Using and Administering LoadLeveler Release 3.0

Document Number SC23-3989-00

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page vi.

Fourth Edition, August 1996

This edition applies to Version 1 Release 3 of IBM LoadLeveler, (5765-145 for RISC System/6000 and RISC System/6000 Scalable POWERparallel Systems), (5765-227 for Sun), (5765-228 for SGI), (5765-287 for HP). This edition applies to subsequent releases and modifications until otherwise indicated in new editions or Technical Newsletters.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

IBM welcomes your comments. A form for your comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, Department 55JA, Mail Station P384
522 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX: (International Access Code)+1+914+432-9405

IBMLink (United States customers only): KGNRCF at KGNVMC
IBM Mail Exchange: USIB27BQ at IBMMAIL
Internet: kgnrcf@vnet.ibm.com

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

Copyright International Business Machines Corporation 1993, 1994, 1995, 1996. All rights reserved. Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.. Copyright 1986, 1987, 1988, 1989, 1990, 1991 by the Condor Design Team.

Contents

Notices . . . vi
Trademarks and Service Marks . . . vi
Who Should Use This Book . . . vii
How this Book is Organized . . . vii
Typographic Conventions . . . viii
Related Publications . . . viii
LoadLeveler Man Pages . . . viii
Accessing This Book in Softcopy Format . . . ix
World Wide Web . . . ix
InfoExplorer Format . . . ix
PostScript Format . . . ix
What's New in Release 3? . . . x
New Application Programming Interfaces (APIs) . . . x
Enhanced Interactive Session Support (ISS) . . . x
Enhanced GUI . . . x
New and Enhanced Keywords . . . xi
Improved Daemons . . . xi
Enhanced Commands . . . xi
Improved Documentation . . . xii
Migration Considerations . . . xii

Overview of LoadLeveler . . . 1

Chapter 1. What is LoadLeveler? . . . 3
How LoadLeveler Works . . . 4

Chapter 2. LoadLeveler Daemons and Job States . . . 13
Daemons and Processes . . . 13
LoadLeveler Job States . . . 18

Using LoadLeveler . . . 21

Chapter 3. Submitting and Managing Jobs . . . 23
Building a Job Command File . . . 23
Submitting a Job Command File . . . 26
Managing Jobs . . . 27
A Simple Task Scenario Using Commands . . . 32
Additional Job Command File Examples . . . 35
Job Command File Keywords . . . 40

Chapter 4. Submitting and Managing Parallel Jobs . . . 55
Supported Parallel Environments . . . 55
Setting Up to Allow Users to Submit Parallel Jobs . . . 55
Restrictions and Limitations for Submitting Parallel Jobs . . . 58
Submitting Parallel Jobs Using llsubmit . . . 58
Migration Issues for Submitting Parallel Jobs . . . 68

Administering LoadLeveler . . . 71

© Copyright IBM Corp. 1993, 1996

Chapter 5. Administering and Configuring LoadLeveler 73
Planning Considerations 73
Quick Set Up 75
Administering LoadLeveler 77
Configuring LoadLeveler 96
Keyword Summary 127

Chapter 6. Using Interactive LoadLeveler 137
Changes from the Previous Release 137
Overview of ISS 137
Using Interactive Session Support 138
Troubleshooting 156
ISS Configuration File 160

Chapter 7. Gathering Job Accounting Data 163
Collecting Job Resource Data on Serial and Parallel Jobs 163
Collecting Job Resource Data Based on Machines 163
Collecting Job Resource Data Based on Events 164
Collecting Job Resource Information Based on User Accounts 164
Collecting the Accounting Information and Storing it into Files 165
Accounting Reports 165
Sample Job Accounting Scenario 166

Chapter 8. Error Logging Facility 169
Error Logging Overview 169

Chapter 9. Routing Jobs to NQS Machines 177
Setting Up the NQS Environment 177
Designating Machines to Which Jobs Will be Routed 178
Sample Routing Jobs to NQS Machines Scenario 178
NQS Scripts 181

Command Reference 183

Chapter 10. LoadLeveler Commands 185
Summary of LoadLeveler Commands 185
llacctmrg - Collect machine history files 186
llcancel - Cancel a Submitted Job 188
llclass - Query Class Information 190
llctl - Control LoadLeveler Daemons 193
llfavorjob - Reorder System Queue by Job 197
llfavoruser - Reorder System Queue by User 199
llhold - Hold or Release a Submitted Job 200
llinit - Initialize Machines in the LoadLeveler Cluster 203
llprio - Change the User Priority of Submitted Job Steps 205
llq - Query Job Status 207
llstatus - Query Machine Status 214
llsubmit - Submit a Job 219
llsummary - Return Job Resource Information for Accounting 221

The Graphical User Interface 227

Chapter 11. Graphical User Interface Overview 229
Starting the Graphical User Interface 229
Building and Submitting Jobs Using the Graphical User Interface 232
Customizing the Graphical User Interface 246
Administrative Uses for the Graphical User Interface 250

Application Programming Interfaces 255

Chapter 12. LoadLeveler Application Programming Interfaces 257
Accounting APIs 257
Prolog and Epilog APIs 259
Checkpointing API 265
The Submit API 266
Parallel Job API 268
Scheduling API 273
Query API 277
User Exits 280

Appendixes 283

Appendix A. Troubleshooting LoadLeveler 285
Frequently Asked Questions 285
Helpful Hints 293
Getting Help from IBM 297

Appendix B. Customer Case Studies 299
Customer 1: Circuit Simulation 299
Customer 2: High-Energy Physics 302
Customer 3: Computer Chip Design 304

Appendix C. Sample Files 313
Sample Administration File 313
Sample Configuration File 316

Glossary 323

Index 327

Notices

References in this publication to IBM* products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe on any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY
USA 10594

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Request

Trademarks and Service Marks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX
AIXwindows
IBM
LoadLeveler
NetView
RISC System/6000
RISC System/6000 Scalable POWERparallel Systems

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

LoadLeveler incorporates Condor, which was developed at the University of Wisconsin-Madison, and uses it with the permission of its authors.

Who Should Use This Book

This book replaces the two books provided with the previous release of LoadLeveler: the *LoadLeveler User's Guide* and the *LoadLeveler Administration Guide*. The material in those manuals has been reorganized, reworked, and combined into one book to increase the usability and retrievability of the information.

This manual is intended for those who are responsible for using and/or administering LoadLeveler.

Tasks involved with using LoadLeveler include submitting parallel, serial, and interactive jobs. Tasks involved with administering Loadleveler 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, used in the AIX* environment. Knowledge of networking and NFS** or AFS** protocols is also helpful.

How this Book is Organized

This books contains the following sections:

- "Overview of LoadLeveler" on page 1 describes what LoadLeveler is and how it works, and includes an explanation of the LoadLeveler daemons and processes.
- "Using LoadLeveler" on page 21 describes how to submit both serial and parallel jobs to LoadLeveler.
- "Administering LoadLeveler" on page 71 describes how to perform administration tasks, such as configuring LoadLeveler, setting up interactive login sessions, gathering accounting data, using the error logging facility, and routing jobs to NQS.
- "Command Reference" on page 183 describes the LoadLeveler commands.
- "The Graphical User Interface" on page 227 describes the LoadLeveler graphical user interface.
- "Application Programming Interfaces" on page 255 describes LoadLeveler's application programming interfaces.
- "Appendixes" on page 283 include the following: Appendix A, "Troubleshooting LoadLeveler" on page 285, Appendix B, "Customer Case Studies" on page 299, and Appendix C, "Sample Files" on page 313.

A glossary and index are also included.

Users of LoadLeveler should, at a minimum, become familiar with "Overview of LoadLeveler" on page 1 and "Using LoadLeveler" on page 21. Administrators should, at a minimum, become familiar with "Administering LoadLeveler" on page 71, and may find it helpful to read Appendix A, "Troubleshooting LoadLeveler" on page 285.

Typographic Conventions

This book uses the following typographic conventions:

Typographic	Usage
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**>.

Related Publications

- *LoadLeveler Memo To Users* and *LoadLeveler Installation Memo* (both shipped as hardcopy and also included on the product media as README files)
- *Implementing High Availability on the RISC/6000 SP*, SG24-4742

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 llcancel**.

The following man pages 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_negotiator**.

Man Page Name	What it Describes
LoadL_admin	The administration file
LoadL_checkpointing	The checkpointing facility
LoadL_config	The configuration file
LoadL_kbdd	The keyboard daemon
LoadL_master	The master daemon
LoadL_negotiator	The negotiator daemon
LoadL_schedd	The schedd daemon
LoadL_startd	The startd daemon
LoadL_starter	The starter process
LoadL_parallelapi	The parallel jobs API.
xloadl	The graphical user interface
xloadl_so	The graphical user interface for submit-only LoadLeveler

Accessing This Book in Softcopy Format

You can access this book in the following ways:

- On the World Wide Web
- In InfoExplorer format
- As a PostScript file

World Wide Web

You can view this book on the World Wide Web using the following URL:

http://www.rs6000.ibm.com

From here, select Software and then select Publications.

InfoExplorer Format

This book is available in InfoExplorer format for AIX customers only. You can obtain the InfoExplorer documentation by selecting the InfoExplorer documentation option during the **installp** process.

Then, you can access the InfoExplorer documentation by typing:

info -l *directory_path*

where *directory_path* is the directory in which the InfoExplorer files were loaded. This is usually called **/usr/lpp/info/data/LoadL**.

PostScript Format

This book is available in PostScript format as **lluadmin.ps**. You can obtain the PostScript version in two ways:

- By anonymous FTP (File Transfer Program) from the **lscftp.kgn.ibm.com** location. The PostScript files can be found in the directory called **/pub/pps/loadleveler**. In this directory, you will also find a file called **legal.readme** that contains more information regarding these PostScript files.

- By selecting the PostScript documentation option during the **installp** process, or by extracting the documentation files from the tar file.

Note: Some PostScript printers need to have carriage return characters stripped from the PostScript file before printing. If your printer needs this done, issue:

```
cat input_file.ps | tr -d '\r' output_file.ps
```

What's New in Release 3?

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

New Application Programming Interfaces (APIs)

- LoadLeveler now includes a scheduling API that allows you to disable the native LoadLeveler scheduler and "plug in" an external scheduler. For more information, see "Scheduling API" on page 273.
- LoadLeveler now includes a query API that allows you to collect information you can use in conjunction with the scheduling API. For more information, see "Query API" on page 277.

Enhanced Interactive Session Support (ISS)

ISS now includes the following functional enhancements:

- CONTINUOUS metrics
- Health checks: ISS performs server status checks between nameserver updates
- State sensitive operation on all types of metrics
- Load capping
- Multiple domains
- Alarm facility to alert you when a pool has no available servers
- Backup capabilities
- Elimination of the use of **rsh** for CUSTOM metrics

For more information, see Chapter 6, "Using Interactive LoadLeveler" on page 137.

Enhanced GUI

The graphical user interface (GUI) has been enhanced to allow you to:

- Customize the Jobs and Machines windows. This includes creating your own pulldown menus.
- Specify primary and secondary sorting on the Jobs window and the Machines window.
- Use UNIX regular expressions in some selection fields.
- Perform sorting and selecting actions against a LoadLeveler group.

For more information, see "The Graphical User Interface" on page 227.

New and Enhanced Keywords

Administration File Keywords

- **maxidle:** this new keyword allows you to control the number of idle jobs allowed for a user or a LoadLeveler group. For more information, see “Step 2: Specify User Stanzas” on page 83 or “Step 4: Specify Group Stanzas” on page 93.
- **wall_clock_limit:** this new keyword allows you to indicate the maximum elapsed time for which a job can run. For more information, see “Limit Keywords” on page 88.
- **maxjobs:** this keyword is now supported as part of a class stanza. For more information, see “Step 3: Specify Class Stanzas” on page 86.

Configuration File Keywords

- **NEGOTIATOR_REJECT_DEFER:** this new keyword specifies the number of seconds the negotiator waits before it considers scheduling a job to a machine that recently rejected the job. For more information, see “Additional Configuration File Keywords” on page 119.
- You can now use the following new keywords in the **SYSPRIO** (system priority) expression:
  - **UserQueuedJobs** specifies the number of jobs either running or queued for a user.
  - **UserTotalJobs** specifies the total number of jobs associated with a user.
  - **GroupQueuedJobs** specifies the number of jobs associated with a LoadLeveler group which are either running or queued.
  - **GroupTotalJobs** specifies the total number of jobs associated with a LoadLeveler group.

  For more information, see “Step 5: Prioritize the Queue Maintained by the Negotiator” on page 103.

Improved Daemons

Most LoadLeveler daemons now use a common C++ framework which allows them to reuse common code and to be *multi-threaded*. This improves the reliability and performance of the daemons. Also, the collector daemon and the negotiator daemon have been merged into one daemon (called the negotiator daemon). Most administration keywords associated with the collector are now ignored, although some are used by the negotiator. Data previously kept by the collector and retrieved every interval by the negotiator is now sent directly to the negotiator, resulting in the most current information being available for scheduling decisions.

Enhanced Commands

- The **llq** command now reports on jobs that are in the NotQueued state. Jobs in this state, unlike jobs in the Idle state, are not considered for scheduling. Also, **llq** now displays additional information as to why a job is not running. For more information, see “llq - Query Job Status” on page 207.
- The **llctl** command now allows you to drain a class of jobs on a startd machine. For more information, see “llctl - Control LoadLeveler Daemons” on page 193.

- The **llstatus** command now reports information on Available Classes (previously called Initiators) and Configured Classes (previously called Classes). For more information, see “llstatus - Query Machine Status” on page 214.

Improved Documentation

This book, *Using and Administering LoadLeveler*, replaces the the two manuals that existed prior to this release: *LoadLeveler User's Guide* and *LoadLeveler Administration Guide*. The existing material has been reorganized to increase the readability and usability of the information, and several new chapters have been added. For more information, see “How this Book is Organized” on page vii.

Migration Considerations

See the *LoadLeveler Installation Memo*, which is shipped as hardcopy with the product, for information on migration.

Overview of LoadLeveler

© Copyright IBM Corp. 1993, 1996

Chapter 1. What is LoadLeveler?

LoadLeveler is a job management system that allows users to run more jobs in less time by matching their processing needs to available resources. As part of its job management system, LoadLeveler serves as a job scheduler and provides a facility for building, submitting, and processing jobs quickly and efficiently in a dynamic environment.

Figure 1 shows the different environments to which LoadLeveler can schedule jobs. Together, these environments comprise the *LoadLeveler cluster*. An environment can include heterogeneous clusters, dedicated nodes, and the RISC System/6000 Scalable POWERparallel System (SP).

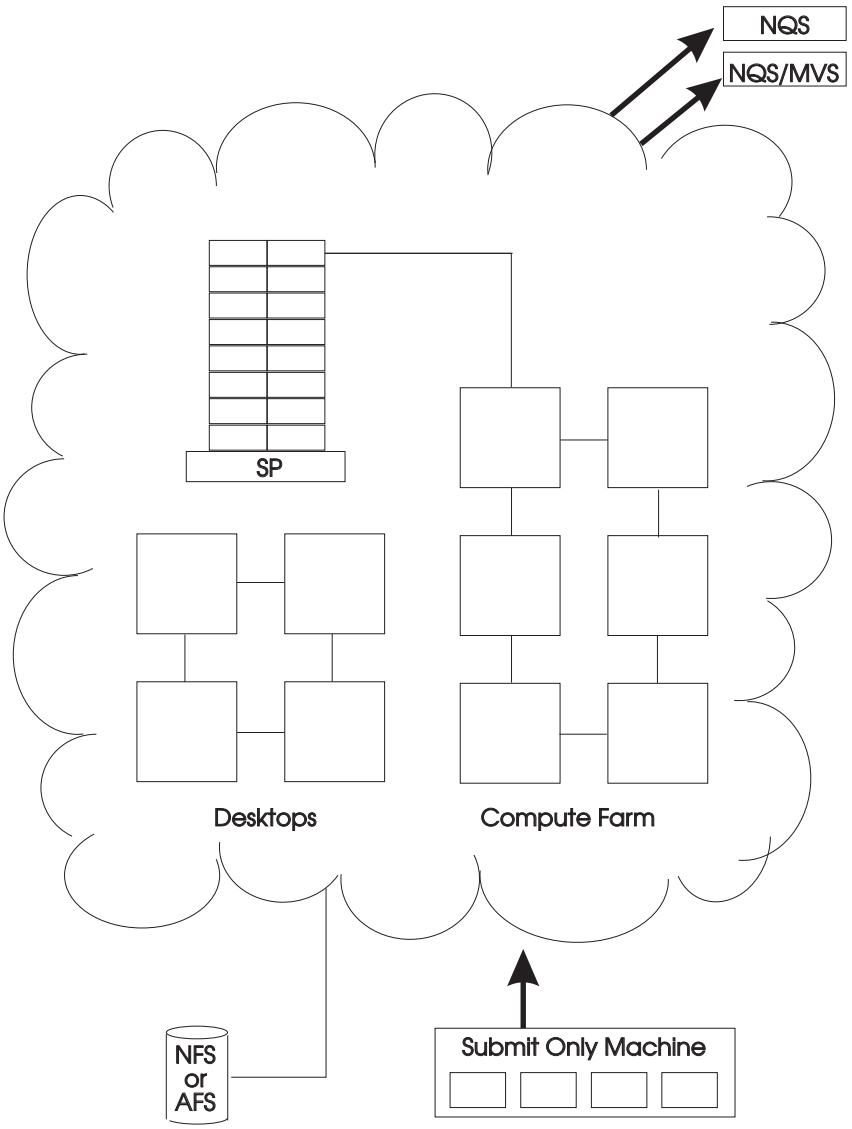


Figure 1. Example of a LoadLeveler Configuration

In addition, LoadLeveler can schedule jobs written for NQS to machines outside of the LoadLeveler cluster for execution. As Figure 1 also illustrates, a LoadLeveler cluster can include *submit-only* machines, which allow users to have access to a

© Copyright IBM Corp. 1993, 1996

limited number of LoadLeveler features. This type of machine is further discussed in “Roles of Machines” on page 5.

How LoadLeveler Works

This section describes how LoadLeveler works by introducing some basic job scheduling concepts.

What Does a Network Job Management and Job Scheduling System Do?

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.

Jobs

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

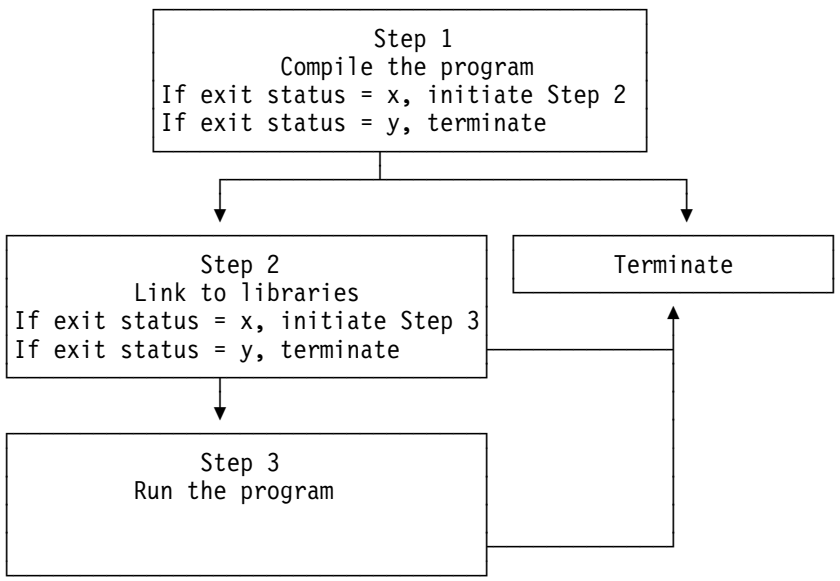


Figure 2. LoadLeveler Job Steps

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

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

You can submit batch jobs to LoadLeveler for scheduling. Batch jobs run in the background and generally do not require any input from the user. Batch jobs can either be *serial* or *parallel*. A serial job runs on a single machine, while a parallel job – a job that was written using a parallel language Application Program Interface (API) – is separated into multiple parts that can be processed simultaneously by several machines.

In addition to running batch jobs, you can also submit interactive jobs using LoadLeveler Interactive Session Support (ISS), which is described in Chapter 6, "Using Interactive LoadLeveler" on page 137.

Machines and Workstations

In order for LoadLeveler to schedule a job on a machine, the machine must be a valid member of the LoadLeveler cluster. A cluster is the combination of all of the different types of machines that use LoadLeveler. The following types of machines can be in a LoadLeveler cluster:

- RISC System/6000 (and compatible hardware running AIX)
- SP System
- Sun SPARCstation
- SGI IRIX System
- HP 9000 Series 700 System

Note that with an SP system, only one SP system or one SP system partition can be part of any given LoadLeveler cluster if you are using the Resource Manager.

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

Roles of Machines: Each machine in the LoadLeveler cluster performs one or more roles that make job scheduling possible. 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 keeps persistent information about the job. Some scheduling machines are known as *public scheduling machines*, meaning 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. This may be 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 will never be available to other LoadLeveler users.

LoadLeveler Daemons

This section lists the daemons that LoadLeveler uses to process jobs. For more detailed information, see "Daemons and Processes" on page 13.

LoadL_master Referred to as the **master** daemon, this daemon manages all LoadLeveler daemons on its machine. The master daemon runs on all machines in the cluster.

LoadL_schedd Referred to as the **schedd** daemon, this daemon manages a list of jobs submitted to the machine. The schedd daemon runs on all scheduling machines in the cluster.

LoadL_startd Referred to as the **startd** daemon, this daemon accepts jobs to be run on the machine where startd runs. The startd daemon runs on all executing machines in the cluster.

LoadL_starter Spawned by the startd daemon, the **starter** process manages a running job on the executing machine. The starter process runs on all executing machines in the cluster.

LoadL_kbdd Referred to as the **keyboard** daemon, this daemon monitors keyboard and mouse activity. The keyboard daemon runs on all executing machines in the cluster.

LoadL_negotiator Referred to as the **negotiator** daemon, this daemon collects job status and machine status from all machines in the LoadLeveler cluster, and makes decisions on where the jobs should be run. The negotiator daemon runs on the LoadLeveler central manager machine.

How Does LoadLeveler Schedule Jobs to Run on Machines?

Once a user submits a job to LoadLeveler, LoadLeveler examines the job in order to determine what resources it needs to run the job. Then, LoadLeveler determines which machines in the LoadLeveler cluster are best suited to run the job. Once the appropriate machine is found, LoadLeveler dispatches the job to the machines. To provide this function, LoadLeveler uses the concept of *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 enters into an internal database that resides on one of the machines in the LoadLeveler cluster until it is ready to be dispatched to another machine to be run, as shown in Figure 3.

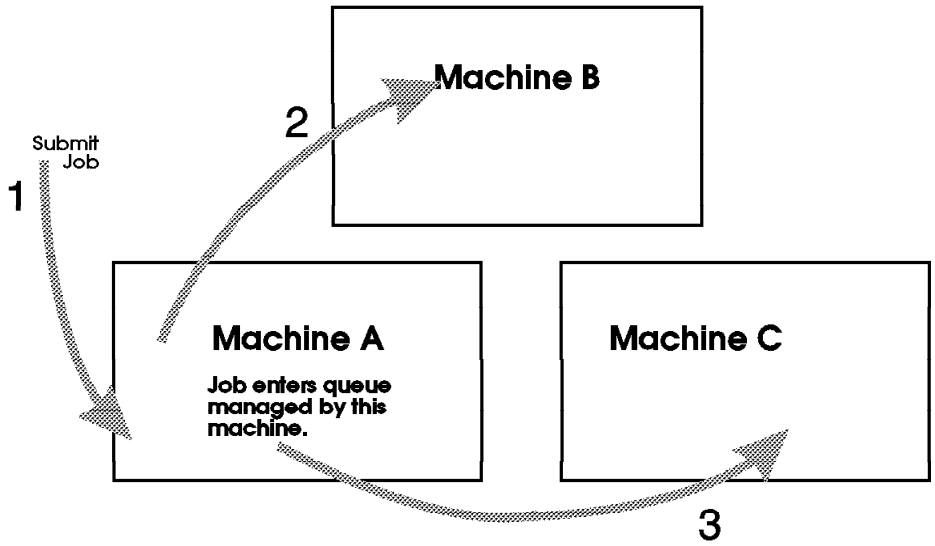


Figure 3. Job Queues

Once LoadLeveler examines the 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 based upon a first-come, first-serve basis. Instead, LoadLeveler examines the requirements and characteristics of the job and the availability of machines and determines the best time for the job to be dispatched.

LoadLeveler also uses the concept of *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, see “Step 3: Specify Class Stanzas” on page 86.

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 78.

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 29.

