
<th>Why Your Job May Not Be Running</th>
<th>Possible Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>The start expression evaluates to false.</td>
<td>Examine the configuration files (both LoadL_config and LoadL_config.local) to determine the START control function expression used by LoadLeveler to start a job. As a problem determination measure, set the START and SUSPEND values, as shown in this example: START: T SUSPEND: F</td>
</tr>
<tr>
<td>The priority of your job is lower than the priority of other jobs.</td>
<td>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 iprior command or the GUI.</td>
</tr>
<tr>
<td>The information the central manager has about machines and jobs may not be current.</td>
<td>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 LoadL_config file. The default polling frequency is five minutes.</td>
</tr>
<tr>
<td>You do not have the same user ID on all the machines in the cluster.</td>
<td>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 user id on one machine, your jobs will not be scheduled to that machine.</td>
</tr>
<tr>
<td>CtSec is enabled and the .rhosts file was not updated.</td>
<td>The .rhosts file should contain entries which specify all the host and user combinations allowed to submit jobs which will run as the local user. Refer to page 409 for more details.</td>
</tr>
</tbody>
</table>

---

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 3, “LoadLeveler command line interface,” on page 23 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:

<table>
<thead>
<tr>
<th>Why Your Job May Not Be Running</th>
<th>Possible Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>The minimum number of processors requested by your job is not available.</td>
<td>Sufficient resources must be available. Specifying a smaller number of processors may help if your job can run with fewer resources.</td>
</tr>
<tr>
<td>The pool in your requirements statement specifies a pool which is invalid or not available.</td>
<td>The specified pool must be valid and available.</td>
</tr>
<tr>
<td>The adapter specified in the requirements statement or the network statement identifies an adapter which is invalid or not available.</td>
<td>The specified adapter must be valid and available.</td>
</tr>
<tr>
<td>PVM3 is not installed</td>
<td>PVM3 must be installed on any machine you wish to use for pvm. The PVM3 system itself is not supplied with LoadLeveler.</td>
</tr>
<tr>
<td>You are already running a PVM3 job on one of the LoadLeveler machines.</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Troubleshooting

<table>
<thead>
<tr>
<th>Why Your Job May Not Be Running</th>
<th>Possible Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>The parallel_path keyword in your job command file is incorrect.</td>
<td>Use parallel_path 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.</td>
</tr>
<tr>
<td>The pvm_root keyword in the administration file is incorrect.</td>
<td>This keyword corresponds to the pvm ep keyword and is required to tell LoadLeveler where the pvm system is installed.</td>
</tr>
<tr>
<td>The file /tmp/pvmd.userid exists on some LoadLeveler machine but no PVM jobs are running.</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Gang scheduler checklist:** Before running the Gang Scheduler, verify that:

- MACHINE_AUTHENTICATE is set to TRUE
- PROCESS_TRACKING is set to TRUE
- Applications are compiled with the multi-threaded library

**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. 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:
Troubleshooting

- 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
```

Verify that another machine has set schedd_host = true and schedd_runs_here = true.

Why won’t my job run on a cluster with both AIX and Linux machines?

The default shell on Linux (in both Red Hat Enterprise Linux and SUSE Linux Enterprise Server) is bash and bash may not be available on AIX. If a job step contains a bash script it will be rejected if it is run on an AIX node. The ksh is available on both AIX and Linux. You can specify which shell to use in the keyword shell in your job command file:

```
# @shell = /bin/ksh
```

Also, AIX and Linux are not binary compatible so jobs written in compiled languages such as C or Fortran must be compiled for the environment they will run on.

Why does a job stay in the pending (or starting) state?

If a job appears to stay in the Pending or Starting state, it is possible the job is continually being dispatched and rejected. Check the setting of the MAX_JOB_REJECT keyword. If it is set to -1 the job will be rejected an unlimited number of times. Try resetting this keyword to a small number, such as 10. Also,
check the setting of the ACTION_ON_MAX_REJECT keyword. These keywords are described in "Step 17: Specify additional configuration file keywords" on page 410.

### 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.

<table>
<thead>
<tr>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>The network goes down but the machines are still running</td>
<td>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.</td>
</tr>
<tr>
<td>The network partitions or goes down.</td>
<td>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.</td>
</tr>
<tr>
<td>The machine with startd goes down.</td>
<td>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 &quot;JOB VACATED&quot;, which causes the job to be restarted elsewhere. In the case that only LoadLeveler failed, it is often possible to &quot;find&quot; 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.</td>
</tr>
<tr>
<td>The central manager machine goes down.</td>
<td>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. 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.</td>
</tr>
</tbody>
</table>
### Troubleshooting

<table>
<thead>
<tr>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>The schedd machine goes down</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>During the time the schedd is down, the central manager will not be able to start new jobs that were submitted to that schedd.</td>
</tr>
<tr>
<td></td>
<td>To recover the resources allocated to jobs scheduled by a schedd machine, see &quot;How do I recover resources allocated by a schedd machine?&quot; on page 468.</td>
</tr>
<tr>
<td>The <code>llsubmit</code> machine goes down</td>
<td>schedd gets its own copy of the executable so it does not matter if the <code>llsubmit</code> machine goes down.</td>
</tr>
</tbody>
</table>

Why does `llstatus` 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 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:

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:

---

**Figure 49. When the primary central manager is unavailable**

**Figure 50. Multiple central managers**
Troubleshooting

- 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 344.

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.
3. Reconfigure the central manager node so that it recognizes the “fenced” schedd daemon. From the central manager machine issue `llctl reconfig`.
4. Issue `llctl -h host purgesched` 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.

Why can’t I find a core file on Linux?

On Linux, when a LoadLeveler daemon terminates abnormally a core file is not generated. Why? Although a LoadLeveler daemon begins its existence as a root process, it uses the system functions `seteuid()` and `setegid()` to switch to effective user ID of `loadl` and effective group ID of `loadl` immediately after startup if the file `/etc/LoadL.cfg` is not defined. If this file is defined, the user ID associated with the `LoadLUserId` keyword and the group ID associated with the `LoadLGroupId` keyword are used instead of the default `loadl` user and group IDs.

On Linux systems, unless the default kernel runtime behavior is modified, the standard kernel action for a process that has successfully invoked `seteuid()` and `setegid()` to have a different effective user ID and effective group ID is not to dump a core file. So, if you want Linux to create a core file when a LoadLeveler daemon terminates abnormally you must use the file `/etc/LoadL.cfg` to set both `LoadLUserId` and `LoadLGroupId` to `root`.  

468 Using and Administering LoadLeveler
On Red Hat Enterprise Linux 3.0 systems, the command `sysctl -w kernel.core_setuid_ok=1` can be used to change the default kernel core file creation behavior of setuid programs. If the `core_setuid_ok` option is enabled, the values of `LoadLUserid` and `LoadLGroupid` in the `/etc/LoadL.cfg` file do not have to be `root` for the successful creation of LoadLeveler core files.

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:

```c
#include <stdio.h>
main()
{
excl("/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.
Troubleshooting

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*POLLLING_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 POLLLING_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*POLLLING_FREQUENCY). This prevents the negotiator from prematurely marking a machine as “down” or prematurely cancelling jobs.

- In a large LoadLeveler cluster, issuing the llctl command with the -g can take minutes to complete. To speed this up, set up a working collective containing the machines in the cluster and use the dsh command; for example, dsh llctl reconfig. This command also allows you to limit your operation to a subset of machines by defining other working collectives.

Hints for running jobs

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

Real Time: 01:08:29
Job Step User Time: 00:30:15
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 350, are:
Troubleshooting