The LoadLeveler Job Cycle

Figure 4 illustrates the information flow through the LoadLeveler cluster:

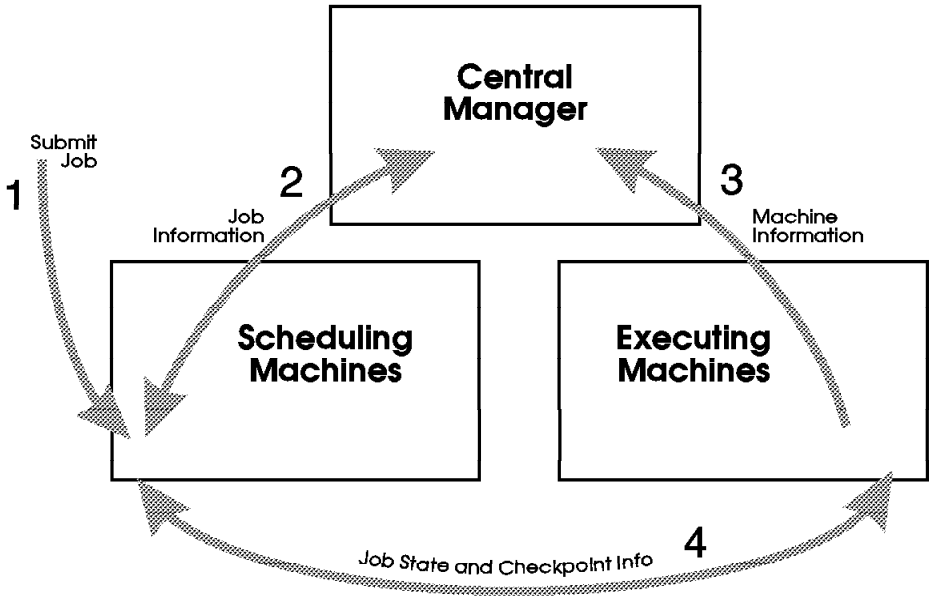


Figure 4. High-Level Job Flow

With LoadLeveler, there is a managing machine known as the **central manager**. Also, there are machines that act as scheduling machines and machines that serve as the executing machines. The arrows in Figure 4 illustrate the following:

- Arrow 1 indicates that a job has been submitted to LoadLeveler.
- Arrow 2 indicates that the scheduling machine contacts the central manager to inform it that a job has been submitted and to find out if a machine exists that matches the job requirements.
- Arrow 3 indicates that the central manager checks to determine if a machine exists that is capable of running the job. Once a machine is found, the central manager informs the scheduling machine which machine is available.
- Arrow 4 indicates that the scheduling machine contacts the executing machine and provides it with information regarding the job.

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

1 Submit a LoadLeveler job:

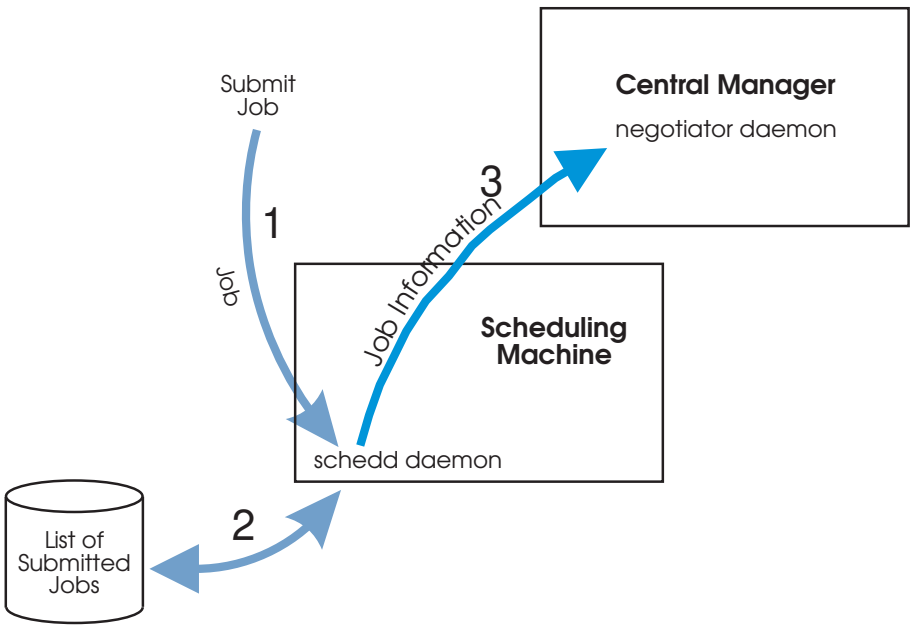


Figure 5. Job is Submitted to LoadLeveler

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:

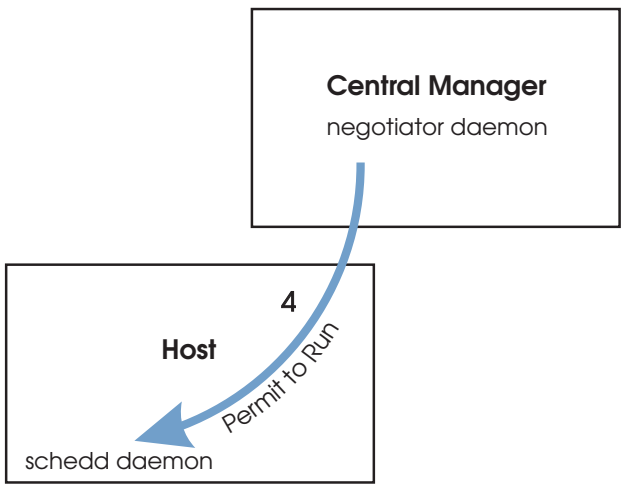


Figure 6. LoadLeveler Authorizes the Job

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 *permit to run*. Once this is done, the job is considered Pending or Starting. (See "LoadLeveler Job States" on page 18 for more information.)

3 Prepare to run:

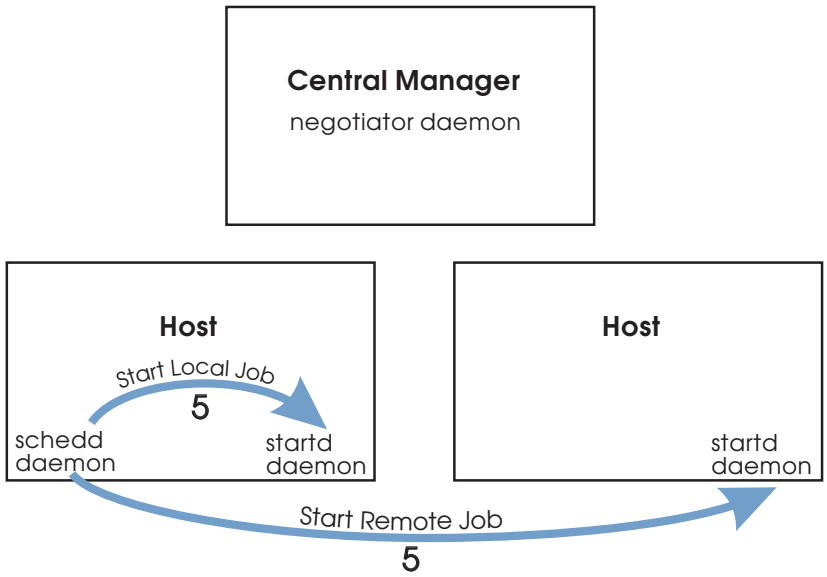


Figure 7. LoadLeveler Prepares to Run the Job

In Figure 7, arrow 5 illustrates that the schedd daemon contacts the startd daemon on the executing machine and requests that it start the job. The executing machine can either be a local machine (the machine from which the job was submitted) or a remote machine (another machine in the cluster).

4 Initiate job:

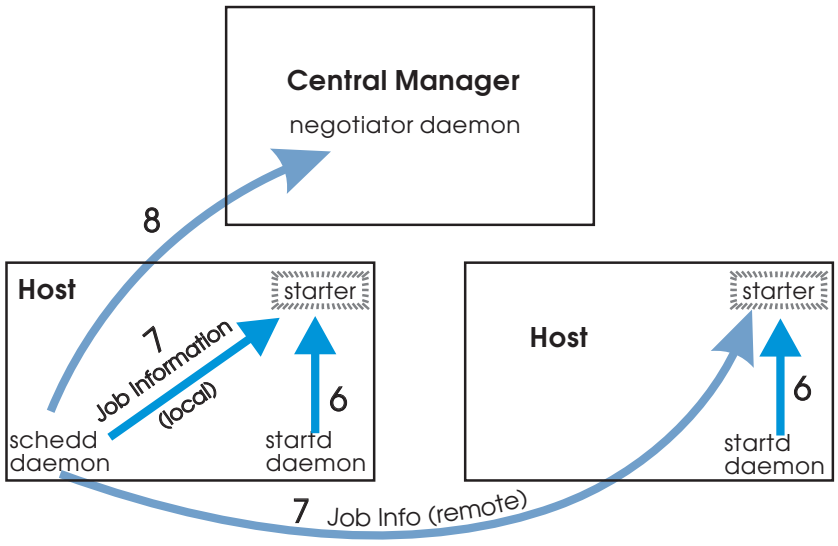


Figure 8. LoadLeveler Starts the Job

The arrows in Figure 8 illustrate the following:

- 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” on page 18 for more information.)

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

5 Complete job:

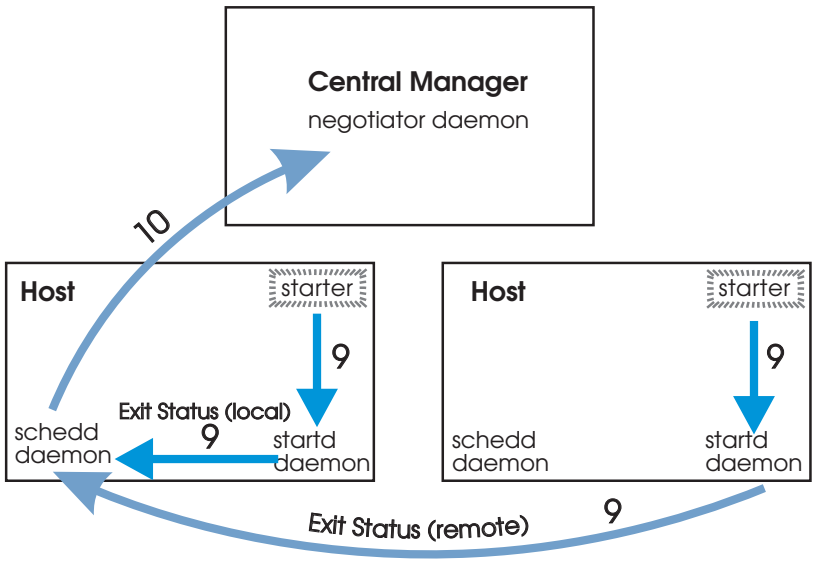


Figure 9. LoadLeveler Completes the Job

The arrows in Figure 9 illustrate the following:

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

Chapter 2. LoadLeveler Daemons and Job States

This chapter presents a detailed explanation of LoadLeveler daemons and processes. Included here is a description of job states, which are controlled by certain daemons. See “LoadLeveler Job States” on page 18 for more information.

Daemons and Processes

This section presents a detailed explanation of LoadLeveler daemons and processes.

The master Daemon

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

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

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

This daemon starts and if necessary, restarts all the LoadLeveler daemons that the machine it resides on is configured to run. As part of its startup procedure, this daemon executes the **.llrc** file (a dummy file is provided under the release /**bin** 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 aborts and all daemons are killed.

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

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

- Kill all daemons and exit
- Kill all daemons and execute a new master
- Re-run the **.llrc** file, reread the configuration files, stop or start daemons as appropriate for the new configuration files

© Copyright IBM Corp. 1993, 1996

- Send drain request to startd and schedd
- Send flush request to startd and send result to caller
- Send suspend request to startd and send result to caller
- Send resume request to startd and schedd, and send result to caller

The schedd Daemon

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

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

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

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

Draining The schedd daemon has been drained by the administrator but some jobs are still running. The state of the machine remains Draining until all running jobs complete. At that time, the machine status changes to Drained.

Available This machine is available to schedule jobs.

The schedd daemon performs the following functions:

- Assigns new job ids when requested by the **llsubmit** command.
- Receives new jobs from the **llsubmit** command. A new job is received as a series of descriptors or *procs* for each job step. A proc is the data structure in memory containing all the information about a job step. A null proc indicates the end of the series. The schedd forwards procs to the negotiator daemon as soon as they are 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 procs is 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.

- Requests that a remote startd daemon kill a job.
- Handles the checkpoint file associated with the job, provided checkpointing has been enabled. For more information, see “Step 13: Enable Checkpointing” on page 113.
- Receives accounting information from startd.

The startd Daemon

The **startd** daemon monitors jobs and machine resources on the local machine and forwards this information to the negotiator daemon. The startd also receives and executes job requests originating from remote machines. The master daemon starts, restarts, signals, and stops the startd daemon.

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

Busy	The maximum number of jobs are running on this machine.
Down	The daemon is not running on this machine. The startd daemon enters this state when it has not reported its status to the negotiator. This can occur when the machine is actually down, or because there is a network failure.
Drained	The startd machine accepts no more jobs; and jobs already running on the startd machine are allowed to complete.
Draining	The startd daemon has been drained by the administrator but some jobs are still running. The state of the machine remains Draining until all running jobs complete. At that time, the machine status changes to Drained.
Flush	All jobs on this machine have been flushed. No new jobs are accepted.
Idle	The machine is not running any jobs.
None	LoadLeveler is running on this machine, but no jobs can run here.
Reserved	The resource manager has this machine reserved for use by interactive jobs.
Running	The machine is running one or more jobs and is capable of running more.
Suspend	All jobs on this machine have been suspended by the administrator.

The startd deamon performs these functions:

- Runs a timeout procedure that includes building a snapshot of the state of the machine that includes static and dynamic data. This timeout procedure is run at the following times:
  - After a job completes.
  - According to the definition of the **POLLING_FREQUENCY** keyword in the configuration file.
- Determines the following information and sends it to the negotiator. These macros are described in “Reserved Macros” on page 123.
  - 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 2: Manage a Job's Status Using Control Expressions” on page 98.
- 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 does the following:

Determines whether its own state permits a new job to run	
If:	**Then this happens:**
Yes, it can start a new job	The startd forks a **starter** process.
No, it cannot start a new job	The startd rejects the request for one of the following reasons: • Jobs have been suspended, flushed, or drained • The job limit set for the **MAX_STARTERS** keyword has been reached • There are not enough classes available for the designated job class

- Receives requests from the master (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 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.
- Periodically generates a new checkpoint file, provided checkpointing has been enabled, and sends it to the scheduling machine.

The negotiator Daemon

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

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

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

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

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

The negotiator performs the following functions:

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

  - Remove a job
  - Hold or release a job
  - Favor or unfavor a user or a job.
- Receives notification of schedd resets indicating that a schedd has restarted.

The kbdd Daemon

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

The kbdd daemon notifies the startd daemon when it detects keyboard or mouse activity; however, kbdd is *not* interrupt driven. It sleeps for the number of seconds defined by the **POLLING_FREQUENCY** keyword in the LoadLeveler configuration file, and then determines if X events, in the form of mouse or keyboard activity, have occurred. For more information on the configuration file, see Chapter 5, "Administering and Configuring LoadLeveler" on page 73.

LoadLeveler Job States

As LoadLeveler processes a job, the job moves into various states. Some states are unique to specific daemons; for example, only the negotiator places a job in the NotQueued state. For more information on daemons, see "Daemons and Processes" on page 13. Possible job states are:

Completed The job has 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 accummulate 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 295. (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 acknowleges 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.)

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 removed (cancelled), either by LoadLeveler or by the user.
Remove Pending	The job is in the process of being removed, but not all associated machines have acknowledged the removal of the job.
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.
System User Hold	The job has been put in system hold and user hold.
User Hold	The job has been put in user hold.
Vacate	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 Vacate state are: the machine where the job was running was flushed, the VACATE expression in he 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 2: Manage a Job's Status Using Control Expressions” on page 98.

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.

Using LoadLeveler

© Copyright IBM Corp. 1993, 1996

Chapter 3. 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 more specific information on parallel jobs, see Chapter 4, "Submitting and Managing Parallel Jobs" on page 55

Performing LoadLeveler Actions

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

- Using LoadLeveler commands. This method is discussed in this chapter.
- Using the LoadLeveler graphical user interface (GUI). This method is discussed in "Building and Submitting Jobs Using the Graphical User Interface" on page 232.

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 specifies the name of the job as well as the job steps that you want to submit, and can contain other LoadLeveler statements. To specify a binary to be executed, you must 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 "Job Command File Keywords" on page 40. Any LoadLeveler keyword or keyword statement is preceded by # @. These lines are treated as comments by the UNIX shell.
- 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. Any line whose first non-blank character is a pound sign (#) and is not a LoadLeveler keyword statement is regarded as a comment.
- Shell command statements: If you use a shell script as the executable, the job command file can include shell commands.
- Macros: See "Job Command File Macros" on page 53 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.

© Copyright IBM Corp. 1993, 1996

Job Command File Syntax

The following general rules apply to job command files.

- Keyword statements begin with # @. There can be any number of blanks between the # and the @.
- Comments begin with #.
- 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 #@. See Figure 15 on page 37 for an example of using the back-slash.
- Keywords are *not* case sensitive. This means you can enter them in lower case, upper case, or mixed case.

Serial Job Command File

Figure 10 is an example of a simple serial job command file which is run from your home directory. The job command file reads input file from your home directory and writes the results to a file in your home directory.

```
#   The name of this job command file is file.cmd.
#   The input file is longjob.in1 and the error file is
#   longjob.err1. The queue statement marks the end of
#   the job and submits the executable.
#
# @ executable = longjob
# @ input = longjob.in1
# @ output = longjob.out1
# @ error = longjob.err1
# @ queue
```

Figure 10. Serial Job Command File

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. All information in the first step is inherited by all succeeding steps.

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 11 on page 25 contains two separate job steps. Notice that `step1` is the first job step to run and that `step2` is a job step that runs only if `step1` exits with the correct exit status.

```
# This job command file lists two job steps called "step1"
# and "step2".  "step2" only runs if "step1" completes
# with exit status = 0.  Each job step requires a new
# queue statement.
#
# @ step_name = step1
# @ executable = executable1
# @ input = step1.in1
# @ output = step1.out1
# @ error = step2.err1
# @ queue
# @ dependency = (step1 == 0)
# @ step_name = step2
# @ executable = executable2
# @ input = step2.in1
# @ output = step2.out1
# @ error = step2.err1
# @ queue
#
```

Figure 11. Job Command File with Multiple Steps

In Figure 11, `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 11, 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 12, `step2` uses the **executable** statement defined in `step1`:

```
# This job command file uses only one executable for
# both job steps.
#
# @ step_name = step1
# @ executable = executable1
# @ input = step1.in1
# @ output = step1.out1
# @ error = step1.err1
# @ queue
# @ dependency = (step1 == 0)
# @ step_name = step2
# @ input = step2.in1
# @ output = step2.out1
# @ error = step2.err1
# @ queue
```

Figure 12. Job Command File with Multiple Steps and One Executable

See “Additional Job Command File Examples” on page 35 for more information.

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 4, “Submitting and Managing Parallel Jobs” on page 55

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 27 for information on submitting a job to a machine outside the cluster.) You can submit a job command file either by using the GUI or the **llsubmit** command.

When you submit a job, LoadLeveler assigns the job a three part identifier. This identifier includes:

- Machine name: the name of the machine that schedules the job, 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 32.

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 Chapter 9, “Routing Jobs to NQS Machines” on page 177 For more information on the **llsubmit** command, see “llsubmit - Submit a Job” on page 219.

Submitting a Job Command File Using a Submit-Only Machine: You can submit jobs from submit-only machines. Submit-only machines allow machines that are not part of the LoadLeveler cluster to submit jobs to the cluster. You can submit a job using either the submit-only version of the GUI or the **llsubmit** command.

To start a submit-only machine, follow the procedure in the *LoadLeveler Installation Memo*, or consult the appropriate README file.

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

Managing Jobs

This sections tells you how to edit a job command file, query the status of a job, place and release a hold on a job, cancel a job, change the priority if a job, checkpoint a job step, and display machine status.

Editing a Job Command File

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

Querying the Status of a Job

Once you submit a job, you can query the status of the job to determine, for example, if it is still in the queue or if it is running. You also receive other job status related information such as the job ID and job owner. You can query the status of a LoadLeveler job either by using the GUI or the **llq** command. For an example of querying the status of a job, see "Step 4: Display the Status of a Job" on page 32.

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 207.

Querying Multiple LoadLeveler Clusters

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

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

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

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

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

For example, you can enter the following query command:

```
$ llq
```

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

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

Note that the machine from which you issue the **llq** command is treated 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 system administrator can place and release a user hold on a job. Only a system 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 33 and "Step 7: Release a Hold on a Job" on page 33.

As a user or system administrator, you can also use the **startdate** keyword described in "Job Command File Keywords" on page 40 to place a hold on a job. This keyword allows you to specify when you want to run the job. If you don't specify this keyword in your job command file, the job will run at the earliest possible time. The syntax is:

startdate = *date time*

where the format of the date is *MM/DD/YY* and the format of the time is *HH:mm(:ss)*. For example, to specify that a job run on August 23, 1996 at 1:30 pm, you would enter:

```
startdate = 08/23/96 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.

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 cancel jobs from a submit-only machine.

Checkpointing a Job Step

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. You can checkpoint serial jobs. You cannot, however, checkpoint parallel jobs. For a detailed explanation of checkpointing, see “Step 13: Enable Checkpointing” on page 113

Setting and Changing the Priority of a Job

LoadLeveler uses the priority of a job to determine its position among a list of all jobs waiting to be dispatched. You can use the **llprio** command to change job priorities. See “llprio - Change the User Priority of Submitted Job Steps” on page 205 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 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 variable you can use in this expression, see “Step 5: Prioritize the Queue Maintained by the Negotiator” on page 103.

How Does a Job's Priority Affect Dispatching Order?

LoadLeveler schedules jobs based on the *adjusted system priorty*, 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 1 on page 30 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 are in the same class, Joe2's priority is swapped with that of Joe1 when the adjusted system priority is calculated. This results in Joe2 getting an adjusted system priority of 9, and Joe1 getting an adjusted system priority of 8. Similarly, the Class A jobs belonging to Rich (Rich1 and Rich3) also have their priorities swapped. The priority of the job Rich2 does not change, since this job is in a different class (Class B).

Table 1. How LoadLeveler Handles Job Priorities

Job	User Priority	System Priority (SYSPRIO)	Class	Adjusted System Priority
Rich1	50	10	A	6
Joe1	10	9	A	8
Joe2	20	8	A	9
Rich2	100	7	B	7
Rich3	90	6	A	10

Working with Machines

Throughout this book, the terms *client*, *workstation*, *machine*, and *node* refer to the machines in your cluster. For information on the roles these machines can play, see “Machines and Workstations” on page 5.

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 33.

- 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.

A Simple Task Scenario 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 Job Command File Examples” on page 35.

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 13:

```
# 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 13. 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 219.

Step 4: Display the Status of a Job

To display the status of the job you just submitted, use the **llq** command. This command returns information about all jobs in the LoadLeveler queue:

```
llq wizard.22
```

where `wizard` is the machine name to which you submitted the job, and `22` is the job ID. You can also query this job using the command **llq wizard.22.0**, where `0` is

the step ID. For more information, see see “llq - Query Job Status” on page 207.

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 29.

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 205.

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 200.

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 200.

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 214.

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 188.

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 214.

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 Job Command File Examples

"Serial Job Command File" on page 24 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 create multiple LoadLeveler scripts, each specifying a different input and output file as described previously in Figure 15 on page 37. 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 14 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 14 results in your program running five times, each time with a different input file. LoadLeveler generates the output file from the LoadLeveler job step IDs. This ensures that the results from the different submissions are not merged.

```
# @ executable = longjob
# @ input = longjob.in.$(stepid)
# @ output = longjob.out.$(jobid).$(stepid)
# @ error = longjob.err.$(jobid).$(stepid)
# @ queue
# @ queue
# @ queue
# @ queue
# @ queue
```

Figure 14. Job Command File with Varying input Statements

To submit the job, type the command:

```
llsubmit longjob.cmd
```

LoadLeveler responds by issuing the following:

```
submit: The job "ll6.23" with 5 job steps has been submitted.
```

The following table illustrates where the five job steps get their input, write their output, and log their messages:

Job:	Gets input from:	Writes output to:	Writes errors to:
23.0	longjob.in.0	longjob.out.23.0	longjob.err.23.0
23.1	longjob.in.1	longjob.out.23.1	longjob.err.23.1
23.2	longjob.in.2	longjob.out.23.2	longjob.err.23.2
23.3	longjob.in.3	longjob.out.23.3	longjob.err.23.3
23.4	longjob.in.4	longjob.out.23.4	longjob.err.23.4

Example 2: Using Macros in a Job Command File

Figure 15 on page 37 shows how you can use LoadLeveler macros 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 **$(base_executable)** macro is set to be the “base” portion of the **executable**, which is **oslsslv**.
- The **$(host)** macro is equivalent to **$(hostname)**. Similarly, **$(jobid)** and **$(stepid)** are equivalent to **$(cluster)** and **$(process)**, respectively.