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:

```bash
# @ step_name = step1
# @ executable = ssba.job
# @ output = ssba.tmp
# @ ...
# @ queue
# @ step_name = append1
# @ dependency = (step1 != 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.
Historically, 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).
Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Intellectual Property Law
2455 South Road, P386
Poughkeepsie, New York 12601-5400
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly-available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which 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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.
Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

- AIX
- IBM
- LoadLeveler
- RISC System/6000
- RISC System/6000 Scalable POWERparallel Systems®

UNIX is a registered trademark of The Open Group in the United States and other countries.

LoadLeveler incorporates Condor, which was developed at the University of Wisconsin-Madison, and uses it with the permission of its authors.

Other company, product, and service names may be trademarks or service marks of others.
Glossary

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 Uniforum 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.

Cluster 1600. See IBM @server Cluster 1600.

cluster security services. A component of RSCT that is used by RSCT applications and other RSCT components to perform authentication within peer domains.

CtSec. Cluster security services.

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.

I

IBM @server Cluster 1600. An IBM @server Cluster 1600 is any PSSP or CSM-managed cluster comprised of POWER microprocessor based systems (including RS/6000 SMPs, RS/6000 SP nodes, and pSeries SMPs).

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.

MPICH. A portable implementation of the full Message-Passing Interface (MPI) standard. MPICH was developed by Argonne National Laboratory to be
highly portable and is used by a large number of
providers of MPI implementations.

**MPICH-GM.** A port of MPICH on top of Myrinet GM
code. Myrinet GM is a low level message-passing
system for Myrinet networks.

**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.

**O**

OSI. Operating System Instance. In the LoadLeveler
documentation, OSI and node are used interchangeably.

**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.

peer domain. A set of nodes configured for high
availability by the configuration resource manager.
Such a domain has no distinguished or master node.
All nodes are aware of all other nodes, and
administrative commands can be issued from any node
in the domain. All nodes also have a consistent view of
the domain membership.

**R**

Reliable Scalable Cluster Technology. A set of
software components that together provide a
comprehensive clustering environment for AIX. RSCT is
the infrastructure used by a variety of IBM products to
provide clusters with improved system availability,
scalability, and ease of use.

RSCT. See Reliable Scalable Cluster Technology.

RSCT peer domain. See peer domain.

**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.
Bibliography

This bibliography helps you find product documentation related to LoadLeveler. You can find IBM product information for most of these products on the World Wide Web. Formats for both viewing and downloading are available.

This bibliography also contains a list of non-IBM publications that discuss parallel computing and other topics related to LoadLeveler.

Information formats

Documentation supporting software licensed programs is no longer available from IBM in hardcopy format. However, you can view, search, and print documentation on the World Wide Web at:


or from the IBM @server Cluster Information Center at:

http://publib.boulder.ibm.com/clresctr/windows

Finding documentation on the World Wide Web

At the time this manual was published, the Web address of the "RS/6000 SP hardware and software documentation" page was:


You can also access most of the related hardware and software books at the IBM @server Cluster Information Center Web site:

http://publib.boulder.ibm.com/clresctr/windows/

You can view a book or download a Portable Document Format (PDF) version of it. To view the LoadLeveler 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:

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.

System planning publications

You can find links to the latest system planning publications on the Web site:


or from the IBM @server Cluster Information Center Web site:

http://publib.boulder.ibm.com/clresctr/windows
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.

You can find links to the RS/6000 SP Switch Router publications on the Web site:


Related hardware publications

For publications on the latest IBM @server pSeries and RS/6000 hardware products, see the Web site:

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
- IBM pSeries High Performance Switch guides

Related software publications

This section lists related software products. These products include:
- IBM Parallel System Support Programs for AIX (PSSP)
- IBM LoadLeveler for AIX (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)

You can find links to the latest documentation for these products on the Web site:


or from the IBM @server Cluster Information Center Web site:

http://publib.boulder.ibm.com/clresctr/windows

The latest HACMP publications are available at:


AIX publications

You can find links to the latest AIX publications on the Web site:

http://www.ibm.com/servers/aix/library
Reliable Scalable Cluster Technology (RSCT) publications

You can find links to the latest RSCT publications on the Web site:

http://www.ibm.com/servers/eserver/clusters/library

Cluster Systems Management (CSM) publications

You can find links to the latest CSM publications on the Web site:

http://www.ibm.com/servers/eserver/clusters/library

DCE publications

You can find links to the latest DCE publications on the Web site:


Redbooks

IBM’s International Technical Support Organization (ITSO) has published a number of Redbooks. For a current list, see the ITSO Web site:

http://www.ibm.com/redbooks

Non-IBM publications

Here are some non-IBM publications that you might find helpful.

• Foster, I., Designing and Building Parallel Programs, Addison-Wesley, 1995.
Index

Special characters
!var 103
/etc/LoadL.cfg file 49, 72
/etc/services file 390
.lrc script 147
$var 103

Numerics
64-bit command line interfaces lclclass -l 433
llq -l 434
llstatus -l 433
keywords supported administration file 432
configuration file 432
job command file 431
support for accounting functions 434
support for command line interfaces 433
support for command line interfaces and GUI 433
support for GUI 434
support for LoadLeveler APIs 434

A
account 350
account number, modify 339
account_no 97
accounting
API 239
collecting data 83
in job command file 97
llacctmrg command 158
llsummary command 228
reports 85
accounting functions
64-bit support 434
accounting, job setup 86, 415
ACCT 385
ACCT_VALIDATION 240, 385
ACTION_ON_MAX_REJECT 410
ACTION_ON_SWITCH_TABLE_ERROR 411
adapter
dedicated 111
shared 111
specifying in job command file 109, 114
adapter information
extracting from SDR 180
adapter stanza keywords 366
adapter_type 367
css_type 367
device_driver_name 367
interface_address 366
interface_name 366
logical_id 367
adapter stanza keywords (continued)
multilink_address 366
multilink_list 367
network_id 367
network_type 366
switch_node_number 367
adapter stanzas
examples 367
format 365
adapter_name 366
adapter_stanzas 345
adapter_type 366
adapter_type_keyword 367
admin 354, 364
admin keywords 364
class stanza 354
ADMIN_FILE 387
administering LoadLeveler
administering LoadLeveler 70
administration file 67, 69
customizing the administration file 344
LoadL_admin file 70
overview 67
Quick Set Up 69
stanzas 344
administration file
customizing 344
keywords 125
multiple statements 429
structure and syntax 70
administration file keyword details 426
administration file keyword summary 426
administration file keywords 425
administrative actions
GUI 33
administrators 69
AFS authentication 411
AFS authentication user exit 317
AFS token handling 317
AFS_GETNEWTOKEN 411
AGGREGATE_ADAPTERS keyword 372
alias 345
alternate central manager 385
API scheduler 370, 372
APIs 239
Gang scheduling 430
ll_cckpt 244
ll_init_cckpt 242
ll_set_cckpt_callbacks 246
ll_unset_cckpt_callbacks 246
APIs, LoadLeveler 39
application programming interfaces
accessing LoadLeveler objects 247
accounting 239
checkpointing serial jobs 242
job control 297
ll_error 285
querying jobs and machines 291
running parallel jobs 286
application programming interfaces (continued)
scheduling 297
submitting jobs 295
workload management 297
Application Programming Interfaces 39, 239
application support 422
Arch
requirement in job command file 115
variable 74
ARCH configuration file keyword 374
arguments 97
attributes (of job step) change 191
authentication process, DCE 317
authentication programs 316

B
backfill scheduler 371
BACKFILL scheduler 370
basic gui functions 27
basics, LoadLeveler 4
batch parallel jobs
naming files for checkpointing 394
BIN 387
blocking 56, 97
blocking factor 56
building jobs
using a job command file 45
using the GUI 325
building jobs using the GUI 28

C
canceled job state 153
cancelling jobs
using lcancel 440
using the GUI 339
capture data
GUI 36
central manager 7, 341, 345, 385, 440, 466
central_manager keyword 346
CENTRAL_MANAGER_HEARTBEAT_INTERVAL 386
CENTRAL_MANAGER_TIMEOUT 386
changing attributes of job steps
using Ilmmodify 191
changing job priority
example 440
using llprio 196
using the GUI 338
checklist
Gang scheduler 463
parallel jobs 463
checkpoint 98
file naming 395
limitations 395
removing old files 400