In Job Step 2:

- This job step is executed only if the return codes from Step 0 and Step 1 are both equal to zero.
- The initial working directory for Step 2 is explicitly specified.

```
# Job step 0 ------------------------------------------------------------------------------
#   The names of the output and error files created by this job step are:
#
#     output: /u/rhclark/OSL/oslsslv_out/lltest1.122.0.out
#     error : /u/rhclark/OSL/oslsslv_err/lltest1_122_0_err
#
# @ job_name = OSL
# @ step_name = step_0
# @ executable = oslsslv
# @ arguments = -maxmin=min -scale=yes -alg=dual
# @ environment = OSL_ENV1=20000; OSL_ENV2=500000
# @ requirements = (Arch == "R6000") && (OpSys == "AIX41")
# @ input  = test01.mps.$(stepid)
# @ output = $(executable)_out/$(host).$(jobid).$(stepid).out
# @ error  = $(executable)_err/$(host)_$(jobid)_$(stepid)_err
# @ queue
#
# Job step 1 ------------------------------------------------------------------------------
#   The names of the output and error files created by this job step are:
#
#     output: /u/rhclark/OSL/oslsslv_out/lltest1.122.1.out
#     error : /u/rhclark/OSL/oslsslv_err/lltest1_122_1_err
#
# @ step_name = step_1
# @ executable = ~rhclark/$(job_name)/oslsslv
# @ arguments = -maxmin=max -scale=no -alg=primal
# @ environment = OSL_ENV1=60000; OSL_ENV2=500000; \
                  OSL_ENV3=70000; OSL_ENV4=800000;
# @ input  = ~rhclark/$(job_name)/test01.mps.$(stepid)
# @ output = ~rhclark/$(job_name)/$(base_executable)_out/$(hostname).$(cluster).$(process).out
# @ error  = ~rhclark/$(job_name)/$(base_executable)_err/$(hostname)_$(cluster)_$(process)_err
# @ queue
#
# Job step 2 ------------------------------------------------------------------------------
#   The names of the output and error files created by this job step are:
#
#     output: /u/rhclark/OSL/oslsslv_out/lltest1.122.2.out
#     error : /u/rhclark/OSL/oslsslv_err/lltest1_122_2_err
#
# @ step_name = OSL
# @ dependency = (step_0 == 0) && (step_1 == 0)
# @ comment = oslsslv
# @ initialdir = /u/rhclark/$(step_name)
# @ arguments = -maxmin=min -scale=yes -alg=dual
# @ environment = OSL_ENV1=300000; OSL_ENV2=500000
# @ input  = test01.mps.$(stepid)
# @ output = $(comment)_out/$(host).$(jobid).$(stepid).out
# @ error  = $(comment)_err/$(host)_$(jobid)_$(stepid)_err
# @ queue
```

Figure 15. Using Macros in a Job Command File

Example 3: Using the Job Command File as the Executable

The name of the sample script shown in Figure 16 on page 39 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.)
- If your LoadLeveler cluster supports multiple hardware architectures, you can submit a job that runs on some or all of the supported platforms. You can direct selected job steps to a specific platform or to a specific machine.

- 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.

  All file names that are not fully qualified are relative to the initial working directory /**u/lixin/TEST/spice**. LoadLeveler will send the job steps 0 and 1 of this job to either an AIX41 or a SunOS5 machine for execution. Job step 0 most likely will be sent to an AIX41 machine since this preference has been stated using the LoadLeveler **preferences** keyword. LoadLeveler will send job step 2 to a Solaris system for execution because a requirement for SunOS5 has been explicitly stated. Similarly, job step 3 will be sent to the machine `load141` for execution because of the explicit requirement for this machine in the **requirements** statement.

```
#!/bin/ksh
# @ job_name = spice_test
# @ arguments = spice3f5_input_$(stepid) spice3f5_output_$(stepid)
# @ input  = spice2g6_input_$(stepid)
# @ output = $(job_name)_output_$(stepid)
# @ error  = $(job_name)_error_$(stepid)
# @ initialdir = /u/lixin/TEST/spice
# @ requirements = ((Arch == "R6000") && (OpSys == "AIX41")) || ((Arch == "SPARC") && (OpSys == "SunOS5"))
# @ preferences  = (Arch == "R6000") && (OpSys == "AIX41") && (Memory > 32)
# @ queue
# @ preferences  = (Arch == "SPARC") && (OpSys == "SunOS5")
# @ queue
# @ requirements = (Arch == "SPARC") && (OpSys == "SunOS5")
# @ queue
# @ arguments = spice3f5_input_benchmark1 spice3f5_output_benchmark1
# @ requirements = (Machine == "load141.kgn.ibm.com") && (Arch == "R6000") && (OpSys == "AIX41")
# @ queue
OS_NAME=`uname`
OS_RELEASE=`uname -r`

case $OS_NAME in
   AIX)
      echo "Running $OS_NAME version of spice3f5" > $2
      AIX_bin/spice3f5 < $1  >> $2  2>&1
      echo "Running $OS_NAME version of spice2g6"
      AIX_bin/spice2g6
      ;;
   SunOS)
      case $OS_RELEASE in
         5*)
            echo "Running $OS_NAME version of spice3f5" > $2
            SOLARIS_bin/spice3f5 < $1  >> $2  2>&1
            echo "Running $OS_NAME version of spice2g6"
            SOLARIS_bin/spice2g6
            ;;
         *)
            echo "spice3f5 for $OS_NAME.$OS_RELEASE is not available" > $2
            echo "spice2g6 for $OS_NAME.$OS_RELEASE is not available"
            ;;
      esac
      ;;
   *)
      echo "spice3f5 for $OS_NAME is not available" > $2
      echo "spice2g6 for $OS_NAME is not available"
      ;;
esac
```

Figure 16. Job Command File Used as the Executable

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. Note that if you specify a keyword in the first job step of a multi-step job, its value is inherited by the other job steps.

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 7, "Gathering Job Accounting Data" on page 163.

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
```

checkpoint

Specifies whether you want to checkpoint your serial program.

Specify **user_initiated** if you coded the LoadLeveler **ckpt** function call in your program to cause a checkpoint to occur.

Specify **system_initiated** if you want LoadLeveler to automatically checkpoint your program at preset intervals. Specify **system_initiated** if you want both system initiated and user initiated checkpointing.

Specify **no** if you do not want your program to be checkpointed. The default is `no`.

When you enable checkpointing, you must include **restart=yes** in your job command file.
If you have existing job command files that specify **checkpoint=yes**, this will default to **system_initiated**.

To use checkpointing, your program must be linked with the LoadLeveler libraries **libload.a** and **libckpt.a**; otherwise LoadLeveler will reject the program at submit time.

The syntax is:

checkpoint = user_initiated | system_initiated | no

For more information, see “Step 13: Enable Checkpointing” on page 113.

class

Specifies the name of a job class defined locally in your cluster. If not specified, the default job class is assigned. You can use the **llclass** command to find out information on job classes.

The syntax is:

class = *name*

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

```
class = largejobs
```

comment

Specifies text describing characteristics or distinguishing features of the job.

core_limit

Specifies the maximum size of a core file.

The syntax is:

core_limit = *hardlimit,softlimit*

Some examples are:

```
core_limit = 125621,10kb
core_limit = 5621kb,5000k
core_limit = 2mb,1.5mb
core_limit = 2.5mw
core_limit = unlimited
core_limit = rlim_infinity
core_limit = copy
```

See “Limit Keywords” on page 88 for more information on the values and units you can use with this keyword.

cpu_limit

Specifies the maximum amount of CPU time that a submitted job step can use.

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 88 for more information on the values and units you can use with this keyword.

data_limit

Specifies the maximum size of the data segment to be used by the job step. If you specify this keyword in the first job step, its value is inherited by the other job steps.

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 88 for more information on the values and units you can use with this keyword.

dependency

Specifies the dependencies between job steps.

The syntax is:

```
dependency = expression
```

where the syntax for the *expression* is:

```
step_name operator value
```

where *step_name* (as described in “step_name” on page 53) must be a previously defined job step and *operator* can be one of the following:

== equal to
!= not equal to
<= less than or equal to
>= greater than or equal to
< less than
> greater than
&& and
|| or

The *value* is usually a number which specifies the job return code to which the *step_name* is set. It can also be one of the following LoadLeveler defined job step return codes:

CC_NOTRUN The return code set by LoadLeveler for a job step which is not run because the dependency is not met.

CC_REMOVED The return code set by LoadLeveler for a job step which is removed from the system (because, for example, **llcancel** was issued against the job step).

In the following example, if the return code for step1 equals zero and the return code for step2 is greater than zero, then the step that contains this dependency statement will run.

```
dependency = (step1 == 0) && (step2 > 0)
```

In the following example, step1 will run with the executable called **myprogram1**. Step2 will run only if LoadLeveler removes step1 from the system. If step2 does run, the executable called **myprogram2** gets run.

```
# Beginning of step1
# @ step_name = step1
# @ executable = myprogram1
# @ ...
# @ queue
# Beginning of step2
# @ step_name = step2
# @ dependency = step1 == CC_REMOVED
# @ executable = myprogram2
# @ ...
# @ queue
```

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. In this case, 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
```

environment

Specifies your initial environment variables when your job step starts. If you only specify this keyword in the first job step, its value is inherited to the other job steps. 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 only specify this keyword in the first job step, its value is inherited to the other job steps. If not specified, 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.
- For Parallel Virtual Machine extended (PVMe) jobs – specifies the PVMe daemon.

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)
# @ executable = /usr/lpp/pvm3/pvmd3e  (for PVMe jobs)
```

Note that the **executable** statement automatically sets the **$(base_executable)** macro, which is the file name of the executable without the directory component. See Figure 15 on page 37 for an example of using the **$(base_executable)** macro.

file_limit

Specifies the maximum size of the file.

The syntax is:

```
file_limit = hardlimit,softlimit
```

For example:

```
file_limit = 120mb,100mb
```

See “Limit Keywords” on page 88 for more information on the values and units you can use with this keyword.

group

Specifies the LoadLeveler group. If not specified, this defaults to the default group. The syntax is:

```
group = group_name
```

For example:

```
group = my_group_name
```

hold

Specifies whether you want to place a hold on your job step when you submit it. If you only specify this keyword in the first job step, its value is inherited to the other job steps. There are three types of holds:

user	Specifies user hold
system	Specifies system hold
usersys	Specifies user and system hold

The syntax is:

```
hold = user|system|usersys
```

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

```
hold = user
```

To remove the hold on the job, you can use either the GUI or the **llhold -r** command.

image_size

Maximum virtual image size, in kilobytes, to which your program will grow during execution. LoadLeveler tries to execute your job steps on a machine that has enough resources to support executing and checkpointing your job step. If your job command file has multiple job steps, the job steps will not necessarily run on the same machine, unless you explicitly request that they do. If you only specify this keyword in the first job step, its value is inherited to the other job steps.

If you do not specify the image size of your job command file, the image size is that of the executable. If you underestimate the image size of your job step, your job step may crash due to the inability to acquire more address space. If you overestimate the image size, LoadLeveler may have difficulty finding machines that have the required resources.

The syntax is:

image_size = *number*

For example, to set an image size of 11 kb, the keyword statement would be:

```
image_size = 11
```

initialdir

The pathname 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. Filenames mentioned in the command file which do not begin with a / are relative to the initial directory. If you specify this keyword only in the first job step, its value is inherited to the other job steps. The initial directory must exist on the submitting machine as well as on the machine where the job runs.

The syntax is:

initialdir = *pathname*

For example:

```
initialdir  = /var/home/mike/ll_work
```

input

Specifies the name of the file to use as standard input (stdin) when your job step runs. If not specified, the file **/dev/null** is used. If you only specify this keyword in the first job step, its value is inherited to the other job steps.

The syntax is:

input = *filename*

For example:

```
input = input.$(process)
```

For PVMe jobs, **input** refers to the executable name or the shell script used as input. For more information on PVMe jobs, see Chapter 4, “Submitting and Managing Parallel Jobs” on page 55

job_cpu_limit

Specifies the maximum CPU time to be used by all processes of a job step. For example, if a job step forks to produce multiple processes, the sum total of CPU consumed by all of the processes is added and controlled by this limit.

The syntax is:

job_cpu_limit = *hardlimit,softlimit*

For example:

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

See “Limit Keywords” on page 88 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 and/or numbers.

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. If you only specify this keyword in the first job step, its value is inherited to the other job steps. Valid entries are:

pvm3 PVM jobs. This is not for PVMe jobs.

parallel For other parallel jobs.

serial For serial jobs.

If not specified, the job type is serial.

The syntax is:

job_type = *string*

For example:

```
job_type = pvm3
```

max_processors

Specifies the maximum number of processors requested for this parallel job. If you only specify this keyword in the first job step, its value is inherited by other job steps. The default is **min_processors**.

The syntax is:

max_processors = *number*

For example:

```
max_processors = 6
```

min_processors

Specifies the minimum number of processors requested for this parallel job. If you only specify this keyword in the first job step, its value is inherited by other job steps. The default is 1.

The syntax is:

min_processors = *number*

For example:

```
min_processors = 4
```

notification

Specifies when the user specified in **notify_user** is sent mail. If you specify this keyword only in the first job step, its value is inherited by other job steps. The options are:

always Notify the user when the job begins, ends, or if it incurs error conditions.

error Notify the user only if the job fails.

start Notify the user only when the job begins.

never Never notify the user.

complete Notify the user only when the job ends. This is the default.

The syntax is:

```
notification = always|error|start|never|complete
```

For example, if you want to be notified with mail only when your job step completes, your notification keyword would be:

```
notification = complete
```

When a LoadLeveler job ends, you may receive UNIX mail notification indicating the job exit status. For example, you could get the following mail message:

```
Your LoadLeveler job
      myjob1
exited with status 4.
```

The return code 4 is from the user's job. LoadLeveler retrieves the return code and returns it in the mail message, but it is not a LoadLeveler return code.

notify_user

Specifies the user to whom mail is sent based on the **notification** keyword. If you only specify this keyword in the first job step, its value is inherited to the other job steps. 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 you only specify this keyword in the first job step, its value is inherited to the other job steps. If not specified, the file /**dev**/**null** is used.

The syntax is:

```
output = filename
```

For example:

```
output = out.$(jobid)
```

parallel_path

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

For example:

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

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

$PVM_ARCH, and $PVM_ROOT are PVM environment variables. For more information, see the appropriate PVM 3.3 documentation.

preferences

List of characteristics that you prefer be available on the remote machine executing job steps in this cluster. LoadLeveler attempts to supply remote machines that meet the preferences. If such a machine is not available, LoadLeveler will then supply machines which meet only the requirements. If you only specify this keyword in the first job step, its value is inherited to the other job steps. See “requirements” on page 50 for a description of the characteristics that can be specified.

The syntax is:

preferences = *Boolean_expression*

Examples:

```
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

List of requirements which a remote machine must meet to execute any job steps in the cluster. You can specify multiple requirements in a single requirements statement.

The syntax is:

```
requirements = Boolean_expression
```

For example,

```
requirements = (Memory >=16) && (Arch == "R6000")
```

See Figure 16 on page 39 for other examples of requirements statements.

The requirements supported are:

Memory The amount of physical memory required in megabytes.

Feature The name of a feature defined locally on your LoadLeveler cluster. When specifying a feature, be sure to specify the feature in the same way in which the feature is specified in the machine configuration file on the machine that supports the feature. To find out what your feature is, use the **llstatus** command.

Machine The name(s) of machines on which to run the jobs. When specifying a machine, be sure to specify it in the same way in which it is specified in the machine configuration file.

To specify multiple machines for a parallel job, you can enter, for example:

```
requirements = (Machine == { "ll6" "ll5" "ll0" })
```

You can also set the 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. The syntax for this is:

```
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
```

Disk Specifies the amount of disk space in kilobytes you believe is required in the LoadLevler **execute** directory on the remote workstation.

Arch Identifies the machine architecture. It describes the particular kind of UNIX platform for which your executable has been compiled. The default is that of the submitting machine.

OpSys Identifies the operating system on the machine. It describes the particular kind of UNIX platform for which your executable has been compiled. The default is that of the submitting machine. The executable must be compiled on a machine that matches these requirements.

Adapter Identifies the adapter on an SP system to use when processing a parallel job step. If you specify a value for this keyword, your job must be run on an SP system. If you only specify this keyword in the first job step, its value is inherited by the other job steps.

The syntax is:

```
Adapter == adapter_name
```

The valid adapter names are:

hps_ip Refers to an SP high-performance switch in IP mode.

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

ethernet Refers to Ethernet. This is the default.

fddi Refers to Fiber Distributed Data Interface (FDDI).

tokenring Refers to Token Ring.

fcs Refers to Fiber Channel Standards.

For example:

```
Adapter == "hps_ip"
```

Pool Identifies the number associated with the parallel cluster. If you specify a value for this keyword, your job must run on the SP system. If you only specify this keyword in the first job step, its value is inherited to the other job steps.

The syntax is:

```
Pool == number
```

For example:

```
Pool == 4
```

restart

Restarts job steps that vacate before completing. If you set **restart=no**, the job is cancelled after it is vacated. If you set **restart = no**, you cannot set **checkpoint = yes**. The default is `restart = yes`.

The syntax is:

```
restart = yes|no
```

rss_limit

Specifies the maximum resident set size.

The syntax is:

rss_limit = *hardlimit*,*softlimit*

For example:

```
rss_limit=120,100
```

Because no units are specified in the above example, bytes are assumed. See “Limit Keywords” on page 88 for more information on the values and units you can use with this keyword.

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. If you specify this keyword only in the first job step, its value is inherited by other job steps.

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 maximum size of the stack that is created. If you specify this keyword only in the first job step, its value is inherited by other job steps.

The syntax is:

stack_limit = *hardlimit*,*softlimit*

For example:

```
stack_limit = 120,100
```

Because no units are specified in the above example, bytes are assumed. See “Limit Keywords” on page 88 for more information on the values and units you can use with this keyword.

startdate

Specifies when you want to run the job step. If not specified, the current date and time are used. If you specify this keyword only in the first job step, its value is inherited by other job steps.

The syntax is:

startdate = *date time*

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

For example, if you want the job to run on June 7th, 1996 and you don't care what time of day it runs, you would issue:

```
startdate = 06/07/96
```

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. If you don't specify step names, 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
```

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. If you specify this keyword only in the first job step, its value is inherited by other job steps.

The syntax is:

user_priority = *number*

where *number* is a number between 0 and 100, inclusive. A higher number indicates the job step will be selected before a job step with a lower number. The default priority is 50. Note that this is not the UNIX *nice* priority.

This priority guarantees the order the jobs are considered for dispatch. It does not guarantee the order in which they will run.

wall_clock_limit

Sets the 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 starter process is executed.

The syntax is:

wall_clock_limit = *hardlimit*,*softlimit*

An example is:

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

See “Limit Keywords” on page 88 for more information on the values and units you can use with this keyword.

Job Command File Macros

LoadLeveler has several macros which you can use in a job command file. These macros are useful for distinguishing between output and error files.

You can refer to macros in mixed case, but you must specify them using the following syntax:

$(*macro_name***)**

The following macros are available to you:

$(host) The hostname of the machine from which the job was submitted. In a job command file, the **$(host)** macro and the **$(hostname)** macro are equivalent.

$(domain) The domain of the host from which the job was submitted.

$(jobid) The sequential number assigned to this job by the submitting machine. The **$(jobid)** macro and the **$(cluster)** macro are equivalent.

$(stepid) The sequential number assigned to this job step when multiple queue statements are used with the job command file. The **$(stepid)** macro and the **$(process)** macro are equivalent.

In addition, the following keywords are also available as macros. However, you must define them in the job command file. These keywords are described in detail in "Job Command File Keywords" on page 40.

$(executable)
$(class)
$(comment)
$(job_name)
$(step_name)

Note that for the **$(comment)** macro, the keyword definition must be a single string with no blanks. Also, the **executable** statement automatically sets the **$(base_executable)** macro, which is the file name of the executable without the directory component. See Figure 15 on page 37 for an example of using the **$(base_executable)** macro.

Example 1

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

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

Example 2

The following job command file creates an **output** file called **computel.step1.March05**.

```
# @ comment    = March05
# @ job_name   = computel
# @ step_name  = step1
# @ executable = my_job
# @ output     = $(job_name).$(step_name).$(comment)
# @ queue
```

Chapter 4. Submitting and Managing Parallel Jobs

This chapter tells you how to submit and manage parallel jobs.

Supported Parallel Environments

LoadLeveler allows you to schedule parallel batch jobs that have been written using Parallel Virtual Machine (PVM) 3.3 or above (a public domain package), or any general parallel programming language (such as the IBM Parallel Environment Library* 1.2.1 or the Parallel Virtual Machine Extended (PVMe)* 1.3.1) which uses the LoadLeveler parallel application programming interfaces. Programmers of parallel programming environments should refer to Chapter 12, “LoadLeveler Application Programming Interfaces” on page 257 for more information on using these interfaces.

Setting Up to Allow Users to Submit Parallel Jobs

Depending upon the environment in which users will be submitting parallel jobs, there are different tasks that you may need to perform. Refer to the following table for more information:

If users are submitting to this environment:	The system administrator needs to do the following:
Cluster of workstations	No special tasks are necessary.
SP system	Optionally configure LoadLeveler to use the resource manager.
Cluster of workstations and SP system	No special tasks are necessary but you can optionally configure LoadLeveler to use the resource manager.

Setting Up to Allow Users to Submit Parallel Jobs to the SP System

You must use the resource manager if you want to use the high-performance switch in user mode or if you want the resource manager to coordinate machine usage between interactive and batch jobs on the SP system.

To use the resource manager to schedule parallel jobs on the SP system, edit the LoadLeveler administration file and add the keyword **rm_host = true** to every SP system machine stanza. When you set **rm_host** to **true**, LoadLeveler does the following:

- Polls the resource manager to identify whether the SP nodes have been dedicated to another client. If a node has been dedicated to another client, the central manager updates the machine status to Reserved and does not attempt to schedule any more jobs on that machine. If a nodes has not been dedicated, the user can specify **Adapter** and **Pool** on the the **requirements** keyword in the job command file.
- Dispatches the job, then allocates machines selected by the negotiator through the resource manager for both serial and parallel jobs. All machines are allocated as shared so that serial, parallel and interactive jobs may run

© Copyright IBM Corp. 1993, 1996

simultaneously. Machines may become dedicated through interactive jobs. The job will return to the Idle state if the allocation fails.

For more information on setting up and configuring the resource manager, refer to the *IBM RS/6000 Scalable POWERparallel Systems: Administration Guide*, (SH26-2486).

Scheduling Considerations

In LoadLeveler 1.2.0 and later releases, the negotiator daemon selects the parallel machines through *temporary reservation*. This means that when a parallel job enters the system, the central manager scans a list of machines with resources matching the job requirements. If sufficient machines to schedule the parallel job are not available, the central manager temporarily reserves as many machines as it can. As machines meeting the job requirements become available, these machines are added to the list of reserved machines. When the job's machine requirements have been met, the job is dispatched.

It is possible that a sufficient number of machines never become available. Therefore, the central manager limits the amount of time the machines may be held in reserve. When this limit has been met, all reserved machines are released and the parallel job is placed in a deferred queue where it waits to reenter the LoadLeveler queue with its original system priority. How long a job attempts to collect machines before releasing them and how long a job waits on the temporary queue are both configurable by specifying the following two keywords in the configuration file:

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 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 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.

For more information, see the NEGOTIATOR_INTERVAL keyword on page 119.

Setting Up to Allow Users to Submit PVM Jobs

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

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

For non-SP 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)** which PVM requires. For example:

```
gallifrey:  type = machine
            central_manager = true
            schedd_host = true
            alias = drwho
            pvm_root = /home/userid/loadl/1.2.0/aix32/pvm3
```

For PVM 3.3.11+ (SP), LoadLeveler does not expect to find PVM installed in **~loadl/pvm3**. PVM 3.3.11+ (SP) must be installed in a directory accessable 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_processors = 15  # max of 15 processors per user per job
```

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

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

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

Note:

> If you use AIX 4.1.x, you should specify the option **-DSOCKADHASLEN** when compiling the public domain PVM package. You can do this by editing the file /**pvm3**/**conf**/**RS6K.def** so that **-DSOCKADHASLEN** is added to the list of C compiler flags. Also, PVM3 assumes the command /**usr**/**ucb**/**rsh** exists on your AIX system. If your AIX 4.1.x system does not have this command, you should login as root and create a symbolic link of /**usr**/**ucb**/**rsh** to /**usr**/**bin**/**rsh**. This restriction does not apply to PVM 3.3.11+.

See “Common Set Up Problems with Parallel Jobs” on page 287 for more information.

Setting Up a Parallel Master Node

LoadLeveler allows you to define a parallel master node which LoadLeveler will select 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 class stanza and the machine stanza of the administration file:

```
# MACHINE STANZA: (optional)
mach1:     type = machine
           master_node_exclusive = true

# CLASS STANZA: (optional)
pmv3:      type = class
           master_node_requirement = true
```

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 78 and “Step 3: Specify Class Stanzas” on page 86.

Restrictions and Limitations for Submitting Parallel Jobs

The following list describes the restrictions and limitations for submitting parallel jobs:

- Job suspend and job checkpointing are not supported.
- For PVMe 1.1.3 and PVM 3.3, dynamic allocation and de-allocation of parallel machines are not supported.
- The negotiator allocates a machine once per job, even if there is more than one initiator available on the machine.
- PVM 3.3.8–3.3.10 (SP) is not supported. PVM 3.3.11+ (SP) is supported.

Submitting Parallel Jobs Using llsubmit

The following examples walk you through the process of submitting parallel jobs.

Configuration Considerations

In order to submit parallel jobs, you need to edit the job command file and add the following keywords:

min_processors
Specifies the minimum number of processors requested for this parallel job. (The default is 1, which implies a serial job.) The syntax is:

```
min_processors = number
```

max_processors
Specifies the maximum number of processors requested for this parallel job. The default is **min_processors**. The syntax is:

```
max_processors = number
```

job_type
Specifies the type of job step to process. The syntax is:

```
job_type = string
```

where *string* is:

pvm3 For PVM 3.3 (non-SP) jobs only. This keyword does not apply to PVMe jobs.

parallel For other parallel jobs including PVMe and AIX Parallel Operating Environment.

serial For serial jobs. This is the default.

parallel_path

Specifies the path to be used when starting a PVM 3.3 (non-SP) slave process. This keyword applies only to PVM 3.3 (non-SP) jobs with **job_type=pvm3**. This keyword is translated into the **ep** keyword as defined in the PVM 3.3 **hosts** file.