© Copyright IBM Corp. 2001, 2003, 2004
checkpoint (continued)
   restarting a job 464
checkpoint and restart limitations 395
checkpoint files, removing 400
checkpoint keywords 392
   administration file 392
   configuration file 393
   job command file 393
checkpoint, take 339
checkpointing
   API 242
   how to checkpoint a job 397
   naming files for interactive parallel jobs 394
   naming serial and batch files 394
   planning considerations 395
   system-initiated 98, 392
   user-initiated 98, 392
checkpointing a job step
   lckpt 162
checkpointing job state 153
   choice button 329
   circular preemption 428
ckpt (subroutine) 242
ckpt_dir 99, 354
ckpt_file 99
ckpt_time_limit 100, 354, 360
class
   job command file keyword 100
   multiple job classes 473
   querying class information 165
Class
   defining for a machine 374
   keyword 374
class stanza keywords
   admin 354
   ckpt_dir 354
   ckpt_time_limit 354
class_comment 354
class, modify 339
ClassSysprio 379
CLIENT_TIMEOUT 391
cluster
   definition 3
   querying multiple clusters 49
   submitting jobs to multiple clusters 49
   cluster with both AIX and Linux machines 391
   troubleshooting 464
CM_COLLECTOR_PORT 391
coexistence
   LoadLeveler 3.1 and 3.2 xxi
   mixed cluster xxi
   POE in a mixed cluster xxi
   requirements, restrictions, and operating characteristics xxi
   collect account data
   GUI 36
   COLLECTOR_DGRAM_PORT 391
   command 157
   command line interface 23
   64-bit support 433
   llclass -l 433
   llq -l 434
   llistatus -l 433
commands
   Gang scheduling 429
   llacctmrg 158
   llcancel 160
   lckpt 162
   llclass 165
   llcll 169
   lldecgymaint 174
   llextendRPD 177
   llextendSRD 180
   llfavorjob 184
   llfavoruser 186
   llhold 187
   llinit 189
   llmodify 191
   llpreempt 194
   llprio 196
   llq 198
   llstatus 216
   llsubmit 226
   llsummary 228
   summary 24
   command 100
   common name space 67
   communication level 109
   complete pending job state 153
   completed job state 153
configuration file
   customizing 72, 368
   keywords 130
   multiple statements 429
   structure and syntax 73
configuration file keyword details 422
configuration file keyword summary 422
configuration file keywords 422
configuration files
   global and local 72
configuration tasks
   GUI 34
configuration wizard
   llg 34
configuring
   Cluster Security Services 407
   DCE 401
   security service 401
configuring LoadLeveler
   global configuration file 72
   introduction 71
   LoadLeveler user ID 72
   local configuration file 72
Connectivity
   requirement in job command file 115
   using with MACHPRIO 380
   variable 74
considerations
   checkpointing 395
   Gang scheduling xxi
   parallel jobs 79
   POE 79
   POE in a mixed cluster xxi
   PVM 80
consumable CPUs, modify 339
consumable memory, modify 339
consumable resources
   introduction 14
   job scheduling 14
   Workload Manager 15
Consumable Resources 376
   when submitting and managing jobs 54
ConsumableCpus
   variable 74
ConsumableMemory
   variable 74
ConsumableVirtualMemory
   variable 75
CONTINUE expression 383
control functions 383
copy 360
COPY_ALL 103
core file on Linux
   troubleshooting 468
core_limit 101, 354, 360
cpu_limit 101, 354, 360, 361
cpu_speed_scale 345, 346, 417
Cpus
   using with MACHPRIO 380
   variable 75
CPUs, modify consumable 339
create account report
   GUI 36
css_type 366, 367
CtSec 407
CurrentTime 75
CUSTOM_METRIC 369
CUSTOM_METRIC_COMMAND 369
customizing
   administration file 344
   configuration file 72, 73
   global and local configuration file 368
CustomMetric 75, 381
GUI GroupTotalJobs GroupSysprio GroupQueuedJobs group

**Graphical User Interface**

- Basic functions: 27
- Building jobs: 28
- Customizing: 25, 28
- Overview: 25
- Starting: 25
- Tasks: 325
- Xloadl: 28
- Xloadl.so: 28

**Group Stanza Keywords**

- Admin: 364
- Exclude_users: 364
- Include_users: 364
- Max_node: 364
- Max_processors: 365
- Max_total_tasks: 365
- Maxidle: 364
- Maxjobs: 364
- Maxqueued: 364
- Priority: 365
- Total_tasks: 365

**Group Stanzas**

- Examples: 365
- Format: 363
- GroupQueuedJobs: 379
- GroupRunningJobs: 379
- GroupSysprio: 379
- GroupTotalJobs: 379
- Gsmonitor daemon: 152

**GUI**

- 64-bit support: 433, 434
- Building jobs: 28
- Configuration tasks: 34
- Customizing: 25, 25
- Machine administration: 33
- Capture data: 36
- Collect account data: 36
- Configure tasks: 34
- Create account report: 36
- Drain: 35
- Flush: 35
- Purge: 35
- Reconfig: 34
- Recycle: 34
- Resume: 35
- Start all: 34
- Start Drained: 34
- Start LoadLeveler: 34
- Stop all: 34
- Stop LoadLeveler: 34
- Version: 37

**Overview**

- Starting: 25
- Tasks: 325
- Xloadl: 28
- Xloadl.so: 28

**GUI (see graphical user interface): 31**

**H**

- Hardware, supported: 422
- Help: 474
- Calling IBM: 474
- In the GUI: 27

**History**

- JOB 387
- HISTORY: 387
- History file: 474
- Troubleshooting: 474
- HISTORY_PERMISSION: 385, 411
- Hold: 105
- Holding jobs: 50, 440
- Using the GUI: 338
- Host: 75
- Hostname: 75
- How to checkpoint a job: 397

**Idle**

- Idle job: 153
- Idle-like job: 192
- Image_size: 105
- Implied START_CLASS values: 428
- Include_groups: 354, 355
- Include_users: 354, 355, 364
- Initialdir: 106
- Initiators: 378
- Input: 106
- Instances: 58
- Integer blocking: 56
- Integrating LoadLeveler with WLM: 77
- Interactions: 428
- Gang scheduling: 428
- Circular preemption: 428
- Implied values: 428
- Job command file: 429
- Last one wins rule: 429
- Reconfiguration: 428
- Restrictions and preemption: 428
- Interactive jobs: 79
- Planning considerations: 79
- Interactive parallel jobs: 394
- Naming files: 394
- Interface_address: 366
- Interface_address_keyword: 366
- Interface_name: 366
- Interface_name_keyword: 366
- Interfaces, Application Programming: 39, 239

**J**

- Job accounting: 83
- Batch: 6
- Building a job command file: 45, 325
- Building using the GUI: 28
- Cancelling: 50, 339
- Change step attributes: 191
- Class name: 100
- Definition: 5
- Diagnosing problems with: 461, 462, 464
- Editing: 49, 335
- Environment variables: 48
- Exit status: 113, 226
- Filter: 318
- Holding: 50, 338
- Interactive: 79
- Parallel: 55, 462
- Preempt a step: 194
- Priority: 51, 196, 338, 352, 357, 365
- Releasing a hold: 339
- Running: 470
- Samples: 459
- Serial: 45
- States: 13
- Status: 49, 198, 201, 336
- Submit-only: 464
- Submitting: 45, 48, 336
- Job accounting setup procedure: 86, 415
- Job command file: 45
- Building: 45
- Environment keyword: COPY_ALL: 103
- Environment keywords: !var: 103
- $var: 103
- Var=value: 103
- Example: 46, 441, 442
- Executable example: 443
- Keywords: 97
- Account_no: 97
- Arguments: 97
- Blocking: 97
- Checkpoint: 98
- Ckpt_dir: 99
- Ckpt_file: 99
- Ckpt_time_limit: 100
- Class: 100
- Comment: 100
- Core_limit: 101
- Cpu_limit: 101
- Data_limit: 101
- Dependency: 102
- Environment: 103
- Error: 104
- Executable: 104
- File_limit: 104
- Group: 105
- Hold: 105
- Image_size: 105
- Initialdir: 106
- Input: 106
- Job_cpu_limit: 106
- Job_name: 107
- Job_type: 107
- Large_page: 107
- Max_processors: 108
- Min_processors: 108
- Network: 109
- Node: 111
- Node_usage: 112
- Notification: 112
- Notify_user: 113
- Output: 113
- Parallel_path: 113
- Preferences: 114
- Queue: 114
- Requirements: 114
- Resources: 117
- Restart: 118
- Restart_from_ckpt: 119
- Restart_on_same_nodes: 119
- Rss_limit: 119
- Shell: 120

---

**continued**
LoadLeveler variables (continued)
  Machine 75
  MasterMachPriority 76
  Memory 75
  OpSys 76
  PagesFreed 76
  PagesScanned 76
  QDate 76
  Speed 76
  state 76
  tilda 76
  UserPri 76
  VirtualMemory 76
local configuration file
configuring 72
customizing the 368
LOCAL_CONFIG 387
LOG 387
log files 387
  GMSMONITOR_LOG 388
  KBDD_LOG 388
  MASTER_LOG 388
  MAX_KBDD_LOG 388
  MAX_NEGOTIATOR_LOG 388
  MAX_STARTER_LOG 388
  NEGOIATOR_LOG 388
  SCHEDD_LOG 388
  STARTD_LOG 388
  STARTER_LOG 388
  logical_id 366
  logical_id keyword 367
machine
administrative actions 33
GUI 33
public scheduling 348
scheduling 7
Machine
requirement in job command file 116
variable 75
machine stanza keywords (continued)
  speed 349
  submit_only 349
machine stanzas
  examples 349
  format 344
machine status 216
MACHINE_AUTHENTICATE 369
machine_mode 345, 347
MACHINE_UPDATE_INTERVAL 411, 470
MACHPRIO 380
Mail keyword 319
mail program 318
MALLOC_TYPE 70, 171, 299
master daemon 147
master node 82
MASTER_DGRAM_PORT 391
master_node_exclusive 345, 347
master_node_requirement 354, 356
MASTER_STREAM_PORT 391
MasterMachPriority 381
variable 76
MAX_CKPT_INTERVAL 398
MAX_JOB_REJECT 411
max_jobs_scheduled 345, 347
max_node 350, 351, 354, 356, 364
max_processors 108, 350, 351, 352, 354,
  356, 364, 365
max_protocol_instances 61, 356
max_smp_tasks 426
MAX_STARTERS 375, 378
limits set by 378
max_total_tasks 350, 352, 354, 356, 364,
  365, 426
maxidle 350, 351, 364, 471
maxjobs 350, 351, 354, 356, 364, 471
maxqueued 350, 351, 364, 471
Memory
  requirement in job command file 116
  using with MACHPRIO 381
  variable 75
memory, modify consumable 339
menu bar 25
messages 343
migration considerations xx1
MIN_CKPT_INTERVAL 398
min_processors 108
mixed cluster
  coexistence xx1
  requirements, restrictions, and operating characteristics xx1
using POE xx1
modify consumable CPUs, consumable
  memory, class, and account
  number 339
monitor_program 296
monitoring programs 296
monitoring, file system 372
Moving from 3.1 to 3.2 xx1
MPI 109
MPICH
  job command file 450
  running jobs 63
MPICH-GM
  job command file 452
  running jobs 63
multilink_address 366
multilink_address keyword 366
multilink_list 366
multilink_list keyword 367
multiple statements
  administration file 429
  configuration file 429

N
name_server 345, 347
naming
  checkpoint files 395
  checkpointing files and directories 393
  checkpointing files for interactive parallel jobs 394
  checkpointing files serial and batch 394