For example:

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

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

$PVM_ARCH, and $PVM_ROOT are PVM environment variables.

For an example of a job command file used with PVM, see "PVM 3.3 (Non-SP)" on page 62. For more information on PVM, see the appropriate PVM 3.3 documentation.

requirements

The list of requirements which a remote machine must meet to execute any job step. The requirements specified for the job are the requirements for all of the machines. Note that you can specify multiple requirements in a single **requirements** statement.

The syntax for the **requirements** statement is:

```
requirements = Boolean_expression
```

For example,

```
requirements = (Adapter == "hps_ip" && Pool == 4)
```

The requirements supported on an SP system are:

Adapter

Specifies the adapter on the SP system to use when processing a parallel job step. If you specify a value for this keyword, your job must run on the SP system. If the high-performance switch in user mode is requested by the job, no other jobs using the high-performance switch in user mode will be allowed on nodes running that job.

The syntax is:

```
Adapter == "adapter_name"
```

Where *adapter_name* can be one of the following:

hps_ip An SP high-performance switch in IP mode.
hps_user An SP high-performance switch in user mode.
ethernet Ethernet. This is the default.

fddi	Fiber Distributed Data Interface (FDDI)
tokenring	Token Ring
fcs	Fiber Channel Standards

For example:

```
Adapter == "hps_ip"
```

Pool

Specifies the number associated with the SP system parallel pool to use. If you specify a value for this keyword, your job must run on the SP system. The default is to select machines from any pool. The syntax is:

Pool == *number*

For example:

```
Pool == 4
```

For more information on adapters and pools, refer to the *IBM RS/6000 Scalable POWERparallel Systems: Administration Guide*, (SH26-2486).

Job Command File Examples

This section contains sample job command files for the following parallel environments:

- IBM AIX Parallel Operating Environment (POE) 1.2.1
- Parallel Virtual Machine Extended (PVMe) 1.3.1
- Parallel Virtual Machine (PVM) 3.3 (non-SP)
- Parallel Virtual Machine (PVM) 3.3.11+ (SP)

POE 1.2.1

Figure 17 shows a sample job command file for POE 1.2.1.

```
#
# @ executable   = /usr/bin/poe
# @ arguments    = /home/userid/my_POE_program -euilib us
# @ job_type     = parallel
# @ requirements = (Adapter == "hps_user")
# @ class        = POE
# @ output = my_POE_program.$(cluster).$(process).out
# @ error  = my_POE_program.$(cluster).$(process).err
# @ min_processors = 8
# @ max_processors = 10
# @ queue
```

Figure 17. POE 1.2.1 Job Command File – Example 1

Figure 18 on page 61 shows a second sample job command file for POE 1.2.1.

```
#
# @ job_type     = parallel
# @ requirements = (Adapter == "hps_ip" && Pool == 4)
# @ class        = POE
# @ output = my_POE_program.$(cluster).$(process).out
# @ error  = my_POE_program.$(cluster).$(process).err
# @ min_processors = 8
# @ max_processors = 10
# @ queue
/usr/bin/poe /home/userid/my_POE_program -infolevel 2
```

Figure 18. POE Job Command File – Example 2

Note the following requirements for POE jobs:

- The job must have **job_type = parallel**.
- You must specify the POE partition manager as the executable (**executable = /usr/bin/poe** as shown in Figure 17 on page 60) or invoke it from the job command file (**/usr/bin/poe /home/userid/my_POE_program -infolevel 2** as shown in Figure 18).
- You must pass the parallel executable to POE as an argument using the full path name.
- If you are running in user space mode (**Adapter == "hps_user"** as shown in Figure 17 on page 60), you must pass the flag **-euilib us** (the default is **ip**) to POE.
- You can invoke the POE partition manager only once.

PVMe 1.3.1

Figure 19 shows a sample job command file for PVMe 1.3.1.

```
#
# @ job_type     = parallel
# @ requirements = (Adapter == "hps_user")
# @ class        = PVMe
# @ output = my_PVMe_program.$(cluster).$(process).out
# @ error  = my_PVMe_program.$(cluster).$(process).err
# @ min_processors = 8
# @ max_processors = 10
# @ queue
/usr/lpp/pvm3/pvmd3e daemon_options -exec /home/userid/ my_PVMe_program program_args
```

Figure 19. Sample PVMe Job Command File

Note the following requirements for PVMe jobs:

- The job must have **job_type = parallel**.
- The PVMe daemon (pvmd3e in the example) must be invoked from the job command file.
- You must pass the parallel executable as an **-exec** argument to the PVMe daemon using the full path name.

PVM 3.3 (Non-SP)

Figure 20 shows a sample job command file for PVM 3.3 (non-SP). Before using PVM, users should contact their administrator to determine which PVM architecture (SP or non-SP) has been installed.

```
# @ executable    = my_PVM_program
# @ job_type      = pvm3
# @ parallel_path = /home/LL_userid/cmds/pvm3/$PVM_ARCH:$PVM_ROOT/lib/$PVM_ARCH
# @ class         = PVM3
# @ requirements  = (Pool == 4)
# @ output = my_PVM_program.$(cluster).$(process).out
# @ error  = my_PVM_program.$(cluster).$(process).err
# @ min_processors = 8
# @ max_processors = 10
# @ queue
```

Figure 20. Sample PVM 3.3 Job Command File

Note the following requirements for PVM 3.3 (non-SP) jobs:

- The job must have **job_type = pvm3**.
- You must specify the parallel executable as the executable.

PVM 3.3.11+ (SP)

Figure 21 on page 63 shows a sample job command file for PVM 3.3.11+ (SP). Before using PVM, users should contact their administrator to determine which PVM architecture (SP or non-SP) has been installed. The SP version should be used when a users require that their jobs run in user space.

```
# @ job_type      = parallel
# @ class         = PVM3
# @ requirements  = (Adapter == "hps_us")
# @ output = my_PVM_program.$(cluster).$(process).out
# @ error  = my_PVM_program.$(cluster).$(process).err
# @ min_processors = 3
# @ max_processors = 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 21. Sample PVM 3.3.11+ (SP) Job Command File

Note the following requirements for PVM 3.3.11+ (SP) 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 daemon. Parallel tasks cannot communicate with PVM daemon directly. The additional task will be spawned on the last processor in the LOADL_PROCESSOR_LIST. For more information on this environment variable set by LoadLeveler see “Obtaining Allocated Host Names” on page 66.

- You must use the PVM daemon and starter path dictated by the LoadLeveler administrator. The **parallel_path** keyword is ignored.
- You must export MP_EUILIB as **us** when running in user space over the switch. MP_PROCS, MP_RMPOOL and MP_HOSTFILE are ignored when running under LoadLeveler.
- You should clean up any temporary PVM log or daemon files before starting the PVM daemon.
- You must start the PVM daemon in the job script, and you must start it in the background (**$daemon_path/pvmd3 &**).
- You must compile your parallel program following the PVM guidelines for PVM 3.3.11+ (SP).
- 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 then 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 21 on page 63.

Figure 22 on page 65 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.

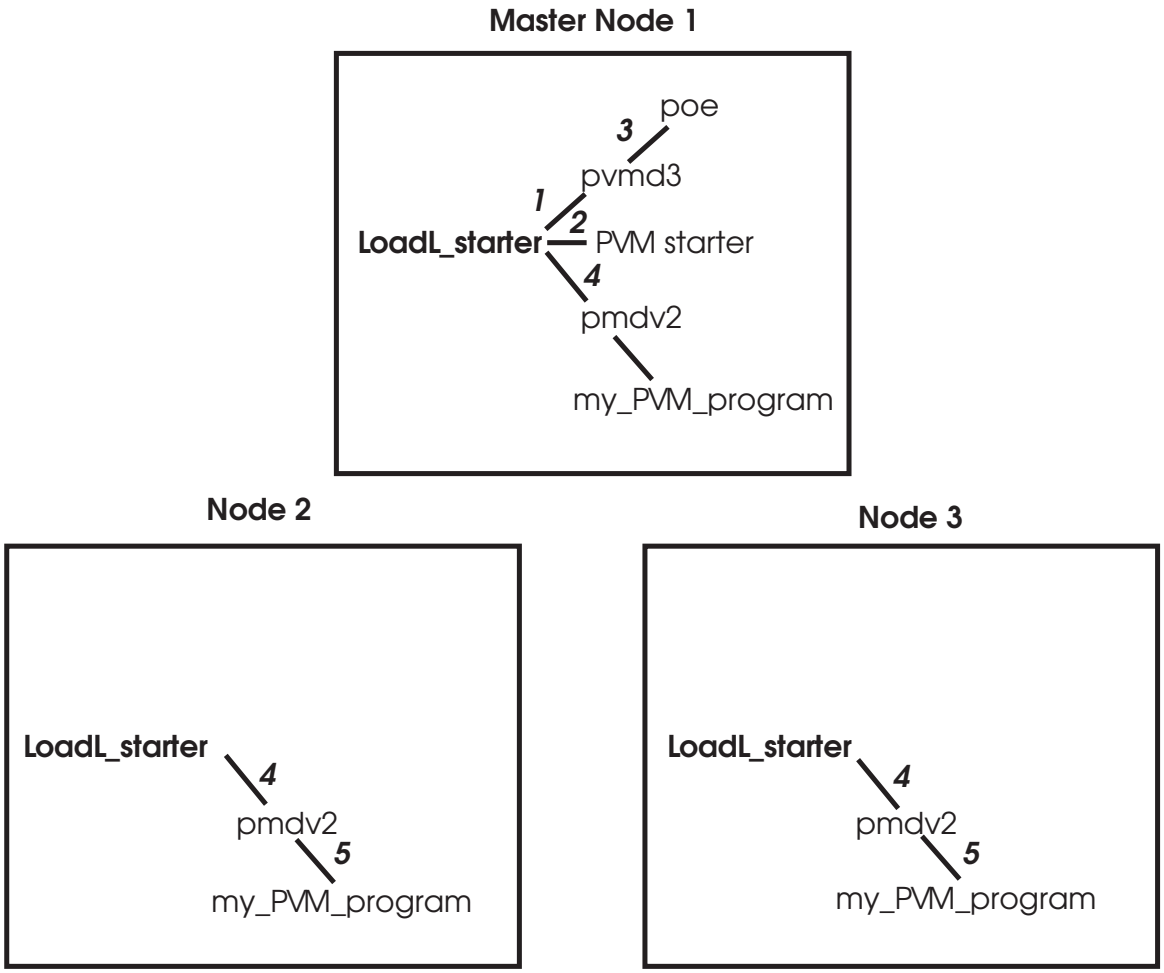


Figure 22. Sequence of Events in a PVM 3.3.11+ Job

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 Details selection in the GUI, users get a list of machines allocated to the parallel job. The following is sample output:

```
=============== Job Step k1bn09.hpssl.kgn.ibm.com.1.0 ===============
        Job Step Id: k1bn09.hpssl.kgn.ibm.com 1.0
           Job Name: k1bn09.hpssl.kgn.ibm.com.1
          Step Name: 0
  Structure Version: 8
              Owner: lluser
         Queue Date: Thu Mar  7 11:11:22 1996
             Status: Running
    Completion Date:
    Completion Code:
      User Priority: 50
       user_sysprio: 0
      class_sysprio: 0
      group_sysprio: 0
    System Priority: -2481
          q_sysprio: -2481
      Notifications: Always
        User Number: 1
 Virtual Image Size: 2 kilobytes
         Checkpoint: no
     Hold Job Until:
                Cmd: test_pvme.cmd
               Args:
                 In: /dev/null
                Out: my_parallel_job.1.0.out
                Err: my_parallel_job.1.0.err
Initial Working Directory: /u/lluser/test_parallel
         Dependency:
       Requirements: (Arch == "R6000") && (Adapter == "hps_user") <<<<<<<<<<<
        Preferences:
           Job Type: General Parallel                              <<<<<<<<<<<
     Min Processors: 2                                             <<<<<<<<<<<
     Max Processors: 2                                             <<<<<<<<<<<
Allocated host count: 2                                            <<<<<<<<<<<
      Allocated host: k1bn09.hpssl.kgn.ibm.com                     <<<<<<<<<<<
      Allocated host+ k1bn12.hpssl.kgn.ibm.com
    Submitting host: k1bn09.hpssl.kgn.ibm.com
        Notify User: lluser@k1bn09.hpssl.kgn.ibm.com
              Shell: /bin/ksh
           LL_group: No_Group
              Class: PVME
            Account: NONE
         Unix_group: staff
```

Obtaining Allocated Host Names

The above output includes information on the 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 as shown in Figure 23 on page 67. This example uses **LOADL_PROCESSOR_LIST** to perform a remote copy of a local file to all of the nodes. (Note that **LOADL_PROCESSOR_LIST** is set by LoadLeveler, not by the user.)

```
# @ executable =  my_POE_program
# @ arguments  =  /home/userid/my_POE_program
# @ output     =  my_POE_program.$(cluster).$(process).out
# @ error      =  my_POE_program.$(cluster).$(process).err
# @ class      =  POE
# @ job_type   =  parallel
# @ min_processors = 8
# @ max_processors = 12
# @ requirements =  (Adapter == "hps_user")
# @ queue

for nodeaddr in $LOADL_PROCESSOR_LIST
do
        rcp localfile $nodeaddr:/home/userid
done
```

Figure 23. Using LOADL_PROCESSOR_LIST in a Shell Script

Also, administrators can create a class for parallel jobs. Users can check the status of their parallel jobs by specifying this class in the Class field on the Jobs window of the GUI.

Obtaining Status of Machines

Administrators and users can obtain the status of machines using either the LoadLeveler **llstatus -l** command or the Machine Details selection on the GUI. The following is sample output:

```
name: "k1bn10.hpssl.kgn.ibm.com"
machine_context:
Running             = 0
ScheddAvail         = 1
StartdAvail         = 1
State               = Reserved                              <<<<<<<<<<
OpSys               = AIX32
Arch                = R6000
Machine             = k1bn10.hpssl.kgn.ibm.com
START               = T
SUSPEND             = F
CONTINUE            = F
VACATE              = F
KILL                = F
SYSPRIO             = (0 -  QDate)
MACHPRIO            = (0 -  LoadAvg)
ScheddRunning       = 0
Pending             = 0
Starting            = 0
Idle                = 0
Unexpanded          = 0
Held                = 0
Removed             = 0
RemovePending       = 0
Completed           = 0
DependantNotRun     = 0
TotalJobs           = 0
ScheddState         = 0
EnteredCurrentState = Mon Mar  6 10:30:05 1995
VirtualMemory       = 36100
Disk                = 23428
KeyboardIdle        = 7034
LoadAvg             = 0.021313
AvailableClasses    = { "PVM3" "PVME" }
Pool                = 0                                     <<<<<<<<<<
Adapter             = { "ethernet" "hps_user" "hps_ip" }    <<<<<<<<<<
ConfiguredCLasses   = { "PVM3" "PVME" }
Feature             = { }
Memory              = 64
Max_Starters        = 2
ConfigTimeStamp     = Thu Mar  7 10:29:59 1996
Cpus                = 1
Speed               = 1.000000
MasterMachPriority  = 0.000000
Subnet              = 129.40.131
CustomMetric        = 1
time_stamp: Thu Mar  7 10:34:59 1996
```

Cancelling Parallel Jobs

Administrators and users can cancel parallel jobs using either the LoadLeveler **llcancel** command or by using the Cancel option on the Actions pulldown menu on the GUI. All you need to know is the LoadLeveler job ID of the parallel job.

Migration Issues for Submitting Parallel Jobs

In this section, “LoadLeveler 1.2+” refers to LoadLeveler release 1.2.0 or later.

The following list describes how submitting parallel jobs to LoadLeveler 1.2+ differs from previous releases:

- In previous releases of LoadLeveler, users could only submit parallel jobs to the RS/6000 Scalable POWERparallel (SP) system. In LoadLeveler 1.2+,

LoadLeveler can schedule parallel jobs to any combination of supported platforms as long as the platforms are at the same LoadLeveler release level and the parallel programming environment is supported on the platform.

- In previous releases of LoadLeveler, when submitting a parallel job to the SP, LoadLeveler did not select the parallel machines or start the parallel jobs. LoadLeveler reserved machines through the resource manager and started the partition manager. The partition manager would then pick up the reserved machines from the resource manager and start the parallel jobs. In LoadLeveler 1.2+, the negotiator daemon selects the parallel machines and starts the parallel jobs. Therefore, it is a new requirement that all parallel machines run a startd daemon. It is no longer required that the resource manager be run at all. However you must run the resource manager if you want to use the high-performance switch in user mode or if you want the resource manager to coordinate machine usage between interactive and batch jobs on the SP system. Serial and parallel jobs may now be configured to run on the same machines.
- In previous releases of LoadLeveler, the LoadLeveler configuration file keyword **POOL_ID** was required. LoadLeveler used to query the resource manager for the number of available parallel machines in the **POOL_ID**. This number was then matched against the user parallel machine requirement. If enough machines were available, LoadLeveler would reserve the machines through the resource manager. In LoadLeveler 1.2.x, **POOL_ID** is not used to query the resource manager, therefore the **POOL_ID** is no longer allowed.
- In previous releases of LoadLeveler, the keyword **parallelmachines** was required in the job command file in order to specify the number of parallel machines needed to run the job. In LoadLeveler 1.2.x, this keyword has been replaced with keywords in the following list. Note that not all of the keywords are required in every circumstance:

  min_processors: Specifies the minimum number of processors requested for this parallel job. The default is 1. This keyword is required if you want to submit a parallel job.

  max_processors: Specifies the maximum number of processors requested for this parallel job. The default is **min_processors**. If you want to accept the default, you do not have to specify this keyword. The administrator may limit the number of processors that a user may request by specifying the **max_processors** keyword in the class stanza of the administration file.

  job_type: Specifies the type of job step to process. The default is a serial job step. This keyword is required if you want to submit a parallel job.

  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. The path is translated into the PVM 3.3 **ep** keyword. For more information, see “Submitting Parallel Jobs Using llsubmit” on page 58.

  requirements: Specifies the list of requirements which a remote machine must meet to execute any job steps. The requirements specified for the job are the requirements for all of the machines. In the **requirements** statement, you can add the following two keywords:

  – **Adapter:** Specifies the adapter on the SP system to use when processing a parallel job step. If you specify a value for this keyword, your job must run on the SP system. If the high-performance switch in user mode is requested by the job, no other jobs using the

high-performance switch in user mode will be allowed on machines running that job.

– **Pool:** Specifies the number associated with the SP system parallel pool to use. If you specify a value for this keyword, your job must run on the SP system.

Administering LoadLeveler

© Copyright IBM Corp. 1993, 1996

Chapter 5. Administering and Configuring LoadLeveler

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. LoadLeveler provides you with one global configuration file that you can use to manage your LoadLeveler cluster.

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 96.

Before working with these two files, you should read the following planning considerations that may 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, you must have the same uid (the system ID number for a user) and gid (the system ID number for a group) on every machine in the cluster. For example, if there are two machines in your LoadLeveler cluster, *machine_1* and *machine_2*, user `john` must have the same user ID and login group ID in the **/etc/passwd** file on both machines. If user `john` has user ID 1234 and login group ID 100 on *machine_1*, then user `john` must have the same user ID and login group ID in **/etc/passwd** on *machine_2*. This ensures that the **getuid** system call returns the same user ID on both systems. (This allows a job to run with the same group ID and user ID of the person who submitted the job.) If you do not have a user ID on one machine, your jobs will not run on that machine.

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.

© Copyright IBM Corp. 1993, 1996

Management
Managing distributed software systems is a primary concern for all system administrators. Allowing users to share filesystems 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, job classes.

Parallel Batch Jobs
If you have an SP system in your cluster, you will have to take into account SP customization considerations. Refer to the *IBM RS/6000 Scalable POWERparallel Systems* documentation for more information. Parallel jobs can also be run on a cluster of heterogeneous workstations.

LoadLeveler Interactive Session Support (ISS)
If you plan to use the optional ISS feature, you must decide which node will act as the ISS domain name server. For more information, refer to Chapter 6, “Using Interactive LoadLeveler” on page 137.

Where to Begin?

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

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

Intermediate or Beginner

If you are experienced in UNIX system administration but are unfamiliar with job scheduling systems or your experience is limited, you may want to start with the section “Administration File Structure and Syntax” on page 77 and read to the end of this chapter. This section provides a relatively slow, step-by-step approach to administering LoadLeveler. If you would rather start up LoadLeveler quickly using mostly default characteristics, follow the procedures in “Quick Set Up” on page 75.

Expert

If you are very familiar with UNIX system administration and job scheduling, and have some idea how you want to distribute your workload, go to “Quick Set Up” on page 75. 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 (and shown in Appendix C, “Sample Files” on page 313) also provide assistance.

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. 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.

Step 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 96 for more information.

Step 2. Identify yourself as the LoadLeveler administrator in the **LoadL_config** file using the **LOADL_ADMIN** keyword. The syntax of this keyword follows:

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 97 for more instructions.

Step 3. Set LoadLeveler to accept all jobs to run by coding the following statements in your **LoadL_config** file:

```
START: T
SUSPEND: F
```

With these settings, the CONTINUE, VACATE, and KILL statements have no effect. See "Step 2: Manage a Job's Status Using Control Expressions" on page 98 for more information on these statements.

Step 4. Code these keywords in your **LoadL_config** file as follows:

```
POLLING_FREQUENCY = 5
POLLS_PER_UPDATE = 24
SCHEDD_INTERVAL = 30
CLIENT_TIMEOUT = 30
NEGOTIATOR_INTERVAL = 30
MACHINE_UPDATE_INTERVAL = 300
```

These are baseline settings that provide a test environment which you can later adjust for more efficient operation. These keywords are explained in "Additional Configuration File Keywords" on page 119.

Step 5. Define a machine to act as the LoadLeveler central manager by coding one machine stanza as follows in the **LoadL_admin** file. (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 78 for more information.

Step 6. Issue the following command for each machine to be included in the LoadLeveler cluster. (Replace *hostname* with the actual name of the machine.)

Note: Issue this command for the central manager machine first.

```
llctl -h hostname start
```

See “llctl - Control LoadLeveler Daemons” on page 193 for more information.

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

```
llctl -g start
```

llctl uses **rsh** or **remsh** to start LoadLeveler on the target machine. Therefore, the administrator using **llctl** must have rsh authority on the target machine.

Administering LoadLeveler

This section explains how to perform administration tasks, and includes a step-by-step approach to administering LoadLeveler in “Customizing the Administration File” on page 78.

Administration File Structure and Syntax

The administration file is called **LoadL_admin** and it lists and defines the *machine*, *class*, *user* and *group* stanzas.

Machine stanza Describes the roles that the machines in the LoadLeveler cluster play.

Class stanza Describes the characteristics of the job classes.

User stanza Lists the LoadLeveler users.

Group stanza Describes the characteristics of a list of users that form a LoadLeveler group.

Stanzas have the following general format:

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

Figure 24. Format of Administration File Stanzas

The *label* identifies the name of the stanza; it can be a machine name, a user name, a class name, or a LoadLeveler group name

The following is a simple example of an administration file illustrating several stanzas:

```
machine_a: type            = machine
           central_manager = true

class_a:   type            = class   # class component of the priority
           priority        = 50      # priority of this class

user_a:    type            = user    # user component of the priority
           priority        = 50      # priority of this user

group_a:   type            = group
           priority        = 50      # priority of this group
```

Figure 25. Sample Administration File Stanzas

The characteristics of a stanza are:

- Every stanza has a label associated with it.
- Every stanza has a **type** field that specifies it as a **user**, **class**, **machine**, or **group** stanza.

- New line characters are ignored. This means that the different parts of a stanza may cross line boundaries or be included on the same line.
- White space is ignored, other than to delimit keyword identifiers. This eliminates confusion between tabs and spaces at the beginning of lines.
- A cross-hatch 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 are not merged. It is not recommended that you have multiple stanzas of the same label because results may be unpredictable.

  Default stanzas specify the default values to be used for stanzas of the same type that do not have all fields specified. Any value may be defaulted. You can specify a separate default stanza for LoadLeveler to maintain for each of the stanza types.

Customizing the Administration File

You can add as many stanzas as you would like to the administration file. This section tells you how to modify this file in a step-by-step manner. You do not have to perform the steps in the order that 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 will still communicate and jobs will still be scheduled to the machine but the machine will be assigned the default values described in the default machine stanza.

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.
- If you want to use the resource manager to schedule parallel jobs on the SP system, you need to add the **rm_host** keyword to the machine stanza.

Machine stanzas take the following format:

```
label: type = machine
    alias = machine_name
    central_manager = true | false | alt
    cpu_speed_scale = true | false
    master_node_exclusive = true | false
    max_jobs_scheduled = number
    name_server = list
    pvm_root = pathname
    rm_host = true | false
    schedd_host = true | false
    speed = number
    submit_only = true | false
```

Figure 26. Format of a Machine Stanza

You can specify the following keywords in a machine stanza:

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.

However, if any machine in your cluster is configured like either of the following two sample machines, then you will have to specify the **alias** keyword for those machines:

1. Machine `yugo.kgn.ibm.com`
   - The hostname command returns yugo.kgn.ibm.com.
   - The Ethernet adapter address 129.40.8.21 resolves to hostname chevy.kgn.ibm.com.

- No adapter address resolves to yugo.

You need to code the machine stanza as:

```
chevy: type = machine
       alias = yugo
```

2. Machine `rover.kgn.ibm.com`
   - The hostname command returns rover.kgn.ibm.com.
   - The FDDI adapter address 129.40.9.22 resolves to hostname rover.kgn.ibm.com.
   - The Ethernet adapter address 129.40.8.22 resolves to hostname bmw.kgn.ibm.com.
   - No route exists via the FDDI adapter to the clusters central manager machine.
   - A route exists from this machine to the central manager via the Ethernet adapter.

   You need to code the machine stanza as:

```
bmw:   type = machine
       alias = rover
```

central_manager = true| **false** | alt
where **true** designates this machine as the LoadLeveler central manager host, where the negotiator daemon runs. You must specify one and only one machine stanza identifying the central manager. For example:

```
machine_a: type = machine
         central_manager = true
```

false specifies that this machine is not the central manager.

alt specifies that this machine can serve as an alternate central manager in the event that the primary central manager is not functioning. For more information on recovering if the primary central manager is not operating, refer to “What Happens if the Central Manager Isn't Operating?” on page 290. 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 8: Specify Alternate Central Managers” on page 107.

cpu_speed_scale = true| **false**
where **true** specifies that CPU time, used for setting limits and in accounting information, is in normalized units for each machine. **false** specifies that CPU time is in native units for each machine. You can use this flag to take a machine's speed into account when determining if a job's CPU limit has been exceeded.

For example, suppose a LoadLeveler cluster consists of two machines, A and B, where machine B is three times as fast as machine A. That is, machine A has **speed=1.0**, and machine B has **speed=3.0**. Suppose a job has

job_cpu_limit = 12. The job will get 12 seconds if it runs on machine A, and four seconds if it runs on machine B.

When you specify **cpu_speed_scale = true**, the CPU time reported by the **llq -x** command is in normalized time rather than native time.

master_node_exclusive = true| **false**
where **true** specifies that this machine is used only as a master node for parallel jobs. For an SP system, a machine you designate as a master node should also have the statement **rm_host = true** in the machine stanza.

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.

name_server = *list*
where *list* is a blank-delimited list of character strings that is used to specify which nameserver(s) 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 /**u/loadl/pvm3**.

rm_host = true/**false**
Defines whether or not this machine is an SP machine which will be communicating with the resource manager. You must use the resource manager if you want to use the high-performance switch in user mode or if you want the resource manager to coordinate machine usage between interactive and batch jobs on the SP system.

When you set **rm_host** to **true**, LoadLeveler does the following:

- Polls the resource manager for SP system specific information such as the adapter, state, pool, and subpool and updates the central manager. If the resource manager has dedicated a machine for interactive or batch use, LoadLeveler takes the machine off line and marks it as reserved. The user will now be able to specify **Adapter** and **Pool** through the **requirements** keyword in the job command file.
- Allocates machines selected by the negotiator through the resource manager for both serial and parallel jobs. All machines are allocated as shared so that serial, parallel and interactive jobs may run simultaneously.

Machines may become dedicated through interactive jobs. The job will return to the queue if the allocation fails.

For more information on setting up and configuring the resource manager, refer to the *IBM RS/6000 Scalable POWERparallel Systems: Administration Guide*, (SH26-2486).

schedd_host = true| **false**
where **true** designates this as a public submission machine, used to receive job submissions from submit-only machines. Submit-only machines do not run LoadLeveler jobs.

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 6: Prioritize the Order of Executing Machines Maintained by the Negotiator" on page 104. 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 weight is 1.0.

The following example illustrates how the **speed** keyword can be used for assigning weights to machines.

If your cluster consisted of five RISC System/6000 machines that you want to have the same weight, you would not have to specify this keyword in the administration file. By default, all machines would have a weight of 1.0. If, however, you add an SP system to your cluster for parallel job processing, you may want to update the local configuration file for each node of the SP system to charge differently for resource consumption on those nodes. You would need to set the **speed** keyword to something other than 1.0 to make the SP nodes have a different weight.

For information on how the **speed** keyword can be used to schedule machines, refer to "Step 6: Prioritize the Order of Executing Machines Maintained by the Negotiator" on page 104.

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**.
- In the machine's **LoadL_config.local** file, do one of the following:
  - Set **START_DAEMONS = FALSE** if you want the machine to be strictly a submit-only machine.
  - Set **SCHEDD_RUNS_HERE = FALSE** if you want to allow the machine to run jobs.

  You must set one of these values in **LoadL_config.local** in order to allow jobs to be submitted to another machine.

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 is set to true:

```
#
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
           alias = brianne
           pvm_root = /u/brianne/loadl/1.2.0/aix32/pvm3
```

Step 2: Specify User Stanzas

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

User stanzas take the following format:

label: **type = user**
account = *list*
default_class = *list*
default_group = *group name*
maxidle = *number*
maxjobs = *number*
maxqueued = *number*
max_processors = *number*
priority = *number*

Figure 27. 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. Additionally, if no **default_class** appears in the default user stanza, then an error occurs when a user attempts to submit a job.

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. Additionally, if no **default_group** appears in the default user stanza, then an error occurs when a user attempts to submit a job.

If you specify **default_group = Unix_Group**, LoadLeveler sets the user's LoadLeveler group to his or her primary UNIX group (as defined in the /**etc**/**passwd** file).

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 295.

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. If no value is found, or the limit found is -1, then 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 295.

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. If no value is found, or the limit found is -1, then 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 295.

max_processors = *number*
where *number* specifies the maximum number of processors this user can request for a parallel job in the job command file. The default is -1, which means there is no limit.

priority = *number*
where *number* can be any integer. If not specified, the default is 50. 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 5: Prioritize the Queue Maintained by the Negotiator" on page 103 for more information about the **UserSysprio** keyword.

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

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

Step 3: Specify Class Stanzas

The information in a class stanza defines characteristics for that class. Class stanzas are optional. They take the following format:

```
label: type = class
    admin= list
    class_comment = "string"
    exclude_groups = list
    exclude_users = list
    include_groups = list
    include_users = list
    master_node_requirement = true | false
    maxjobs = number
    max_processors = number
    nice = value
    NQS_class = true | false
    NQS_submit = name
    NQS_query = queue names
    priority = number
    core_limit = hardlimit,softlimit
    cpu_limit = hardlimit,softlimit
    data_limit = hardlimit,softlimit
    file_limit = hardlimit,softlimit
    job_cpu_limit = hardlimit,softlimit
    rss_limit = hardlimit,softlimit
    stack_limit = hardlimit,softlimit
    wall_clock_limit = hardlimit,softlimit
```

Figure 28. Format of a Class Stanza

You can specify the following keywords in a class stanza:

admin = *list*
: where *list* is a blank-delimited list of administrators for this class. These administrators can hold, release, and cancel jobs in this class.

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 length of the string cannot exceed 1024 characters.

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.

include_groups = *list*
where *list* is a blank-delimited list of groups who are allowed to submit jobs of that *class name*. If provided, this list limits groups of that class to those on the list. Do not specify both a list of included groups and a list of excluded groups. Only one of these may be used for any class. The default is to include all groups.

include_users = *list*
where *list* is a blank-delimited list of users who are permitted to submit jobs of that *class name*. If provided, this list limits users of that class to those on the list. Do not specify both a list of included users and a list of excluded users. Only one of these may be used for any class. The default is to include all users.

master_node_requirement = true|**false**
where **true** specifies that parallel jobs in this class require the master node feature. For these jobs, LoadLeveler allocates the first node (called the "master") on a machine having the **master_node_exclusive = true** setting in its machine stanza. If most or all of your parallel jobs require this feature, you should consider placing the statement **master_node_requirement = true** in your default class stanza. Then, for classes that do not require this feature, you can use the statement **master_node_requirement = false** in their class stanzas to override the default setting. One machine per class should have the **true** setting; if more than one machine has this setting, normal scheduling selection is performed.

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. If no value is found, or the limit found is -1, then no limit is placed on the number of jobs a user can submit.

max_processors = *number*
where *number* specifies the maximum number of processors a user submitting jobs to this class can request for a parallel job (via the **max_processors** job command file keyword). The default is -1 which means that there is no limit.

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.

For more information, consult the appropriate UNIX documentaion.

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 Chapter 9, “Routing Jobs to NQS Machines” on page 177.

priority = *number*
where *number* can be any integer. If not specified, the default is 50. 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 5: Prioritize the Queue Maintained by the Negotiator” on page 103 for more information about the **ClassSysprio** keyword.

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 2. Types of Limit Keywords

Limit	How It Is Enforced
core_limit	Per process
cpu_limit	Per process
data_limit	Per process
file_limit	Per process
job_cpu_limit	Per job step
rss_limit	Per process
stack_limit	Per process
wall_clock_limit	Per job step

Note: Only **job_cpu_limit** and **wall_clock_limit** are supported on the HP platform.

Individual keywords are described in “Specifying Limits in the Class Stanza” on page 91. The following section gives you a general overview of limits.

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

Limits 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**. This is the total amount of CPU that the entire serial job step can use. 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.

Limits include both hard limits and soft limits. A **hard limit** is the higher resource limit and a **soft limit** is the lower resource limit.

You can specify limits in either the class stanza of the administration file or in the job command file. For a per process limit, the limit you set in the administration file overrides the system limit (also called the machine limit).

Exceeding Limits: Process limits are enforced by the operating system. Job step limits are enforced by LoadLeveler.

Exceeding Job Step Limits: When a hard limit is exceeded LoadLeveler sends a *non-trappable* signal to the process (except in the case of a parallel job). When a soft limit is exceeded, LoadLeveler sends a *trappable* signal to the process. The following chart summarizes the actions that occur when a job step limit is exceeded:

Table 3. Exceeding Job Step Limits

Type of Job	When a Soft Limit is Exceeded	When a Hard Limit is Exceeded
Serial	SIGXCPU or SIGKILL issued	SIGKILL issued
Parallel (non-PVM)	SIGXCPU issued to both the user program and to the parallel daemon	SIGTERM issued
PVM	SIGXCPU issued to the user prgram	**pvm_halt** invoked to shut down PVM

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

Exceeding 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.

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

Syntax: The syntax for setting a limit is

```
limit_type = hardlimit,softlimit
```

For example:

```
core_limit = 120kb,100kb
```

To specify only a hard limit, you can enter, for example:

```
core_limit = 120kb
```

To specify only a soft limit, you can enter, for example:

```
core_limit = ,100kb
```

In a keyword statement, you cannot any blanks between the numerical value (100 in the above example) and the units (kb). Also, you cannot have any blanks to the left or right of the comma when you define a limit in a job command file.

For limit keywords that refer to a data limit — such as **data_limit**, **core_limit**, **file_limit**, **stack_limit**, and **rss_limit** — the hard limit and the soft limit are expressed as:

integer[.fraction][units]

where *integer* and *fraction* represent numerical strings of up to eight characters. *units* can be:

b bytes
w words
kb kilobytes (2^{10} bytes)
kw kilowords (2^{10} words)
mb megabytes (2^{20} bytes)
mw megawords (2^{20} words)
gb gigabytes (2^{30} bytes)

If no units are specified, bytes are assumed.

For limit keywords that refer to a time limit — such as **cpu_limit**, **job_cpu_limit**, and **wall_clock_limit** — the hard limit and the soft limit are expressed as:

[hours][:minutes][:seconds][.fraction]

Fractions are rounded to seconds.

You can use the following character strings with all limit keywords:

rlim_infinity Represents the largest positive number.
unlimited Has same effect as **rlim_infinity**.
copy Uses the limit currently active when the job is submitted.

See Table 4 for more information on specifying limits.

Table 4. Setting limits

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

Specifying Limits in the Class Stanza: You can specify the following limit keywords:

core_limit = *hardlimit,softlimit*
Specifies the maximum size of a core file.

Examples:

```
core_limit = unlimited
core_limit = 30mb
```

For more information, see "Overview of Limits" on page 88.

cpu_limit = *hardlimit,softlimit*
Specifies the maximum CPU time to be used by each individual process of a job step. For example, if you impose a **cpu_limit** of five hours and you have a job step composed of five processes, each process can consume five CPU hours; the entire job step can therefore consume 25 total hours of CPU.

Examples:

```
cpu_limit = 12:56:21       # hardlimit = 12 hours 56 minutes 21 seconds
cpu_limit = 56:00,50:00    # hardlimit = 56 minutes 0 seconds
                           # softlimit = 50 minutes 0 seconds
cpu_limit = 1:03           # hardlimit = 1 minute 3 seconds
cpu_limit = unlimited      # hardlimit = 2,147,483,647 seconds
                           # (X'7FFFFFFF')
cpu_limit = rlim_infinity  # hardlimit = 2,147,483,647 seconds
                           # (X'7FFFFFFF')
cpu_limit = copy           # current CPU hardlimit value on the
                           # submitting machine.
```

For more information, see "Overview of Limits" on page 88.

data_limit = *hardlimit,softlimit*
Specifies the maximum size data segment to be used by each process of the submitted job.

Examples:

```
data_limit = 125621         # hardlimit = 125621 bytes
data_limit = 5621kb         # hardlimit = 5621 kilobytes
data_limit = 2mb            # hardlimit = 2 megabytes
data_limit = 2.5mw          # hardlimit = 2.5 megawords
data_limit = unlimited      # hardlimit = 2,147,483,647 bytes
                            #             (X'7FFFFFF')
data_limit = rlim_infinity  # hardlimit = 2,147,483,647 bytes
                            #             (X'7FFFFFF')
data_limit = copy           # copy data hardlimit value from submitting
                            # machine.
```

For more information, see "Overview of Limits" on page 88.

file_limit = *hardlimit,softlimit*
Specifies the maximum size of a file. For more information, see "Overview of Limits" on page 88.

job_cpu_limit = *hardlimit,softlimit*
Specifies the maximum total CPU time to be used by all processes of a job step. That is, if a job step forks to produce multiple processes, the sum total of CPU consumed by all of the processes is added and controlled by this limit.

For example:

```
job_cpu_limit = 10000
```

For more information on this keyword, see the **JOB_LIMIT_POLICY** keyword in Chapter 7, "Gathering Job Accounting Data" on page 163. For more general information on limits, see "Overview of Limits" on page 88.

rss_limit = *hardlimit,softlimit*
Specifies the maximum resident size. For more information, see "Overview of Limits" on page 88.

stack_limit = *hardlimit,softlimit*
Specifies the maximum size of a stack. For more information, see "Overview of Limits" on page 88.

wall_clock_limit = *hardlimit,softlimit*
Specifies the maximum elapsed time for which a job can run. Note that LoadLeveler uses the time the starter process is executed as the start time of the job. For parallel jobs, the **wall_clock_limit** is enforced only on the master node. 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 occured. This keyword is not supported for NQS jobs. Also, if the startd daemon terminates abnormally with running jobs, any wall clock limits are not supported when the daemon is restarted.

For more information, see "Overview of Limits" on page 88.

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
        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=04:00:00     # 4 hour run time limit
        data_limit=80mb,60mb   # max 80 MB data segment
                               # min 60 MB data segment
        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=unlimited     # no run time limit
        data_limit=120mb        # max 120 MB data segment
        core_limit=30mb         # max 30 MB core file
        file_limit=120mb        # max file size 120 MB
        stack_limit=unlimited   # unlimited stack size
        rss_limit=150mb         # max resident set size 150 MB
        job_cpu_limit = 3600,2700  # hard limit 60 minutes
                                   # soft limit 45 minutes
        wall_clock_limit=12:00:00,11:59:55 # hard limit is 12 hours
```

Example 5: Creating a Class to Route Jobs to NQS Machines:

```
nqs:    type=class               # class for NQS jobs
        NQS_class=true
        NQS_submit=pipe_queue    # NQS pipe queue name
        NQS_query=one two three  # list of queue names
```

You can use the class names in control expressions in both the global and local configuration file.

Example 6: Creating a Class for PVM Jobs:

```
PVM3:   type=class              # class for PVM jobs
        priority=60             # ClassSysprio (max=100)
        max_processors=15       # maximum number of processors
```

Example 7: Creating a Class for Master Node Machines:

```
sp-6hr-sp:  type=class               # class for master node machines
            priority=50              # ClassSysprio (max=100)
            cpu_limit = 06:00:00     # 6 hour limit
            job_cpu_limit = 06:00:00 # hard limit is 6 hours
            core_limit = lmb         # max 1MB core file
            master_node_requirement = true # master node definition
```

Step 4: Specify Group Stanzas

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

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

```
label: type = group
    admin = list
    exclude_users = list
    include_users = list
    maxidle = number
    maxjobs = number
    maxqueued = number
    max_processors = number
    priority = number
```

Figure 29. Format of a Group Stanza

You can specify the following keywords in a group stanza:

admin = *list*
where *list* is a blank-delimited list of administrators for this group. These administrators can hold, release, and cancel jobs submitted by users in the group.

exclude_users =*list*
where *list* is a blank-delimited list of users that do not belong to the group. Do not specify both a list of included users and a list of excluded users. Only one of these may be used for any group. The default is that no users will be excluded.

include_users =*list*
where *list* is a blank-delimited list of users that belong to the group. If provided, this list limits users of that group to those on the list. Do not specify both a list of included users and a list of excluded users. Only one of these may be used for any group. The default is that all users will be included.

You can associate a UNIX group with a LoadLeveler group of the same name by specifying **include_users = Unix_Group**. For example, suppose the users Tim and John are both in the UNIX group "chemistry" as defined in the /**etc**/**group** file. The example below includes Tim and John in the LoadLeveler group "chemistry":

```
chemistry:   type = group
             include_users = Unix_Group
```

maxidle = *number*
where *number* is the maximum number of idle jobs this group can have in queue. That is, *number* is the maximum number of jobs which the negotiator will consider for dispatch for this group. Jobs above this maximum are placed in the NotQueued state. This prevents groups from flooding the job queue. If the group stanza does not specify **maxidle** or if there is no group stanza at all, the maximum number of jobs that can be simultaneously in queue for the group is defined in the default stanza. 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 group.

For more information, see "Controlling the Mix of Idle and Running Jobs" on page 295.

maxjobs = *number*
where *number* is a maximum number of jobs this group can run at any time. If the group stanza does not specify the **maxjobs** or if there is no group stanza at

all, the maximum number of jobs that can be simultaneously run the group is defined in the default stanza. 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 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 295.

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 **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. If no value is found, or the limit found is -1, then 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 295.

max_processors = *number*
where *number* specifies the maximum number of processors a user specifying this group can request for a parallel job in their job command file (via the **max_processors** job command file keyword). The default is -1, which means there is no limit.

priority = *number*
where *number* can be any integer. 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 50. 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 here in the administration file have no effect. See "Step 5: Prioritize the Queue Maintained by the Negotiator" on page 103 for more information about the **GroupSysprio** keyword.

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 **group_a** has five members and these user's jobs have a priority of 100:

```
# Define group stanzas
group_a:  type = group
          priority = 100
          include_users = bob sally joe mary kathy
```

Configuring LoadLeveler

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

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

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

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

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

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 you have your configuration files on this ID.

You can also override the /**etc**/**LoadL.cfg** file. For an example of when you might want to do this, see “Querying Multiple LoadLeveler Clusters” on page 27.

The Configuration Files

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

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

- *Global Configuration File:* This file by default is called the **LoadL_config** file and it contains configuration information common to all nodes in the LoadLeveler cluster.
- *Local Configuration File:* This file is generally called **LoadL_config.local** (although it is possible for you to rename it). This file contains specific configuration information for an individual node. The **LoadL_config.local** file is in the same format as **LoadL_config** and the information in this file overrides any information specified in **LoadL_config**. It is an optional file that you use to modify information on a local machine. Its full pathname is specified in the **LoadL_config** file by using the **LOCAL_CONFIG** keyword. See “Step 9: Specify Where Files and Directories are Located” on page 108 for more

information. "Customizing the Global and Local Configuration Files" on page 97 describes how to tailor this file to suit your needs. Appendix C, "Sample Files" on page 313 contains a sample 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 "Customizing the Global and Local Configuration Files" and in "Additional Configuration File Keywords" on page 119.
- User-defined keywords, described in "User-Defined Keywords" on page 121.
- Reserved keywords (or macros), described in "Macros" on page 123

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 (\).

Customizing the Global and Local Configuration Files

This section presents a step-by-step approach to configuring LoadLeveler. You do not have to perform the steps in the order that they appear here. Other keywords which are not specifically mentioned in any of these steps are discussed in "Additional Configuration File Keywords" on page 119.

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**. They can also invoke the administrator-only GUI functions. For more information, see "Administrative Uses for the Graphical User Interface" on page 250.

LoadLeveler administrators also receive mail describing problems that are encountered by the master daemon.

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

For example, to grant administrative authority to users `bob` and `mary`, enter the following in the configuration file:

```
LOADL_ADMIN = bob mary
```

Step 2: 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 conditions are defined, following normal C conventions, in the following expressions:

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)*
Determines whether a machine can run a LoadLeveler job. When the expression evaluates to **T**, LoadLeveler considers dispatching a job to the machine.

When you use a START expression that is based on the CPU load average, the negotiator may evaluate the expression as **F** even though the load average indicates the machine is Idle. This is because the negotiator adds a compensating factor to the startd machine's load average every time the negotiator assigns a job. For more information, see the NEGOTIATOR_LOADAVG_INCREMENT keyword on page 119.

SUSPEND: *expression that evaluates to T or F (true or false)*
Determines whether running jobs should be suspended. When **T**, LoadLeveler temporarily suspends jobs currently running on the machine. Suspended LoadLeveler jobs will either be continued or vacated. This keyword is not supported for parallel jobs.

CONTINUE: *expression that evaluates to T or F (true or false)*
Determines whether suspended jobs should continue execution. When **T**, suspended LoadLeveler jobs resume execution on the machine.

VACATE: *expression that evaluates to T or F (true or false)*
Determines whether suspended jobs should be vacated. When **T**, suspended LoadLeveler jobs are removed from the machine and placed back into the queue (provided you specify **restart=yes** in the job command file). If a

checkpoint was taken, the job restarts from the checkpoint. Otherwise, the job restarts from the beginning.

KILL: *expression that evaluates to T or F (true or false)*
Determines whether or not vacated jobs should be killed 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 30 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.

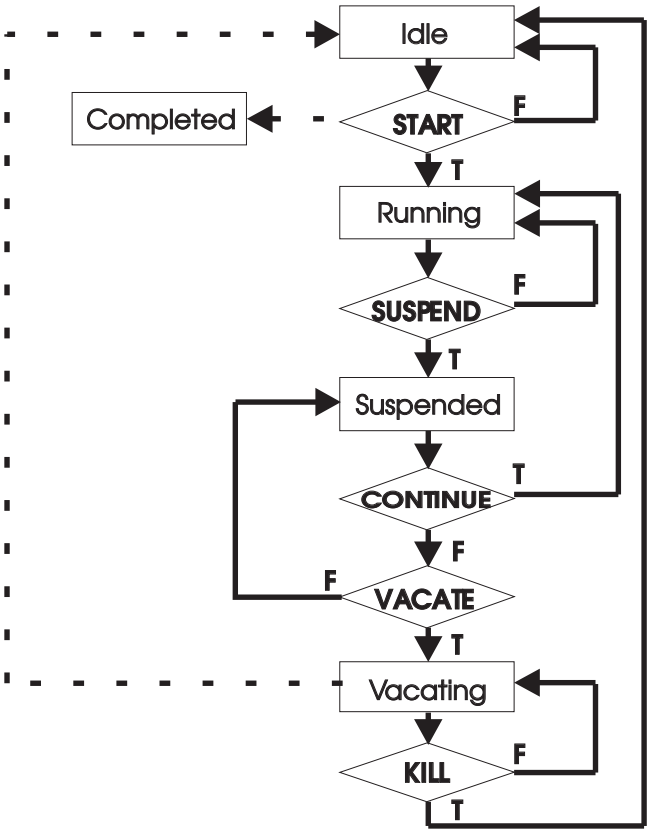


Figure 30. How Control Expressions Affect Jobs

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

Step 3: Define LoadLeveler Machine Characteristics

You can use the following keywords to define the characteristics of machines in the LoadLeveler cluster:

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 RISC System/6000, the keyword would look like:

```
ARCH = R6000
```

To define a machine as being a Sun SPARCstation, the macro would look like:

```
ARCH = SPARC
```

Class = {" *class1***" "***class2***" ...} | { "No_Class" }**

where "*class1*" "*class2*" *...* is a blank delimited list of class names. This keyword determines whether a machine will accept jobs of a certain job class.

You need to specify **default_class** in the default user stanza of the administration file to set a default class. If you don't, an error will be generated.

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, can be identified in the administration file. See “Setting Up a Single Machine To Have Multiple Job Classes” on page 296 and “Step 3: Specify Class Stanzas” on page 86 for more information on classes.

Defining Classes – Examples

Example 1: This example defines the default class:

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

This is the default. The machine will only run one LoadLeveler job at a time that has either defaulted to, or explicitly requested class **No_Class**. A LoadLeveler job with class **CPU_bound**, for example, would not be eligible to run here. Only one LoadLeveler job at a time will run on the machine.

Example 2: This example specifies multiple classes. 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 4: Specify How Many Jobs a Machine Can Run” on page 102 for more information on **MAX_STARTERS**.

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

Example 3: This example specifies multiple classes. 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.

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

Example 4: This example specifies multiple classes. 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.

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

Example 5: This example specifies multiple classes including PVM jobs:

```
Class = { "No_Class" "ClassA" "ClassA" "PVM3" }
```

CUSTOM_METRIC = *number*
Specifies a machine's relative priority to run jobs. This is an an arbitrary number which you can use in the MACHPRIO expression. If you specify neither **CUSTOM_METRIC** nor **CUSTOM_METRIC_COMMAND**, **CUSTOM_METRIC = 1** is assumed. For more information, see “Step 6: Prioritize the Order of Executing Machines Maintained by the Negotiator” on page 104.