  naming for checkpointing 393
  negotiator daemon 151
  job states 13
  keywords 411
NEGOTIATOR_CYCLE_DELAY 411
NEGOTIATOR_CYCLE_TIME_LIMIT 412
NEGOTIATOR_INTERVAL 411, 470
NEGOTIATOR_LOADAVG_INCREMENT 411, 412
NEGOTIATOR_PARALLEL_DEFER 412
NEGOTIATOR_PARALLEL_HOLD 412
NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL 412
NEGOTIATOR_REJECT_DEFER 412
NEGOTIATOR_REMOVE_COMPLETED 413
NEGOTIATOR_RESCAN_QUEUE 413
NEGOTIATOR_STREAM_PORT 391
network 109
network_id 366
network_type 367
network_type_keyword 367
network_type_keyword 366
nice 354, 356
node keyword 56, 111
node_usage 112, 429
not run job state 153
notification 112
notify_user 113
NotQueued job state 153
NQS
  options 418
  routing jobs to NQS machines 48, 87
  scripts 88
  jobs
    cancelling 420
    obtaining status 420
    submitting 418
NQS machine job routing procedure 89,
  417
NQS scripts 420
NQS_class 88, 354, 357
NQS_DIR 88, 387
NQS_query 88, 354, 357
NQS_submit 88, 354, 357
priority (of jobs) keyword in class stanza 357
keyword in group stanza 365
keyword in user stanza 352
system priority 51
user priority 51, 196, 352
procedure
job accounting setup 86, 415
NQS machine job routing 89, 417
process
starter 151
PROCESS_TRACKING 400
PROCESS_TRACKING_EXTENSION 400
productivity aids 470
Programming Interfaces, Application 39, 239
prolog programs 319
public scheduling machine 348
public scheduling machines 7, 52, 440
PUBLISH_OBITUARIES 413
pull-down menus 26
creating 29
purge
GUI 35
PVM 109
job command file 447
planning considerations 80
restrictions 81
pvm_root 345, 347

QDate 76, 379
query a job
llq command 198
using the GUI 337
query API 291
querying API 291
querying class information
llclass command 165
querying multiple clusters 49
questions and answers 461
queue 114
queue, see job queue 7

R
reconfig
GUI 34
reconfiguration 428
recycle
GUI 34
reject pending job state 154
REJECT_ON_RESTRICTED_LOGIN 413
rejected job state 154
release from hold 32
RELEASED_DIR 387
remove pending job state 154
removed job state 154
requirements 114
requirements, coexistence xxi
resources 345, 347
job command file keyword 117
resources, consumable
job scheduling 14
Workload Manager 15
restart 118
restart (continued)
limitations 395
restarting a checkpointed job 464
restart_from_cpt 119
restart_on_same_nodes 119
RESTARTS_PER_HOUR 413
restrictions
checkpoint and restart 395
checkpointing 395
Gang scheduling and
preemption 428
PVM 81
restrictions, coexistence xxi
resume
GUI 35
resume pending job state 154
RESUME_ON_SWITCH_TABLE_ERROR_CLEAR 414
rlim infinity 360
routing jobs, NQS machines 89, 417
RSCT peer domain
extracting data from 177
rs_limit 119, 354, 361
rule, last one wins 429
running job state 154
running jobs at a specific time of
day 471
running-like job states 193

SAVELOGS keyword 390
scaling considerations 470
schedd
troubleshooting 474
schedd daemon 148, 465
recovery 468
schedd_fenced 345, 348
schedd_host 345, 348, 470
SCHEDD_INTERVAL 414
SCHEDD_RUNS_HERE 376
SCHEDD_STATUS_PORT 391
SCHEDD_STREAM_PORT 391
SCHEDD_SUBMIT_AFFINITY 376, 470
SCHEDULE_BY_RESOURCES 376
scheduler
keyword definitions 370
option details 370
SCHEDULER_API 370
SCHEDULER_TYPE 370
schedulers
API 297, 370
Backfill 370
choosing 370
Default 370
external 297, 370
Gang 370
supported keywords 55
scheduling
Gang
administration file keywords 425
APIs 430
circular preemption 428
collection file keywords 422
implied START_CLASS
values 428
Job command file 429
reconfiguration 428
scheduling (continued)
  Gang (continued)
    restrictions 428
  Gang keywords 422
  parallel jobs 79
scheduling interactions, Gang 428
  scheduling machine 7
  public 348
  scheduling, job
    consumable resources 14
    script not executing
    troubleshooting 469
SDR
  extracting information from 180
  security
    configuring Cluster Security Services 407
    configuring DCE 401
  security credentials
    DCE 315
  security service
    configuring 401
serial checkpointing
  ckt subroutine 242
  serial job command files 46
  serial jobs
    naming files for checkpointing 394
    service numbers 390
    service_class 109
    shell 120, 329
    short report, accounting 86
    signals 287
    spacct_excluse_enable 345, 349
    speed 345, 349, 417
    Speed 76, 380
    SPOOL log 387
    stack_limit 120, 354, 361
stanzas
  adapter 365
  class 353
  default 71
  label 71
  machine 344
  type 71
  user 344
  start all
    GUI 34
  start Drained
    GUI 34
  START expression 383
  start failure
    MALLOCTYPE 70, 171, 299
start LoadLeveler
  GUI 34
  START_CLASS 424
  implied values 428
  START_DAEMONS 376
  startd daemon 149, 470
  STARTD_RUNS_HERE 376
  STARTD_STREAM_PORT 391
  startddate 120
  starter process 151
  starting job state 154
  state 76
  states, job 13, 153
status 216, 226
status, obtaining
  parallel jobs 63
  step_name 121
stop all
  GUI 34
stop LoadLeveler
  GUI 34
striping
  definition of 58
  submitting jobs 58
Striping
  examples of requesting striping in network statement 61
  Understanding over a single network 61
  Understanding over multiple networks 60
structure
  administration file 70
  SUBMIT_FILTER 318
  submit_only keyword 345, 349
  submit-only machine
    cancelling jobs from 50
    definition 3
    keywords 349
    master daemon interaction 147
    querying jobs from 49
    querying multiple clusters 49
    schedd daemon interaction 148
    submitting jobs from 48
    troubleshooting 464
    types 7
submitting jobs
  across multiple clusters 49
  using a job command file 48
  using an API 295
  using llsubmit 439
  using llsubmit command 226
  using the GUI 336
subroutines
  ckt 242
  GetHistory 240
  il.control 298
  il.deallocate 277
  il.free_jobs 292
  il.free_nodes 294
  il.free_objs 277
  il.get_data 257
  il.get_hostlist 287
  il.get_jobs 291
  il.get_nodes 293
  il.get_objs 252
  il.modify 302
  il.next_obj 276
  il.preempt 305
  il.query 248
  il.reset_request 251
  il.set_request 249
  il.start_host 289
  il.start_job 306
  il.start_job.ex 308
  il.terminal_job 312
  llacctval (user exit) 239
  llfree_job_info 296
  llsubmit 295
summary
  commands 24
support services 474
  support, 64-bit keywords 431, 432
  support, application 422
  SUSPEND expression 383
  switch_node_number 366
  switch_node_number keyword 367
syntax
  administration file 70
  sys/wait.h 322
syshold 32
SYSPRIO 51, 378
system and user hold job state 155
system hold job state 154
system priority 51
system-initiated checkpointing 98, 392

T
  take checkpoint 339
  task assignment 56
  task_geometry 56, 121
  tasks_per_node 121
  tasks_per_node keyword 56
TCP/IP service and port numbers 390
terminated job state 154
tilde 76
tm_hour 76
tm_isdst 76
tm_mday 77
tm_min 76
tm_mon 77
tm_sec 76
tm_wday 77
tm_yday 77
tm_year 77
tm4_year 77
total_tasks 122, 350, 352, 354, 357, 364, 365
total_tasks keyword 56
TotalMemory
  requirement in job command file 116
trademarks 477
troubleshooting 461
  .login script not executing 469
  .profile script not executing 469
  central manager isn’t operating 466
  checkpointed job won’t restart 464
  core file on Linux 468
  Gang scheduler checklist 463
  history file and schedd 474
  job stays in pending or starting state 464
  job won’t run 461
  job won’t run on cluster with both AIX and Linux machines 464
  llstatus does not agree with liq 466
  mksysb created when running jobs 469
  parallel job won’t run 462
  PVM 463
  recovering resources 468
  running jobs when a machine goes down 465
  set up problems with parallel jobs 463
  setuid = 0 469
  submit-only job won’t run 464
U
Understanding striping over a single network  61
Understanding striping over multiple networks  60
unfavor jobs  32
unfavor users  32
UNIX group  350
unlimited blocking  56, 97
UPDATE_ON_POLL_INTERVAL_ONLY  414
user exit
  llacctval  239
user exits  315
    monitoring programs  296
user hold job state  155
user name  67
user priority  51
user stanza keywords  350
  account  350
  default_class  350
  default_group  350
  default_interactive_class  351
    max_node  351
    max_processors  351, 352
    max_total_tasks  352
    maxidle  351
    maxjobs  351
    maxqueued  351
    total_tasks  352
user stanzas
  examples  352
    format  344
user_priority  123
user-defined variables  73
user-initiated checkpointing  98, 392
UserPrio  76, 379
UserQueuedJobs  379
UserRunningJobs  379
UserSysprio  379
UserTotalJobs  379

V
VACATE expression  383
vacate pending job state  155
vacated job state  155
var=value  103
variables
  configuration file
    user-defined  73
  LoadLeveler  123
  user-defined  73, 74
version
  GUI  37
VirtualMemory
    using with MACHPRIO  381
    variable  76
Reader’s comments – We’d like to hear from you

IBM LoadLeveler for AIX 5L and Linux
Using and Administering
Version 3 Release 2

Publication No. SA22-7881-02

Overall, how satisfied are you with the information in this book?

<table>
<thead>
<tr>
<th></th>
<th>Very Satisfied</th>
<th>Satisfied</th>
<th>Neutral</th>
<th>Dissatisfied</th>
<th>Very Dissatisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall satisfaction</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>

How satisfied are you that the information in this book is:

<table>
<thead>
<tr>
<th></th>
<th>Very Satisfied</th>
<th>Satisfied</th>
<th>Neutral</th>
<th>Dissatisfied</th>
<th>Very Dissatisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accurate</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Complete</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Easy to find</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Easy to understand</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Well organized</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Applicable to your tasks</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>

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. ________________________________