CUSTOM_METRIC_COMMAND = *command*
Specifies an executable and any required arguments. The exit code of this command is assigned to **CUSTOM_METRIC**. If this command does not exit normally, **CUSTOM_METRIC** is assigned a value of 1. This command is forked every (**POLLING_FREQUENCY** * **POLLS_PER_UPDATE**) period. For more information, refer to Chapter 6, “Using Interactive LoadLeveler” on page 137.

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 enter the following:

```
Feature = {"abc" "xyz"}
```

When submitting a job that requires both of these products, the user should enter the following in their job command file:

```
requirements = (Feature == "abc") && (Feature == "xyz")
```

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 idenitified 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.

START_DAEMONS = true|false
Specifies whether to start the LoadLeveler daemons on the node. When **true**, the daemons are started.

In most cases, you will probably want to set this keyword to **true**. An example of why this keyword would be set to **false** is if you want to run the daemons on most of the machines in the cluster but some individual users with their own local configuration files do not want their machines to run the daemons. The individual users would modify their local configuration files and set this keyword to **false**. Because the global configuration file has the keyword set to **true**, their individual machines would still be able to participate in the LoadLeveler cluster.

Also, to define the machine as strictly a submit-only machine, set this keyword to **false**. For more information, see the submit-only section on page 82.

SCHEDD_RUNS_HERE = true|false
Specifies whether the schedd daemon runs on the host. If you do not want to run the schedd daemon, specify **false**.

To define the machine as an executing machine only, set this keyword to **false**. For more information, see the submit-only section on page 82.

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 (unless you are running on Sun machine or an HP machine).

Step 4: 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 100.

The syntax for **MAX_STARTERS** is:

MAX_STARTERS = *number*
where *number* specifies the maximum number of job steps that can run simultaneously on a machine. If not specified, the default is the number of elements in the **Class** statement. For parallel jobs, **MAX_STARTERS** is the number of jobs that can be initiated from a **startd**.

For example, if the configuration file contains these statements:

```
Class = { "A" "B" "B" "C"}
MAX_STARTERS = 2
```

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" }
```

then the maximum number of jobs 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.

If this 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 5: 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. 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 LoadLeveler 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 29 for more information on job priorities.

You can use the following keywords to define the **SYSPRIO** expression:

ClassSysprio The priority for the class of the job, defined in the class stanza in the administration file. The default is 0.

GroupQueuedJobs The number of jobs associated with a LoadLeveler group which are either running or queued. (That is, jobs which are in one of these states: Running, Starting, Pending, or Idle.)

GroupRunningJobs The number of jobs for the LoadLeveler group which are in one of these states: Running, Starting, or Pending.

GroupSysprio The priority for the group of the job, defined in the group stanza in the administration file. The default is 0.

GroupTotalJobs The total number of jobs associated with this LoadLeveler group. Total jobs are all jobs reported by the **llq** command.

QDate	The difference in the UNIX date when the job enters the queue and the UNIX date when LoadL_negotiator starts up.
UserPrio	The user-defined priority of the job, specified in the job command file with the **user_priority** keyword. The default is 50.
UserQueuedJobs	The number of jobs either running or queued for the user. (That is, jobs which are in one of these states: Running, Starting, Pending, or Idle.)
UserRunningJobs	The number of jobs for the user which are in one of these states: Running, Starting, or Pending.
UserSysprio	The priority of the user who submitted the job, defined in the user stanza in the administration file. The default is 0.
UserTotalJobs	The total number of jobs associated with this user. Total jobs are all jobs reported by the **llq** command.

Usage Notes for the SYSPRIO Keyword

- It is recommended that you do not use **UserPrio** in the **SYSPRIO** expression, since user jobs are already ordered by **UserPrio**.
- You can use the **UserRunningJobs**, **GroupRunningJobs**, **UserQueuedJobs**, **GroupQueuedJobs**, **UserQueuedJobs**, **GroupQueuedJobs UserTotalJobs**, and **GroupTotalJobs** parameters to prioritize the queue based on current usage. You should also set **NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL** so that the priorities are adjusted according to current usage rather than usage only at submission time.

Using the SYSPRIO Keyword – Examples

Example 1: This example creates a FIFO job queue based on submission time:

```
SYSPRIO : 0 - (QDate)
```

Example 2: This example accounts for Class, User, and Group system priorities:

```
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 6: 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. 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.

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_LOADAVG_INCREMENT keyword on page 119.

You can use the following keywords in the **MACHPRIO** expression:

LoadAvg The Berkeley one-minute load average of the machine, reported by startd.

Cpus The number of active processors of the machine, reported by startd.

Speed The relative speed of the machine, defined in a machine stanza in the administration file. The default is 1.

Memory The size of real memory in megabytes of the machine, reported by startd.

VirtualMemory The size of available swap space in kilobytes of the machine, reported by startd.

Disk The size of free disk space in kilobytes on the filesystem where the executables reside.

CustomMetric Allows the administrator to control the ordering of machines. (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.

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 allows the administrator control over machine ordering:

```
MACHPRIO : CustomMetric
```

Also, you need to 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 **LoadL_config** file. For example:

```
CUSTOM_METRIC = 5
```

You can override this value for an individual machine by specifying a different value in that machine's **LoadL_config.local** file.

Example 5: This example gives master nodes the highest priority:

```
MACHPRIO : (MasterMachPriority * 10000)
```

Step 7: Define Job Accounting

LoadLeveler provides accounting information on completed LoadLeveler jobs. For detailed information on this function, refer to Chapter 7, “Gathering Job Accounting Data” on page 163.

The accounting macro allows you to turn accounting on and off:

ACCT = *flag* **(optional)**
The available flags are:

A_ON	Turns accounting data recording on. If specified without the **A_DETAIL** flag, total resource consumption for each job step is recorded.
A_OFF	Turns accounting data recording off. This is the default.
A_VALIDATE	Turns accounting 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 = *$(RELEASEDIR)/llacctval* **(optional)**
Keyword used to identify the module that will be called to perform account validation. An installation may choose to replace the **llacctval** module with its own accounting routine by specifying the module 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.

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
ACCT_VALIDATION     = $(BIN)/llacctval
GLOBAL_HISTORY      = $(SPOOL)
```

Step 8: Specify Alternate Central Managers

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

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

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

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

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

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:

```
CENTRAL_MANAGER_HEARTBEAT_INTERVAL = 45    # 45 seconds
CENTRAL_MANAGER_TIMEOUT = 30               # 30 * 45 seconds
```

For more information on central manager backup, refer to “What Happens if the Central Manager Isn't Operating?” on page 290.

Step 9: Specify Where Files and Directories are Located

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

To specify the location of the:	Specify these keywords:
Administration File	**ADMIN_FILE =** *pathname* **(required)** points to the administration file containing user, class, group, and machine stanzas. For example, `ADMIN_FILE = $(tilde)/admin_file`

Local Configuration File	**LOCAL_CONFIG =** *pathname*
	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:
	`LOCAL_CONFIG = $(tilde)/$(host).LoadL_config.local`
	`LOCAL_CONFIG = $(tilde)/LoadL_config.$(host).$(domain)`
	`LOCAL_CONFIG = $(tilde)/LoadL_config.local.$(hostname)`
	If you are using a local file system, an example is:
	`LOCAL_CONFIG = /var/LoadL/LoadL_config.local`
	See “Reserved Macros” on page 123 for information about the **tilde**, **host**, and **domain** macros.

Local Directory	The following subdirectories reside in the local directory. It is possible that the local directory and LoadLeveler's home directory are the same. **EXECUTE =** *local directory*/**execute (required)** defines the local directory to store the executables of jobs submitted by other machines. **LOG =** *local directory*/**log (required)** 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. **SPOOL =** *local directory*/**spool (required)** Defines the local directory where LoadLeveler keeps the local job queue and checkpoint files, as well as: **HISTORY = $(SPOOL)/history (required)** defines the pathname where a file containing the history of local LoadLeveler jobs is kept.
Release Directory	**RELEASEDIR =** *release directory* **(required)** 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. **BIN = $(RELEASEDIR)/bin (required)** defines the directory where LoadLeveler binaries are kept. **LIB = $(RELEASEDIR)/lib (required)** defines the directory where LoadLeveler libraries are kept. **NQS_DIR =** *NQS directory (optional)* defines the directory where NQS commands **qsub**, **qstat**, and **qdel** reside. The default is **/usr/bin**.

Step 10: Record and Control Log Files

The LoadLeveler daemons and processes keep log files according to the specifications in the configuration file. A number of macros are used to describe where LoadLeveler maintains the logs and how much information is recorded in each log. These macros, shown in Table 5, 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 111 describes the events that can be reported through logging controls.

Table 5. Log Control Statements

Daemon/ Process	**Log File** *(required)* **(See note 1)**	**Max Length** *(required)* **(See note 2)**	**Debug Control** *(required)* **(See note 4)**
Master	**MASTER_LOG =** *path*	**MAX_MASTER_LOG =** *bytes*	**MASTER_DEBUG =** *flags*
Schedd	**SCHEDD_LOG =** *path*	**MAX_SCHEDD_LOG =** *bytes*	**SCHEDD_DEBUG =** *flags*
Startd	**STARTD_LOG =** *path*	**MAX_STARTD_LOG =** *bytes*	**STARTD_DEBUG =** *flags*
Starter	**STARTER_LOG** = *path*	**MAX_STARTER_LOG =** *bytes*	**STARTER_DEBUG =** *flags*
Negotiator	**NEGOTIATOR_LOG =** *path*	**MAX_NEGOTIATOR_LOG =** *bytes*	**NEGOTIATOR_DEBUG =** *flags*
Kbdd	**KBDD_LOG =** *path*	**MAX_KBDD_LOG =** *bytes*	**KBDD_DEBUG =** *flags*

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.

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

   TRUNC_MASTER_LOG_ON_OPEN = true|**false**
   TRUNC_STARTD_LOG_ON_OPEN = true|**false**
   TRUNC_SCHEDD_LOG_ON_OPEN = true|**false**
   TRUNC_KBDD_LOG_ON_OPEN = true|**false**
   TRUNC_STARTER_LOG_ON_OPEN = true|**false**
   TRUNC_NEGOTIATOR_LOG_ON_OPEN = true|**false**
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 111. 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_CKPT Logs various steps in the checkpointing process. Logs calls to read and write by the xdr routines.

D_DAEMON Logs information regarding basic daemon set up and operation, including information on the communication between daemons.

D_DBX Bypasses certain signal settings to permit debugging of the processes as they execute in certain critical regions.

D_EXPR Logs steps in parsing and evaluating control expressions.

D_FULLDEBUG Logs details about most actions performed by each daemon but doesn't log as much activity as setting all the flags.

D_MACHINE Logs machine control functions and variables when making decisions regarding starting, suspending, resuming, and aborting remote jobs.

D_NQS Provides more information regarding the processing of NQS files.

D_STANZAS Displays internal information about the parsing of the administration file.

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

D_JOB Logs job requirements and preferences when making decisions regarding whether a particular job should run on a particular machine.

D_PROC Logs information about jobs being started remotely such as the number of bytes fetched and stored for each job.

D_NEGOTIATE Displays the process of looking for a job to run in the negotiator. It only pertains to this daemon.

D_SCHEDD Displays how the schedd works internally.

D_STARTD Displays how the startd works internally.

D_STARTER Displays how the starter works internally.

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.

Step 11: 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 confiuration 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.

MASTER_STREAM_PORT = *port number*
The default is 9616.

NEGOTIATOR_STREAM_PORT = *port number*
The default is 9614.

SCHEDD_STREAM_PORT = *port number*
The default is 9605.

STARTD_STREAM_PORT = *port number*
The default is 9611.

COLLECTOR_DGRAM_PORT = *port number*
The default is 9613. This keyword is used by the negotiator daemon.

STARTD_DGRAM_PORT = *port number*
The default is 9615.

MASTER_DGRAM_PORT = *port number*
The default is 9617.

As stated earlier, if LoadLeveler does not find the value in the configuration file, it looks in the /**etc**/**services** file. If the value is not found in this file, the default is used. The following is an example of this file illustrating the port numbers:

```
LoadL_master     9616/tcp   # Master port number for stream port
LoadL_negotiator 9614/tcp   # Negotiator port number
LoadL_schedd     9605/tcp   # Schedd port number for stream port
LoadL_startd     9611/tcp   # Startd port number for stream port
LoadL_negotiator 9613/udp   # Negotiator port number for dgram port
LoadL_startd     9615/udp   # Startd port number for dgram port
LoadL_master     9617/udp   # Master port number for dgram port
```

Step 12: Write Prolog and Epilog Programs

An administrator can write **prolog** and **epilog** programs that can run before and after a LoadLeveler job runs, respectively. The prolog and epilog programs enable you to perform site specific setup and clean up activities for each job started by LoadLeveler.

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.

You can specify prolog and epilog programs, using 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.

For security reasons, the **JOB_USER_PROLOG** and **JOB_USER_EPILOG** keywords must be coded on the machine where the jobs run and on the machine from which the jobs were submitted.

For more information on writing prolog and epilog programs, refer to "Prolog and Epilog APIs" on page 259.

Step 13: Enable Checkpointing

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. Users can checkpoint their serial jobs. They cannot, however, checkpoint their parallel jobs.

At checkpoint time, a checkpoint file is created on the executing machine. The checkpoint file contains the text data, stack segments, register contents, and the status of the open files at the time of the checkpoint.

You can submit checkpointing jobs only to machines of the same architecture. For example, you can submit checkpointing jobs from a RISC System/6000 to RISC System/6000, from an SGI to another SGI machine, or from an HP to another HP machine.

When a job is vacated from a machine, a checkpoint does not automatically occur. Instead, the most recent checkpoint file taken before the job was vacated is stored on disk by the scheduling machine to be rescheduled to another available machine.

If the executing machine fails, when the machine restarts, LoadLeveler reads the checkpoint file and reschedules the job to be executed on an available machine. LoadLeveler waits for the executing machine to restart before scheduling the job to run on another machine in order to ensure that only one copy of the job will run.

The LoadLeveler administrator can manually force jobs enabled for checkpointing on a machine to be checkpointed and vacated using the **llctl** flush command. Note that no new jobs are redispatched to this host until the **resume** option is issued on the **llctl** command. For more information on this command, refer to "llctl - Control LoadLeveler Daemons" on page 193.

Guidelines to Follow When Checkpointing Jobs

Review the following guidelines before submitting a checkpointing job:

Reserve Adequate Disk Space in the Execute Directory: A checkpointed job uses a lot of disk space so unless there is enough disk space in the directory, the job may fail.

At checkpoint time, a core dump of the application is created. A new user executable is created from the original executable and the core file. The original executable may be small in size but the new executable can be much larger. This is because the new executable is made up from the original one and from sections of the core file.

Edit Your Job Command File: Add the following line to your job command file to ensure that the core file is not truncated:

```
# @ core_limit = unlimited,unlimited
```

Checkpoint Jobs Whose States are Simple to Checkpoint and Recreate: For some processes, it is impossible to obtain or recreate the state of the process. For this reason, users should only checkpoint jobs whose states are simple enough to checkpoint and recreate. A job that is long-running, computation-intensive, and runs as a single process, is an example of a job well suited for checkpointing.

Avoid Using Certain System Calls in Checkpointed Jobs: In order to prevent unpredictable results from occurring, checkpointing jobs should not use the following system calls:

- Administrative (for example, audit and swapqry)
- Signals
- Forks
- Dynamic loading
- Shared memory
- Semaphores
- Messages
- Internal timers
- Set user/group id
- Threads

Another limitation of checkpointing jobs is file I/O. Since individual write calls are not traced, the file recovery scheme requires that all I/O operations, when repeated, must yield the same result. A job that opens all files as read only can be checkpointed. A job that writes to a file and then reads the data back may also be checkpointed. An example of I/O that could cause unpredictable results is reading, writing, and then reading again the same area of a file.

Choose the Host in Your Cluster On Which You Perform the Compilation of Your Checkpointable Executable: If different versions of the operation system are running on the hosts of your cluster, perform your compilation on the host(s) with the oldest version of the operation system. Runtime environments are upward compatible but they may not be downward compatible.

How to Checkpoint a Job

There are several ways to checkpoint a job. To determine which type of checkpointing you want to specify, refer to the following table:

To specify:	Do this:
Exactly when LoadLeveler will checkpoint your job.	Add the following option to your job command file: **checkpoint = user_initiated** You can also select this option on the Build a Job window of the GUI. To do this, code the LoadLeveler **ckpt()** function call in the FORTRAN or C program. Whenever this call is invoked, a checkpoint of the program will be taken.
That LoadLeveler automatically checkpoint your job.	Add the following option to your job command file: **checkpoint = system_initiated** You can also select this option on the Build a Job window of the GUI. For this type of checkpointing to work, system administrators must set certain frequencies in the configuration file to specify how often LoadLeveler would take a checkpoint of the job. The two frequencies are set with the following two keywords: **MIN_CKPT_INTERVAL =** *number* **MAX_CKPT_INTERVAL =** *number* where *number* specifies a 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. The system administrator can set these keyword values globally in the global configuration file so that all machines in the cluster have the same value, or he can specify a different value for each machine by modifying the local configuration files. **Note:** If the job command files you used with previous releases of LoadLeveler state **checkpoint=yes**, they will default to **checkpoint=system_initiated**. **Note:** If you want to have both user initiated and system initiated checkpoint, you would select **system_initiated** as well. In this case, you have to code the **ckpt** function call and specify the keywords.
That your job not be checkpointed	Add the following option to your job command file: **checkpoint = no** You can also select this option on the Build a Job window of the GUI. This option is the default.

Ensure all User's Jobs are Linked to Checkpointing Libraries

All checkpointing jobs must be linked with the LoadLeveler libraries **libload.a** and **libckpt.a**. LoadLeveler checks to make sure the job is linked correctly.

Checkpointing for the RISC System/6000: To support checkpointing, LoadLeveler has altered the UNIX system call mechanism by providing a special version of the C library (**libc.a**). Therefore, you must link the program with the LoadLeveler libraries. Linking instructions vary slightly from language to language. Jobs that run without checkpoint do not require re-linking. In AIX, your LoadLeveler administrator, as part of the installation process, should have activated the following commands to allow you to do the compile/link:

llcc (for use with C)

llxlf (for use with FORTRAN)

C Example

```
llcc -o MYPROG.ckpt MYPROG.c
```

FORTRAN Example

```
llxlf -o MYPROG.ckpt MYPROG.f
```

Checkpointing for the SunOS: When submitting a checkpoint job to a SunOS machine, link your programs with LoadLeveler's version of crt0.o, and LoadLeveler's libckpt.a and libloadl.a. You can find these in the **/usr/lpp/LoadL/sunosnfs/lib** directory. Since the UNIX compilers automatically link in crt0.o, you must link your program by calling ld directly, substituting "loadl_rt0.o" for "crt0.o". You must also tell the linker to use static linking, since LoadLeveler does not support the use of shared libraries.

C Example

```
/usr/lang/acc -c MYPROG.ckpt MYPROG.c

/bin/ld -dc -dp -e start -X -o MYPROG_SUN -Bstatic
/u/loadl/lib/loadl_rt0.o  \
/usr/lang/SC2.0.1/cg87/_crt1.o \
/usr/lang/SC2.0.1/values-Xt.o \
/u/loadl/lib/libckpt.a \
/u/loadl/lib/libloadl.a \
/usr/lib/libresolv.a \
-L/usr/lang/SC2.0.1/cg87 \
-L/usr/lang/SC2.0.1 \
-u _fix_libc_ MYPROG.ckpt -lm -lansi -lc
```

FORTRAN Example

```
ld -o a.out -dc -dp -e start -Bstatic -u _MAIN_ -u _main -X
/usr/lpp/LoadL/sunosnfs/lib/loadl_rt0.o /usr/lang/SC1.0/cg87/_crt1.o
-L/usr/lang/SC1.0/cg87 -L/usr/lang/SC1.0 a.o b.o c.o -lF77 -lm
/usr/lpp/LoadL/sunosnfs/lib/libckpt.a
/usr/lpp/LoadL/sunosnfs/lib/libloadl.a -lc
```

Checkpointing for the SGI: When checkpointing jobs to an SGI machine, link your programs with LoadLeveler's version of crt1.o (loadl_rt0.o) and LoadLevler's libckpt.a and libloadl.a. You can find these in the directory **/usr/lpp/LoadL/sginfs/lib** Since IRIX compilers automatically link in crt1.o, you must link your programs by calling ld directly, substituting "loadl_rt0.o" for "crt1.o".

C Example

Compiling:

```
cc -c test01.c -non_shared
```

Linking:

```
LL_LIBDIR=/usr/lpp/LoadL/sginfs/lib export LL_LIBDIR /usr/bin/ld -elf -o test01
-_SYSTYPE_SVR4 -Wx,-G 8 -non_shared -g0
 -nocount $LL_LIBDIR/loadl_rt0.o -count test01.o
 -nocount $LL_LIBDIR/libckpt.a
 -nocount -L/usr/lib/nonshared -lC -lm -lc
  /usr/lib/nonshared/crtn.o
```

FORTRAN Example

Compiling:

```
f77 -c test01.f
```

Linking:

```
/usr/bin/ld -o test01 -Xnobsschange -G 8 -g0 -nocount
/usr/lpp/LoadL/sginfs/lib/loadl_rt0.o -count test01.o -nocount
/usr/lpp/LoadL/sginfs/lib/libckpt.a /usr/lpp/LoadL/sginfs/lib/libloadl.a -lF77 -lm -lU77
-lI77 -lisam -lmpc -lrpcsvc -lsun -lc /usr/lib/crtn.o
```

Checkpointing for the HP: When checkpointing jobs to an HP-UX system, link your programs with LoadLeveler's version of crt0.o (loadl_rt0.o) and LoadLevler's libckpt.a and libloadl.a. You can find these in the directory /usr/lpp/LoadL/hpnfs/lib. Since HP-UX compilers automatically link in crt0.o, you must link your programs by calling ld directly, substituting "loadl_rt0.o" for "crt0.o".

C Example

Compiling:

```
cc -c test01.c
```

Linking:

```
/bin/ld -o test01 /usr/lpp/LoadL/hpnfs/lib/loadl_rt0.o test01.o -a archive
/usr/lpp/LoadL/hpnfs/lib/libckpt.a /usr/lpp/LoadL/hpnfs/lib/libloadl.a /usr/lib/libBSD.a
-lm -lc
```

FORTRAN Example

Compiling:

f77 -c test01.f

Linking:

/bin/ld -o test01 /usr/lpp/LoadL/hpnfs/lib/loadl_rt0.o test01.o -a archive
/usr/lpp/LoadL/hpnfs/lib/libckpt.a /usr/lpp/LoadL/hpnfs/lib/libloadl.a /usr/lib/libBSD.a
-lcl -lisamstub -lc

Additional Configuration File Keywords

This section describes keywords that were not previously mentioned in the configuration steps described in “Customizing the Global and Local Configuration Files” on page 97. In general, you should be able to use these keywords 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 | CANCEL
where HOLD specifies that jobs are placed in User Hold status when their rejection count has reached the value of the **MAX_JOB_REJECT** keyword. CANCEL specifies that jobs are canceled when their rejection count has reached the value of the **MAX_JOB_REJECT** keyword. The default is HOLD. When a job is rejected, LoadLeveler sends a mail message stating why the job was rejected.

AFS_GETNEWTOKEN = *programname*
where *programname* 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 280.

KBD_DEVICE = *filename*
where *filename* specifies the keyboard device file. This keyword only applies to SunOS machines and is used by the startd daemon when monitoring X events. The directory /**dev** is assumed. The default is **kbd**.

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.

MAX_JOB_REJECT = *number*
where *number* specifies the number of times a job can be rejected before it is removed (cancelled) or put in User Hold status. That is, a rejected job is redispatched until the **MAX_JOB_REJECT** value is reached. The default is -1, meaning a job is redispatched an unlimited number of times. A job that cannot run for various reasons (such as a **uid** mismatch, unavailable resources, or wrong permissions) on one machine will be rejected on that machine, and LoadLeveler will attempt to run the job on another machine. A value of 0 means that if the job is rejected, it is immediately removed. (For related information, see the **NEGOTIATOR_REJECT_DEFER** keyword in this section.)

MOUSE_DEVICE = *filename*
where *filename* specifies the mouse device file. This keyword only applies to SunOS machines and is used by the startd daemon when monitoring X events. The directory /**dev** is assumed. The default is **mouse**.

NEGOTIATOR_INTERVAL = *number*
where *number* specifies the interval, in seconds, at which the negotiator daemon negotiates with machines that are available to run jobs. This daemon also negotiates with machines whenever job states or machine states change. The default is 60 seconds.

NEGOTIATOR_LOADAVG_INCREMENT = *number*
where *number* specifies the value the negotiator adds to the startd machine's load average whenever a job in the Pending state is queued on that machine. This value is used to compensate for the increased load caused by starting another job. The default value is .5.

NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL = *number*
where *number* specifies the amount of time in seconds between calculation of the **SYSPRIO** values for waiting jobs. A value of 0 (which is the default) means the **SYSPRIO** values are not recalculated. 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.

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 5: Prioritize the Queue Maintained by the Negotiator" on page 103.

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, set internally by LoadLeveler, is **NEGOTIATOR_INTERVAL** multiplied by 3.

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 frequency, 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**. A value of 24 is the default.

PUBLISH_OBITUARIES = true|false
where **true** specifies that the master daemon send mail to the administrator(s) identified by **LOADL_ADMIN** keyword, when any of the daemons it manages dies abnormally.

RESTARTS_PER_HOUR = *number*
where *number* specifies how many times the master daemon attempts to restart a daemon that dies abnormally. Because one or more of the daemons may be unable to run due to a permanent error, the master only attempts **$(RESTARTS_PER_HOUR)** restarts within a 60 minute period. Failing that, it sends mail to the administrator(s) identified by the **LOADL_ADMIN** keyword and exits. The default is 12.

SCHEDD_INTERVAL = *number*
where *number* specifies the interval, in seconds, at which the schedd daemon checks the local job queue and updates the negotiator daemon. A value of 120 should be sufficient.

SCHEDULER_API = YES|**NO**
where **YES** disables the native LoadLeveler scheduling algorithm. Specifying **YES** implies you will use the scheduling API to communicate to LoadLeveler scheduling decisions made by an external scheduler.

Omit this keyword if you do not plan to use an external scheduler. For more information, see “Scheduling API” on page 273.

User-Defined Keywords

This type of keyword, which is generally created and defined by the user, can be named using any combination of letters and numbers. A user-defined keyword is set equal to values, where the *value* defines conditions, names files, or sets numeric values. For example, you can create a keyword 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 ($) and parentheses. For example, the literal **$(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 parentheses, 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*.

LoadLeveler includes the following "user-defined" keywords in the sample configuration files shipped with the product. For examples of using these keywords, see the configuration file in Appendix C, "Sample Files" on page 313.

BackgroundLoad = 0.7
defines the variable **BackgroundLoad** and assigns to it the floating point constant 0.7. This might be used as a noise factor indicating no activity.

CPU_Busy = LoadAvg >= $(HighLoad)
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 $(HighLoad).

CPU_Idle = LoadAvg <= $(BackgroundLoad)
defines the variable **CPU_Idle** and reassigns to it at each evaluation the Boolean value **true** or **false** depending on whether the Berkeley one-minute load average is equal or less than $(BackgroundLoad).

HighLoad = 1.5
defines the variable **HighLoad** and assigns to it the floating point constant 1.5. This keyword can be used as a saturation level at which no further jobs are started, and assumes you are only running one job on the machine.

This value is based upon how many jobs you want the machine to run. Therefore, you might want to set **HighLoad** to:

```
(MAX_STARTERS * $(JobLoad)) + $(BackgroundLoad)
```

HOUR = (60 * $(MINUTE))
defines the variable **HOUR** and assigns to it the constant integer value 360.

JobLoad = 0.8
defines the variable **JobLoad** and assigns to it the floating point constant 0.8. This can be used to define the load on the machine caused by running a job.

KeyboardBusy = KeyboardIdle < (15 * $(MINUTE))
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.

LowLoad = $(BackgroundLoad)
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.

This value is based upon how many jobs you want the machine to run. Therefore, you might want to set **LowLoad** to:

```
(MAX_STARTERS * $(JobLoad)) + $(BackgroundLoad) - 1
```

MINUTE = 60
defines the variable **MINUTE** and assigns to it the constant integer value 60.

StateTimer = (CurrentTime - EnteredCurrentState)
defines the variable **StateTimer** and reassigns to it at each evaluation the number of seconds since the current state was entered.

See the configuration files in Chapter 12, "LoadLeveler Application Programming Interfaces" on page 257 for additional examples.

Macros

LoadLeveler has a set of reserved keywords (or macros) that you can set in the configuration file. Reserved keywords include:

- Numerical and alphabetic constants
- Mathematical operators
- Reserved macros

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 in your control function expressions. The operators are listed in order of precedence. All of these operators are evaluated from left to right:

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

Reserved Macros

Reserved macros 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. For example, the **LoadAvg** macro is represented as **LoadAvg** when used in an expression. LoadLeveler provides the following reserved macros that you can use in your statements:

Arch : $(ARCH)
indicates the system architecture.

AvailableClasses
the job classes currently available on the machine. Note that every machine has available classes as well as supported classes. Available classes are those which are currently available – that is, not in use – on the machine, while the class statement in the configuration file defines all the classes of jobs (both in use and available) supported on the machine. For more information on the class statement, refer to “Step 3: Define LoadLeveler Machine Characteristics” on page 100.

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.

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. For example, if a job is submitted to host machine **silver** and the domain of the submitting machine is **ore.met.com** and the domain for the machine is not specified elsewhere, then **$(domain)** would have the value **ore.met.com**.

EnteredCurrentState
the value of **CurrentTime** when the current state (START, SUSPEND, etc) was entered.

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. For example, if a job is submitted to host machine **gold.kgn.ibm.com**, then **$(host)** returns the value **gold**. **$(hostname)** returns **gold.kgn.ibm.com**.

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
the name of the current machine.

Machine : $(hostname)
indicates the name of the current host. String-valued functions are provided for programs that may want to utilize these control functions to evaluate them.

Memory
the physical memory installed on the machine in megabytes.

OpSys : $(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.

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

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 on the machine in kilobytes.

Time: You can use the following time macros in the START, SUSPEND, CONTINUE, VACATE, and KILL expressions. If you use these macros 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 5PM and 8AM during the month of October, factoring in an adjustment for Daylight Savings Time, you can issue:

```
START: (tm_mon == 9) && (tm_hour < 8) && (tm_hour > 17) && (tm_isdst = 1)
```

Date:

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

tm_wday
number of days since Sunday (0-6).

tm_yday
number of days since January 1 (0-365).

tm_mon
number of months since January (0-11).

tm_year
the number of years since 1900 (0-99).

Macros that contain references to other macros are bound to that macros's definition, not its current value. For example,

```
X : "xxx"
Y : X
X : "yyy"
```

results in **Y** being evaluated as *yyy*.

Macros that contain macro invocations are bound to the current value of the macro, for example,

```
X = "xxx"
Y : "$(X)"
X = "yyy"
```

results in **Y** having a value of *xxx*.

Keyword Summary

This section contains summaries keywords you can use in the administration file and those you can use in the configuration file.

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.

Admin. File Keyword	Stanza(s)	Brief Description	For Details
account	User, Group	A list of account numbers available to a user submitting jobs.	"Step 2: Specify User Stanzas" on page 83
admin	Group, Class	A list of administrators for a group or class.	"Step 3: Specify Class Stanzas" on page 86
alias	Machine	Lists one or more alias names to associate with the machine name.	"Step 1: Specify Machine Stanzas" on page 78
central_manager	Machine	When **true**, this designates the machine as the LoadLeveler central manager.	"Step 1: Specify Machine Stanzas" on page 78
class_comment	Class	Text characterizing the class	"Step 3: Specify Class Stanzas" on page 86
core_limit	Class	Specifies the maximum and/or minimum size core file a job can create.	"Limit Keywords" on page 88
cpu_limit	Class	The maximum and/or minimum CPU time a job can use.	"Limit Keywords" on page 88
cpu_speed_scale	Machine	Determines whether CPU time is normalized according to machine speed.	"Step 1: Specify Machine Stanzas" on page 78
data_limit	Class	The maximum and/or minimum size data segment a job can use.	"Limit Keywords" on page 88
default_class	User	A class name that is the default value assigned to jobs submitted by users for which no class statement appears.	"Step 2: Specify User Stanzas" on page 83
default_group	User	A group name to which the user belongs.	"Step 2: Specify User Stanzas" on page 83
exclude_groups	Class	A list of groups names identifying those who cannot submit jobs of a particular class.	"Step 3: Specify Class Stanzas" on page 86
exclude_users	Class, Group	A list of user names identifying those who cannot submit jobs of a particular class or who are not members of the group.	"Step 3: Specify Class Stanzas" on page 86
file_limit	Class	The maximum and/or minimum size of a file that a job can create.	"Limit Keywords" on page 88
include_groups	Class	A list of groups names identifying those who can submit jobs of a particular class.	"Step 3: Specify Class Stanzas" on page 86
include_users	Class, Group	A list of user names identifying those who can submit jobs of a particular class or who do belong to the group.	"Step 3: Specify Class Stanzas" on page 86
job_cpu_limit	Class	The maximum and/or minimum amount of CPU time an individual job step can use per processor.	"Limit Keywords" on page 88

Admin. File Keyword	Stanza(s)	Brief Description	For Details
master_node_exclusive	Machine	When **true**, this machine is used only as a master node for parallel jobs.	"Step 1: Specify Machine Stanzas" on page 78
master_node_requirement	Class	When **true**, jobs in this class have the requirement that they run on a master node having the **master_node_exclusive** setting.	"Step 3: Specify Class Stanzas" on page 86
maxidle	User, Group	Maximum number of idle jobs this user or group can have simultaneously.	"Step 2: Specify User Stanzas" on page 83
maxjobs	User, Class, Group	Maximum number of jobs this user, class, or group can have running simultaneously.	"Step 2: Specify User Stanzas" on page 83
max_jobs_scheduled	Machine	The maximum number of jobs that this machine can run.	"Step 1: Specify Machine Stanzas" on page 78
max_processors	User, Class, Group	The maximum number of processors a user can request for a parallel job.	"Step 2: Specify User Stanzas" on page 83
maxqueued	Group, User	The maximum number of jobs a single group or user can have queued at the same time.	"Step 2: Specify User Stanzas" on page 83
name_server	Machine	A list of nameservers used for a machine.	"Step 1: Specify Machine Stanzas" on page 78
nice	Class	Increments the *nice* value of a job.	"Step 3: Specify Class Stanzas" on page 86
NQS_class	Class	When **true**, any job submitted to this class is routed to an NQS machine.	"Step 3: Specify Class Stanzas" on page 86
NQS_query	Class	A list of queue names to use to monitor and cancel jobs.	"Step 3: Specify Class Stanzas" on page 86
NQS_submit	Class	A name that identifies the name of the NQS pipe queue to which the job will be routed.	"Step 3: Specify Class Stanzas" on page 86
priority	User, Class, Group	A number that identifies the priority of the appropriate user, class, or group.	"Step 2: Specify User Stanzas" on page 83
pvm_root	Machine	A directory in which PVM 3.3 is installed.	"Step 1: Specify Machine Stanzas" on page 78
rm_host	Machine	When **true**, defines the machine as an SP system machine which can communicate with the resource manager.	"Step 1: Specify Machine Stanzas" on page 78
rss_limit	Class	The maximum and/or minimum resident set size for a job.	"Limit Keywords" on page 88
schedd_host	Machine	When **true**, this machine is used to help submit-only machines access LoadLeveler hosts that run LoadLeveler jobs.	"Step 1: Specify Machine Stanzas" on page 78
speed	Machine	The weight associated with the machine.	"Step 1: Specify Machine Stanzas" on page 78
stack_limit	Class	The maximum and/or minimum size for a stack.	"Limit Keywords" on page 88
submit_only	Machine	When **true**, designates this as a submit-only machine.	"Step 1: Specify Machine Stanzas" on page 78
type	All	The type of stanza.	"Administering LoadLeveler" on page 77
wall_clock_limit	Class	The maximum and/or minimum amount of elapsed time for which a job can run.	"Limit Keywords" on page 88

Configuration File Keywords and Macros

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 keywords, and macros. 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 Keyword	Brief Description	For Details
ACCT	Turns the accounting function on (or off).	"Step 7: Define Job Accounting" on page 106
ACCT_VALIDATION	The module called to perform account validation.	"Step 7: Define Job Accounting" on page 106
ACTION_ON_MAX_REJECT	Specifies whether a job is cancelled or put in a User Hold status when the job exceeds the **MAX_JOB_REJECT** value.	"Additional Configuration File Keywords" on page 119
ADMIN_FILE	Points to the administration file containing user, class, and machine list stanzas.	"Step 9: Specify Where Files and Directories are Located" on page 108
AFS_GETNEWTOKEN	The filter which can be used to renew an AFS token.	"Additional Configuration File Keywords" on page 119
ARCH	The standard architecture of the system.	"Step 3: Define LoadLeveler Machine Characteristics" on page 100
BIN	The directory where LoadLeveler binaries are kept.	"Step 9: Specify Where Files and Directories are Located" on page 108
CENTRAL_MANAGER_HEARTBEAT_INTERVAL	The amount of time in seconds that defines how frequently primary and alternate central manager communicate with each other.	"Step 8: Specify Alternate Central Managers" on page 107
CENTRAL_MANAGER_TIMEOUT	The number of heartbeat intervals that an alternate central manager will wait before declaring that the primary central manager is not operating.	"Step 8: Specify Alternate Central Managers" on page 107
Class	The class of jobs that can run on the machine.	"Step 3: Define LoadLeveler Machine Characteristics" on page 100
CLIENT_TIMEOUT	The maximum time, in seconds, that a daemon waits to respond to a process over TCP/IP.	"Step 11: Define Network Characteristics" on page 111
COLLECTOR_DGRAM_PORT	The port number used when connecting to a daemon.	"Step 11: Define Network Characteristics" on page 111
CONTINUE	Continue expression. Determines if a job should continue.	"Step 2: Manage a Job's Status Using Control Expressions" on page 98

Configuration File Keyword	Brief Description	For Details
CUSTOM_METRIC	A machine's relative priority to run jobs.	"Step 3: Define LoadLeveler Machine Characteristics" on page 100
CUSTOM_METRIC_COMMAND	An executable whose exit code is value is assigned to **CUSTOM_METRIC**.	"Step 3: Define LoadLeveler Machine Characteristics" on page 100
EXECUTE	The local directory to store the executable checkpoints of jobs submitted by other machines.	"Step 9: Specify Where Files and Directories are Located" on page 108
Feature	A string specifying unique characteristics of a machine.	"Step 3: Define LoadLeveler Machine Characteristics" on page 100
GLOBAL_HISTORY	The directory containing the global history files.	"Step 7: Define Job Accounting" on page 106
HISTORY	The pathname of the history file for local LoadLeveler jobs.	"Step 9: Specify Where Files and Directories are Located" on page 108
JOB_ACCT_Q_POLICY	The amount of time in seconds that determines how often the startd daemon updates the schedd daemon with accounting data of running jobs.	Chapter 7, "Gathering Job Accounting Data" on page 163
JOB_EPILOG	Pathname of the epilog program.	"Step 12: Write Prolog and Epilog Programs" on page 112
JOB_LIMIT_POLICY	The amount of time in seconds that LoadLeveler checks to see if **job_cpu_limit** has been exceeded.	Chapter 7, "Gathering Job Accounting Data" on page 163
JOB_PROLOG	Pathname of the prolog program.	"Step 12: Write Prolog and Epilog Programs" on page 112
JOB_USER_EPILOG	Pathname of the user epilog program.	"Step 12: Write Prolog and Epilog Programs" on page 112
JOB_USER_PROLOG	Pathname of the user prolog program.	"Step 12: Write Prolog and Epilog Programs" on page 112
KBDD	KBDD expression. Location of kbdd executable (Loadl_kbdd).	"LoadLeveler Daemons" on page 6
KBDD_DEVICE	Keyboard device file.	"Additional Configuration File Keywords" on page 119
KILL	Kill expression. Determines if vacated jobs should be killed.	"Step 2: Manage a Job's Status Using Control Expressions" on page 98
LIB	The directory where LoadLeveler libraries are kept.	"Step 9: Specify Where Files and Directories are Located" on page 108
LOADL_ADMIN	List of LoadLeveler administrators.	"Step 1: Define LoadLeveler Administrators" on page 97

Configuration File Keyword	Brief Description	For Details
LOCAL_CONFIG	Pathname of the optional local configuration file containing information specific to a node in the LoadLeveler network.	"Step 9: Specify Where Files and Directories are Located" on page 108
LOG	Local directory for storing log files.	"Step 9: Specify Where Files and Directories are Located" on page 108
MACHINE_AUTHENTICATE	Specifies whether machine validation is performed.	"Step 3: Define LoadLeveler Machine Characteristics" on page 100
MACHINE_UPDATE_INTERVAL	The time, in seconds, during which machines must report to the central manager.	"Additional Configuration File Keywords" on page 119
MACHPRIO	Machine priority expression	"Step 6: Prioritize the Order of Executing Machines Maintained by the Negotiator" on page 104
MAIL	Name of a local mail program used to override default mail notification.	"Using Your Own Mail Program" on page 281
MASTER	Location of the master executable (LoadL_master).	"LoadLeveler Daemons" on page 6
MASTER_DGRAM_PORT	The port number used when connecting to the daemon.	"Step 11: Define Network Characteristics" on page 111
MASTER_STREAM_PORT	The port number to used when connecting to the daemon.	"Step 11: Define Network Characteristics" on page 111
MAX_CKPT_INTERVAL	The maximum number of seconds between checkpoints for running jobs.	"Step 13: Enable Checkpointing" on page 113
MAX_JOB_REJECT	The number of times a job is rejected before it is cancelled or put in User Hold status.	"Additional Configuration File Keywords" on page 119
MAX_STARTERS	The maximum number of jobs that can run simultaneously.	"Step 4: Specify How Many Jobs a Machine Can Run" on page 102
MIN_CKPT_INTERVAL	The minimum number of seconds between checkpoints for running jobs.	"Step 13: Enable Checkpointing" on page 113
NEGOTIATOR	Location of the negotiator executable (LoadL_negotiator).	"LoadLeveler Daemons" on page 6
NEGOTIATOR_INTERVAL	The time interval, in seconds, at which the negotiator daemon updates the status of jobs in the LoadLeveler cluster and negotiates with machines that are available to run jobs.	"Additional Configuration File Keywords" on page 119
NEGOTIATOR_LOADAVG_INCREMENT	The factor added to the startd machine's load average to compenstate for the increased load caused by starting another machine.	"Additional Configuration File Keywords" on page 119
NEGOTIATOR_PARALLEL_DEFER	The length of time that a job is given to accumulate processors.	"Scheduling Considerations" on page 56

Configuration File Keyword	Brief Description	For Details
NEGOTIATOR_PARALLEL_HOLD	The length of time a job attempts to collect machines before releasing them.	"Scheduling Considerations" on page 56
NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL	The amount of time in seconds between calculation of the **SYSPRIO** values for waiting jobs.	"Additional Configuration File Keywords" on page 119
.NEGOTIATOR_REJECT_DEFER	The amount of time in seconds the negotiator waits before it considers scheduling a job to a machine that recently rejected the job.	"Additional Configuration File Keywords" on page 119
NEGOTIATOR_REMOVE_COMPLETED	The amount of time the negotiator keeps information on completed and removed jobs.	"Additional Configuration File Keywords" on page 119
NEGOTIATOR_RESCAN_QUEUE	The amont of time the negotiator waits to rescan the job queue for machines that temporarily have non-runnable jobs.	"Additional Configuration File Keywords" on page 119
NEGOTIATOR_STREAM_PORT	The port number used when connecting to the daemon.	"Step 11: Define Network Characteristics" on page 111
NQS_DIR	The directory where NQS commands reside.	"Step 9: Specify Where Files and Directories are Located" on page 108
OBITUARY_LOG_LENGTH	The number of lines from the ned of the file that are appended to the Master_Log.	"Additional Configuration File Keywords" on page 119
POLLING_FREQUENCY	The frequency in seconds the startd daemon uses to evaluate the load on the local machine and to decide whether to suspend, resume, or abort jobs.	"Additional Configuration File Keywords" on page 119
POLLS_PER_UPDATE	The frequency, in POLLING_FREQUENCY intervals, with which the startd daemon updates the central manager.	"Additional Configuration File Keywords" on page 119
PUBLISH_OBITUARIES	When **true**, specifies that the master daemon sends mail to the administrator(s) when any daemon it manages dies abnormally.	"Additional Configuration File Keywords" on page 119
RELEASEDIR	The directory where all the LoadLeveler software resides.	"Step 9: Specify Where Files and Directories are Located" on page 108
RESTARTS_PER_HOUR	The number of times the master daemon attempts to restart a daemon that dies abnormally.	"Additional Configuration File Keywords" on page 119
SCHEDD	Location of the schedd executable (LoadL_schedd).	"LoadLeveler Daemons" on page 6
SCHEDD_RUNS_HERE	Specifies whether this daemon will run on the host.	"Step 3: Define LoadLeveler Machine Characteristics" on page 100
SCHEDD_STREAM_PORT	The port number used when connecting to the daemon.	"Step 11: Define Network Characteristics" on page 111
SCHEDULER_API	When **YES**, disables the native LoadLeveler scheduling algorithm.	"Additional Configuration File Keywords" on page 119

Configuration File Keyword	Brief Description	For Details
SPOOL	The local directory where LoadLeveler keeps the local job queue and checkpoint files.	"Step 9: Specify Where Files and Directories are Located" on page 108
START	Start expression. Determines if a machine can run a job.	"Step 2: Manage a Job's Status Using Control Expressions" on page 98
STARTD	Location of the startd executable (LoadL_startd).	"LoadLeveler Daemons" on page 6
STARTER	Location of the starter executable (LoadL_starter).	"LoadLeveler Daemons" on page 6
STARTD_RUNS_HERE	Specifies whether this daemon will run on the host.	"Step 3: Define LoadLeveler Machine Characteristics" on page 100
START_DAEMONS	Specifies whether to start the daemons on the machine.	"Step 3: Define LoadLeveler Machine Characteristics" on page 100
STARTD_DGRAM_PORT	The port number used when connecting to the daemon.	"Step 11: Define Network Characteristics" on page 111
STARTD_STREAM_PORT	The port number used when connecting to the daemon.	"Step 11: Define Network Characteristics" on page 111
SUBMIT_FILTER	The program you want to run to filter a job script when the job is submitted.	"Filtering a Job Script" on page 281
SUSPEND	Suspend expresson. Determines if a job should be suspended.	"Step 2: Manage a Job's Status Using Control Expressions" on page 98
SYSPRIO	System priority expression.	"Step 5: Prioritize the Queue Maintained by the Negotiator" on page 103
TRUNC_KBDD_LOG_ON_OPEN	When **true**, specifies the log file is restarted with every invocation of the daemon.	"Step 10: Record and Control Log Files" on page 110
TRUNC_MASTER_LOG_ON_OPEN	When **true**, specifies the log file is re started with every invocation of the daemon.	"Step 10: Record and Control Log Files" on page 110
TRUNC_NEGOTIATOR_LOG_ON_OPEN	When **true**, specifies the log file is restarted with every invocation of the daemon.	"Step 10: Record and Control Log Files" on page 110
TRUNC_SCHEDD_LOG_ON_OPEN	When **true**, specifies the log file is restarted with every invocation of the daemon.	"Step 10: Record and Control Log Files" on page 110
TRUNC_STARTD_LOG_ON_OPEN	When **true**, specifies the log file is restarted with every invocation of the daemon.	"Step 10: Record and Control Log Files" on page 110
TRUNC_STARTER_LOG_ON_OPEN	When **true**, specifies the log file is restarted with every invocation of the daemon.	"Step 10: Record and Control Log Files" on page 110
VACATE	The vacate expression. Determines whether suspended jobs should be vacated.	"Step 2: Manage a Job's Status Using Control Expressions" on page 98

Configuration File Keyword	Brief Description	For Details
X_RUNS_HERE	When **true**, specifies you want to start the keyboard daemon (unles you are running on Sun machine or an HP machine).	"Step 3: Define LoadLeveler Machine Characteristics" on page 100

User-Defined Keywords

The following table serves only as a reference. These keywords are described in more detail in "User-Defined Keywords" on page 121.

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

Reserved Macros

The following table serves only as a reference. These keywords are described in more detail in "Reserved Macros" on page 123.

Keyword	Brief Description	For Details
ARCH or Arch	Standard architecture of the system.	"Reserved Macros" on page 123
AvailableClasses	Job classes currently available on the machine.	"Reserved Macros" on page 123
ClassSysprio	Job priority for the class.	"Step 5: Prioritize the Queue Maintained by the Negotiator" on page 103
Cpus	Number of CPU's installed.	"Reserved Macros" on page 123
CurrentTime	The UNIX date that includes the current system time, in seconds, since January 1, 1970.	"Reserved Macros" on page 123
Disk	Free disk in megabytes on the filesystem where checkpoints are stored.	"Reserved Macros" on page 123
domain or domainname	Dynamically indicates the domain name of the current host machine where the program is running.	"Reserved Macros" on page 123

Keyword	Brief Description	For Details
EnteredCurrentState	Value of CurrentTime when the current state was entered.	"Reserved Macros" on page 123
GroupQueuedJobs	The number of jobs either running or queued for the LoadLeveler group.	"Step 5: Prioritize the Queue Maintained by the Negotiator" on page 103
GroupRunningJobs	The number of jobs currently running for the LoadLeveler group.	"Step 5: Prioritize the Queue Maintained by the Negotiator" on page 103
GroupSysprio	The job priority for the group.	"Step 5: Prioritize the Queue Maintained by the Negotiator" on page 103
GroupTotalJobs	The total number of jobs associated with the LoadLeveler group.	"Step 5: Prioritize the Queue Maintained by the Negotiator" on page 103
host or hostname	Dynamically indicates the name of the host machine where the program is running.	"Reserved Macros" on page 123
KeyboardIdle	Number of seconds since the keyboard or mouse was last used.	"Reserved Macros" on page 123
LoadAvg	Berkeley one-minute load average.	"Reserved Macros" on page 123
Machine	Name of the current machine.	"Reserved Macros" on page 123
Memory	Physical memory installed on the machine in megabytes.	"Reserved Macros" on page 123
OpSys	Indicates the operating system on the host where the program is running.	"Reserved Macros" on page 123
QDate	Difference in seconds between when the negotiator starts up and when the job is submitted.	"Reserved Macros" on page 123
State	State of the startd. Can be None, Busy, Running, Idle, Suspend, Flush, or Drain.	"Reserved Macros" on page 123
Status	Status of the job; an integer code corresponding to Idle (I), Running (R), Completed (C), Removed (X), Pending (P), System_hold (S), User_hold (U), or Deferred (D).	"Reserved Macros" on page 123
tilde	Dynamically defines the pathname of the LoadLeveler home directory.	"Reserved Macros" on page 123
tm_hour	Number of hours since midnight (0-23).	"Reserved Macros" on page 123
tm_isdst	Daylight Savings Time flag: positive when in effect, zero when not in effect, negative when information is unavailable.	"Reserved Macros" on page 123
tm_mday	Number of the day of the month (1-31).	"Reserved Macros" on page 123
tm_min	Number of minutes after the hour (0-59).	"Reserved Macros" on page 123
tm_mon	Number of months since January (0-11).	"Reserved Macros" on page 123
tm_sec	Number of seconds after the minute (0-59).	"Reserved Macros" on page 123
tm_wday	Number of days since Sunday (0-6).	"Reserved Macros" on page 123
tm_yday	Number of days since January 1 (0-365).	"Reserved Macros" on page 123
tm_year	Number of years since 1900 (0-99).	"Reserved Macros" on page 123
UserPrio	User defined priority of a job.	"Step 5: Prioritize the Queue Maintained by the Negotiator" on page 103
UserQueuedJobs	The number of jobs either running or queued for the user.	"Step 5: Prioritize the Queue Maintained by the Negotiator" on page 103

Keyword	Brief Description	For Details
UserRunningJobs	The number of jobs currently running for the user.	“Step 5: Prioritize the Queue Maintained by the Negotiator” on page 103
UserSysprio	The priority of the user who submitted the job.	“Step 5: Prioritize the Queue Maintained by the Negotiator” on page 103
UserTotalJobs	The total number of jobs associated with the this user.	“Step 5: Prioritize the Queue Maintained by the Negotiator” on page 103
VirtualMemory	The size of the available swap space on the machine in kilobytes.	“Reserved Macros” on page 123

Chapter 6. Using Interactive LoadLeveler

Interactive LoadLeveler consists of Interactive Session Support (ISS). You can use ISS to balance interactive workloads across a pool of servers in a manner which is completely transparent to users and applications. ISS is useful for applications such as email servers, World Wide Web servers, distributed parallel database queries, distributed logins, and other TCP/IP applications.

Changes from the Previous Release

This version of ISS includes many functional enhancements and improves the overall RAS (reliability, availability, and serviceability) of ISS. Changes and additions include:

- CONTINUOUS metrics
- Health checks: ISS performs server status checks between nameserver updates
- State sensitive operation on all types of metrics
- Load capping
- Multiple domains
- Alarm facility to alert you when a pool has no available servers
- Backup capabilities
- Elimination of the use of **rsh** for CUSTOM metrics

Overview of ISS

Without ISS, users wishing to gain access to a pool of shared computing resources from TCP/IP applications must request a connection to a specific server within the pool by supplying a machine name or internet address which identifies the particular server. Traditionally there was no facility for leveling of the load on the shared pool, and the distribution of logged in users across the shared pool was arbitrary.

In TCP/IP networks a Domain Name System (DNS) nameserver can be used to translate machine names to internet addresses. ISS is used in conjunction with such a nameserver to provide load balancing. Instead of specifying the machine name of a particular server, the user specifies the name of a pool which has been set up by the administrator. ISS monitors the load on each of the servers in the pool, and alters the nameserver's mapping from domain names to internet addresses as the load profile changes. This efficiently balances the user load across all the available servers. Instructions on how to set up pools appear in “Using Interactive Session Support” on page 138.

All servers in the pool should be equivalent as far as the user is concerned, and it is usually desirable to create a common data space and common name space. This means that whichever server a user is connected to, the user will be able to log in using a consistent user name and password, and will be able to access the appropriate data. You can accomplish this by using a distributed file system such as AFS or NFS, or by using a shared disk architecture.

© Copyright IBM Corp. 1993, 1996

You can configure ISS to suit your environment in terms of frequency of access, total number of users and types of access (for example, short queries, long-running queries or CPU-intensive loads).

ISS supports multiple pools of server nodes. This allows a large shared resource, such as an SP, to be divided into a number of pools for separate purposes. For example, you might dedicate four nodes of a 16-node SP to long-running queries which are balanced according to a load average statistic and use the remaining 12 nodes for short queries and update transactions which you might balance using a round-robin scheduling algorithm. It is possible to define overlapping pools if desired.

To use ISS, run the program called **issmonitor** (also called the monitor in this chapter) on the nameserver machine. Depending on your pool configuration the **issmonitor** may in turn start an instance of the **issagent** program on each server which appears in at least one custom pool. See "Using ISS Custom Configuration" on page 141 for more information.

You can configure a second machine to act as a standby ISS monitor, which you can start if the first monitor fails. ISS allows a standby monitor to take over even when you use custom metrics and agents are running on the servers. To use a second monitor, both machines must be either in the site nameserver file or in the clients' /**etc/resolv.conf** file.

You can configure ISS with any of the following methods:

- Using the ISS round-robin configuration
- Using the ISS custom configuration
- Using a LoadLeveler configuration

With the custom and round-robin configurations, you do not need to run LoadLeveler to use ISS. For more information on configuration planning, see Table 6 on page 140.

Using Interactive Session Support

This section lists and describes the steps you, as an administrator, need to perform in order to allow your end users to use ISS to perform interactive login sessions.

Step 1: Installing ISS

You need to install ISS on the machine running the TCP/IP domain nameserver which has authority over the subdomain over which interactive sessions are to be leveled. For more information about TCP/IP domains and how to configure the nameserver, see "Configure the TCP/IP nameserver" on page 150.

You must install ISS using the **root** id.

If you plan to use LoadLeveler pools with ISS, you need to create the user ID **loadl** and group **loadl**.

Note: For more information on installing ISS, refer to the *LoadLeveler Installation Memo*, or to the appropriate README file on the product tape.

Step 2: Planning your ISS configuration

In order to configure ISS you need to understand the concept of pools and the different types of load measurement that may be used for a pool. This section will present a brief description of these terms, including factors which affect the choice of measurement type for each pool.

Pools

A pool is a group of servers which are equally suitable for a particular purpose. A pool is identified by a domain name. This is the name that users will type when they want to access the pool. The pool name will be mapped to the internet address of one of the nodes in the pool, which allows the load created by TCP/IP connected applications to be spread across the servers.

A pool can relate to any computing resource which has a number of servers. For example you might define a pool which contains some or all of the nodes of an SP or a cluster.

ISS supports multiple pools, so you could partition your computing resource into two or more pools, which may optionally overlap. You could even define multiple pools which each contain an identical group of servers, but this may limit the usefulness of having separate pool definitions.

Types of Measurement

Each pool you configure must have a measurement type associated with it. This tells ISS how you want it to monitor the load profile across the servers in the pool. You can choose from 3 types of measurement, as described in the following table. For each pool you wish to configure, you'll need to decide which of the measurement types is appropriate. Your pools are independent of each other and the choice of measurement type for a pool does not affect the choices available for other pools.

Refer to the following table to determine which method best suits your needs:

Table 6. ISS Configuration Methods

Select this method:	If these circumstances apply:	See:
ISS Round-Robin Configuration	You do not want to measure the actual loads on the servers in the pool, and believe that simply cycling through them will provide a good distribution of load for the types of activity which will be performed on the servers. In the pool configuration you supply a list of machine names and/or internet addresses of the servers in the pool. A pool configured to use round-robin measurement is independent of LoadLeveler.	"Using ISS Round-Robin Configuration" on page 140.
ISS Custom Configuration	You want to measure load in a manner which is not provided by LoadLeveler, either because it does not measure the same thing, or because you want to measure load more (or less) frequently than your LoadLeveler configuration would allow. In the pool configuration you supply a list of machine names and/or internet addresses of the servers in the pool. A pool configured to use custom measurement is independent of LoadLeveler.	"Using ISS Custom Configuration" on page 141.
LoadLeveler Configuration	There is an existing LoadLeveler configuration in place and you want to define the pool by using the machine descriptions available in the LoadLeveler configuration files. These files contain stanzas that describe the characteristics of each machine and the type of workload that it supports. You must define a class for ISS within LoadLeveler.	"Using A LoadLeveler Configuration" on page 141.

You can have several pools with each operating in a different manner. For example, you could configure a LoadLeveler pool and additionally provide a number of pools which use ISS custom metrics or round-robin metrics.

Using ISS Round-Robin Configuration

An alternative means of leveling interactive logins across a pool is to cycle through the servers in the pool in a round-robin fashion, with each server being selected for a configurable interval. This method does not pay any attention to the actual physical load on each of the servers but instead cycles through the servers on a configurable time basis to spread the users across the servers.

When you use ISS in this way, ISS is independent of LoadLeveler and LoadLeveler does not have to be running. You configure ISS by defining pools as described in "Step 3: Configuring ISS" on page 141. These pools are not related to any other LoadLeveler configuration information.

There are advantages and disadvantages in using the round-robin method and its use is more appropriate in certain environments than in others. The main advantage is that using round-robin avoids the overhead (on the servers) associated with the measurement of load which is introduced by the other measurement types. The main disadvantage of round-robin scheduling is that ISS does not notice if a particular server is getting very busy, and will continue to schedule it periodically as the server to which new users of the pool connect. Round-robin scheduling works well provided the client load has a reasonably uniform profile in terms of frequency of access and duration of access.

For all measurement methods, including round-robin, ISS notices if a server fails to respond to attempts to ping it and does not schedule such servers.

Using ISS Custom Configuration

You can use ISS independently of LoadLeveler. If you do this you can use any metric that you choose, and you define the metric in the ISS configuration file. This metric can be anything that can be executed on each of the servers and which will return a numeric result. To support this flexibility there is a parameter in the ISS configuration file which determines how this result is interpreted. This is necessary because you may decide to define load in terms of a quantity for which smaller values are preferable (for example, the number of logged on users), or in terms of a quantity for which larger values are preferable (for example, the amount of available memory). The interpretation is controlled by the **POLICY** parameter. When you want lower numbers to represent lower loading, you need to set **POLICY** to **MIN**, which indicates that ISS should minimize the metric. When you want higher numbers to represent higher loading, you need to set **POLICY** to **MAX**, so that ISS maximizes the metric. See the description of the **POLICY** keyword in "Step 3: Configuring ISS" for more details. The following example illustrates a custom metric which counts the number of processes running on a system:

```
ps -ef | wc -l
```

When you use ISS in this way, ISS is independent of LoadLeveler and LoadLeveler does not have to be running. You configure ISS by defining pools as described in "Step 3: Configuring ISS." These pools are not related to any other LoadLeveler configuration information.

Using A LoadLeveler Configuration

You can use ISS in conjunction with a LoadLeveler pool configuration. A pool is defined by a particular class and sets of requirements and preferences. An administrator can define multiple pools which can have different combinations of class, requirements, and preferences. Pools can also overlap.

When you use ISS in this way, LoadLeveler must be running because ISS uses LoadLeveler to determine the load on each of the servers in the pool. ISS uses the metrics configured in LoadLeveler to determine the loading and availability of servers in the pool. You should configure LoadLeveler by specifying the machine descriptions and then configure ISS, using the same classes, requirements and preferences used in the LoadLeveler files. See "Step 3: Configuring ISS" for more details.

Step 3: Configuring ISS

Configuring ISS is a two-part process. The first part is to configure the ISS configuration file to suit your needs, and the second part is to configure the nameserver.

Configure the ISS Configuration File

ISS has its own configuration file which defines the pools of servers and the metrics to use for each pool. A single ISS configuration file can contain definitions for LoadLeveler pools, ISS custom pools, and ISS round-robin pools. The name of the configuration file is supplied as a command line argument when ISS is run and you can name the file anything you want. A sample **lss_config** configuration file is found in **/usr/lpp/LoadL/iss**. Copy the example file and edit it to specify your pools

and their characteristics. You may have a variety of configuration files for different workloads and other requirements.

Syntax of the ISS Configuration File: The ISS configuration file is an ASCII file consisting of a series of defintions. Each definition must be at the start of a line, and starts with a keyword which is followed by possible further keywords and then a value. Blank lines are ignored and lines that begin with semicolons (;) or hash-symbols (#) are treated as comments. All keywords are case insensitive. The case of parameters is preserved, since they may contain custom metrics or hostnames.

The majority of keywords relate to a specific pool definition, and this is always the pool definition in which they appear. There are some keywords, however, that do not relate to any particular pool. They are: **MONITOR_TRACE**, **AGENT_TRACE**, **AGENT_LOG** and **ISS_PORT**, which can appear at the start of any line in the configuration file, but are most clearly expressed at the top of the file.

The keywords which relate to a specific pool can occur in any order within the pool definition with the exception that the **NAME** keyword must be the first entry within each pool.

For each pool definition, you need to supply all keywords which are required for the type of metric used to monitor that pool. Other keywords are either optional or are ignored. The following table defines which pool keywords are required for each type of metric and which pool keywords are optional or ignored. **MONITOR_TRACE**, **AGENT_TRACE**, **AGENT_LOG** and **ISS_PORT** do not appear in the table because they are not specific to any particular pool.

Table 7. Parameter Significance for Metrics

Keyword	**Metric**		
	LOADLEVELER	**CUSTOM**	**ROUNDROBIN**
CLASS	**required**	ignored	ignored
HEARTBEAT_INTERVAL	**required**	**required**	**required**
HEARTBEATS_PER_UPDATE	**required**	**required**	**required**
LIMIT	ignored	optional	ignored
METRIC	**required**	**required**	**required**
NAME	**required**	**required**	**required**
NAMED_DATAFILE	optional (1)	optional (1)	optional (1)
NAMED_REVERSEFILE	optional	optional	optional
POLICY	ignored	**required**	ignored
POOL_ALARM	optional	optional	optional
PREFERENCES	optional	ignored	ignored
REQUIREMENTS	optional	ignored	ignored
SERVERS	ignored	**required**	**required**

Note: (1) – The first pool must contain the **NAMED_DATAFILE** keyword. This keyword is optional for all subsequent pools since they can inherit the previous pool's value.

ISS can level the load of more than one pool of machines at a time. To do this, you can configure multiple pool names in the ISS configuration file with an associated set of keywords and values to define each pool.

Now would be a good time to refer to the sample configuration file supplied with ISS, or the sample **Iss_config** in "ISS Configuration File" on page 160.

ISS Configuration File Keywords

This section gives details of the definitions in the ISS configuration file. Keywords appear in alphabetical order.

AGENT_LOG

Place this keyword at the start of any line in the configuration file to tell ISS the name (and maximum size) of the log file to which agent trace output should be written on servers. If **AGENT_TRACE** is turned on, then the **AGENT_LOG** keyword must be provided and must specify the name of a file into which trace data can be written. There is an optional size parameter which can be used to limit the maximum size (in bytes) to which the log file can grow. When the log file reaches this size, it is moved to *filename*.**old**, and a new one is started, using *filename*. The size parameter therefore controls the minimum amount of history that is available. Note that there could be up to **2*****size* bytes of trace data, since *filename*.**old** and *filename* can each extend up to *size* bytes.

The syntax is:

AGENT_LOG *filename size*

For example, to write all agent trace data to /**tmp**/**issagent.log** and limit the size of trace files to 10K bytes, enter the following:

```
AGENT_LOG /tmp/issagent.log 10000
```

This keyword is mandatory if **AGENT_TRACE** is defined, and is ignored otherwise.

AGENT_TRACE

Place this keyword at the start of any line in the configuration file to cause ISS to produce trace output on any servers which are running **issagent**. This output is sent to the log file specified in **AGENT_LOG** and can be useful for testing a new configuration and helping to resolve any problems. If **AGENT_TRACE** is specified, then **AGENT_LOG** must also be specified.

You can use this keyword dynamically to turn trace on and off. See "Step 4: Reconfiguring ISS While it is Running" on page 154 for more information.

The syntax is:

AGENT_TRACE

This keyword is optional.

CLASS

Use this keyword when setting the **METRIC** keyword to **LOADLEVELER**. The value assigned to it must be a valid LoadLeveler class. This keyword allows you to restrict login sessions to those machines that support this class. The syntax is:

CLASS *LoadLeveler_Class*

For example, to set this keyword to the class **Fast_Response**, enter the following line:

```
CLASS Fast_Response
```

Note that it is important to ensure that other LoadLeveler jobs do not interfere with the running of the ISS pools you configure. You can achieve this isolation by using a class for your ISS LoadLeveler pools which will not be used by any of your other LoadLeveler jobs. A good candidate would be **Iss_Class**, which would need to be defined in the LoadLeveler configuration file for each server that appears in one or more ISS LoadLeveler pools. See “Reserved Macros” on page 123 for more information on the **Initiators** keyword.

If a pool's **METRIC** keyword is not set to **LOADLEVELER**, the **CLASS** keyword is ignored.

HEARTBEAT_INTERVAL

Indicates the time in seconds which ISS allows to elapse before it checks the well-being of the server currently at the top of it's ranking for this pool. It checks that the server is functional by issuing an ICMP **ping** command; this is the same as if you typed **ping** at the UNIX command line. While this does not test the state of everything on the server, it does at least ensure that the server is reachable at the IP level. If the server fails to respond, then the next server in the ranking is tested and selected. The **HEARTBEAT_INTERVAL** does not itself define how frequently **named** daemon updates can occur. This is defined by the product of **HEARTBEAT_INTERVAL** and **HEARTBEATS_PER_UPDATE**.

You can use this keyword to optimize the overall responsiveness of the pool in specific user workload scenarios. The syntax is:

HEARTBEAT_INTERVAL *heartbeat_interval*

For example, to initiate a heartbeat every 30 seconds, enter the following:

```
HEARTBEAT_INTERVAL 30
```

This keyword is required, regardless of the type of metric used.

HEARTBEATS_PER_UPDATE

Accepts a non-negative integer which indicates the number of heartbeats after which ISS will determine whether a **named** daemon update is required. This allows you to configure an update interval which is relatively large while using a relatively fast heartbeat, so that the well-being of the top-ranked server is checked frequently, but without putting too much load on the **named** daemon by updating it too often. The update interval is determined from the product of **HEARTBEAT_INTERVAL** and **HEARTBEATS_PER_UPDATE**.

There is a special case for this keyword, which is when it is set to 0. For LOADLEVELER and ROUNDROBIN metric types, this is equivalent to setting it to 1, but for CUSTOM metrics it tells ISS to use “live” reporting of loads. That is, rather than waiting for even 1 heartbeat before checking whether the name daemon needs updating, it will force an update whenever it receives a load figure from a server which alters which server is at the top of the ranking. This allows ISS to operate in a real-time mode, without waiting before updating. This setting should be used with caution, however, since it is possible that you could overload the **named** daemon if you have frequent load reports from a large number of servers.

You can use this keyword to optimize the overall responsiveness of the pool in specific user workload scenarios. The syntax is:

HEARTBEATS_PER_UPDATE *heartbeat_count*

For example, to prevent the **named** daemon updates from occurring more frequently that once every three heartbeats, enter the following:

```
HEARTBEAT_INTERVAL 3
```

In this example, if **HEARTBEAT_INTERVAL** is 10 seconds, you have limited the frequency of updates of the **named** daemon to once every 30 seconds.

This keyword is required, regardless of the type of metric used.

ISS_PORT

Place this keyword at the start of any line in the configuration file to tell ISS the port number you used when you configured the issagent on the servers in any CUSTOM pools.

It is recommended you put this entry near the top of the **lss_config** file, together with the other general definitions.

The syntax is:

ISS_PORT *port_number*

This keyword is required if you have any CUSTOM pools; otherwise this keyword is ignored.

LIMIT

Applies a load cap to the servers in a CUSTOM pool.

If **POLICY** is set to **MIN** you favor lower metric results, and the value you specify for **LIMIT** is therefore interpreted as an upper limit. If **POLICY** is set to **MAX** you favor higher metric results, and the value you specify for **LIMIT** is interpreted as a lower limit.

You may optionally specify an overflow server which is recommended for use when all other servers have reached capacity. This server may also appear in the SERVERS list, but will not be load capped if all other servers have reached the value you specify for **LIMIT**.

When all servers reach the defined **LIMIT**, and if no overflow server has been defined, then the pool name is removed from the **named** daemon, and the **POOL_ALARM** (if configured for this pool) is triggered.

You can use this keyword to control the total workload that will be directed to the pool, which in turn controls the responsiveness of the pool. The syntax is:

LIMIT *limit overflow_server*

For example, to limit the maximum number of users logged into any one server to 10, and to direct any further requests to a machine called homebase, enter the following:

```
LIMIT 10 homebase
```

This example assumes that a CUSTOM metric which measures the number of logged in users is being used. For example, **w | wc -l**, and that policy is set to **MIN**. The server homebase may or may not appear in the **SERVERS** list, but in either case will not be load capped when the other servers have all reached capacity.

This keyword is optional for **CUSTOM** metrics, and is ignored for other types.

METRIC

This keyword controls which of the three methods of operation ISS uses to level login sessions across the pool to which this keyword relates (see “Using Interactive Session Support” on page 138 for details).

The syntax is:

```
METRIC { LOADLEVELER | CUSTOM [CONTINUOUS] string | ROUNDROBIN }
```

Where:

LOADLEVELER

Specifies that ISS treats this pool as a LoadLeveler pool. This means that the pool consists of machines defined in LoadLeveler that meet the criteria specified in the **CLASS** and **REQUIREMENTS** keywords and which best match the criteria specified in the **PREFERENCES** keyword. The metric used is the metric being used by LoadLeveler.

To configure a pool to use a LoadLeveler metric, add the following line to the ISS configuration file:

```
METRIC LOADLEVELER
```

Refer to the table Table 7 on page 142 for details of which other keywords are mandatory and which are optional when this type of metric is used.

CUSTOM [CONTINUOUS] *string*

This value specifies that ISS is to use a custom metric for this pool. The metric used is defined by the *string* supplied. On each server configured in one or more **CUSTOM** pools, ISS starts an instance of issagent, which detects changes in load and sends them to the monitor. The monitor then uses this value to determine which machine would be the best one for new users to be connected to, based on the load-leveling policy specified by the **POLICY** keyword.

So that ISS can start the isagent program, you need to edit the **/etc/services** file and **/etc/inetd.conf** file on any servers that appear in **CUSTOM** pools, and add the following lines.

To **/etc/services** add:

```
issagent portnumber/tcp
```

where *portnumber* is a unique port number that you choose.

To **/etc/inetd.conf** add:

```
issagent stream tcp nowait root path/issagent issagent
```

where *path* is the path name of the installation directory in which the **issagent** executable is installed.

Remember to refresh **inetd**, so that it reads the new configuration. You can do this by entering:

```
refresh -s inetd
```

The *string* is one or more UNIX commands, probably separated by the “|” pipe-through operator. The end result of these commands must be an integer, between 0 and 99,999 inclusive, sent to standard output.

You should consider testing new metric strings by entering them on the command line before putting them in the configuration file.

You can measure the load either periodically or in a “live” (real-time) manner. The optional keyword **CONTINUOUS** indicates that the metric defined by *string* is not expected to return, but rather that it will run indefinitely, returning a series of load figures to stdout. Such a metric can periodically report a load figure. Or, it can be written to report load when it detects changes in the state of the server, which could occur at arbitrary times. When you use **CONTINUOUS** in conjuction with **HEARTBEATS_PER_UPDATE = 0**, you configure ISS in “live” mode. This setting should be used with caution, however, since it is possible that you could overload the **named** daemon if you have frequent load reports from a large number of servers.

It is important not to use the **CONTINUOUS** keyword with metrics that are not **CONTINUOUS**, and to always ensure that it is used with metrics which are **CONTINUOUS**.

Refer to the table Table 7 on page 142 for details of which other keywords are mandatory and which are optional when this type of metric is used.

Examples of Metrics

Some examples of custom metrics which have been used in a number of applications of ISS are described in this section. Note that all of these require **POLICY MIN** unless otherwise stated.

When specifying metrics, the metric string must fit on one line in the configuration file after the **METRIC** and **CUSTOM** keywords. If the load-leveling algorithm is highly complex, you should encode the algorithm in a shell script or in an application which is stored such that it is accessible on each of the servers. The custom metric will then be an invocation of that script or program which returns the computed metric value to standard output as an integer between 0 and 99,999 inclusive.

Example 1–Number of Running Processes: The following metric returns the number of processes running on the server. This value should be minimized when you want to identify the least loaded server.

```
METRIC CUSTOM ps -ef | wc -l
```

Example 2– System Idle Time: The following example uses **vmstat** to find the system idle time during a one second sample period. This value should be maximized when you want to identify the least loaded server.

```
METRIC CUSTOM vmstat 1 2 | tail -1 | awk '{print $16}'
```

ROUNDROBIN

Specifies that ISS is to use round-robin scheduling for this pool. This means that users requests are automatically rotated around the machines in the pool irrespective of their loading, at time intervals defined by the product **HEARTBEAT_INTERVAL * HEARTBEATS_PER_UPDATE**. The round-robin metric does not measure the load on the servers in the pool. Instead, it cycles around the servers in the order in which they appear in the **SERVERS** list.

To configure a pool for round-robin scheduling, add the following metric line in the configuration file:

```
METRIC ROUNDROBIN
```

Refer to the table Table 7 on page 142 for details of which other keywords are mandatory and which are optional when this type of metric is used.

MONITOR_TRACE
Place this keyword at the start of any line in the configuration file to cause ISS to produce trace output at the monitor. This output is sent to **stdout** and can be useful for testing a new configuration and helping to resolve any problems.

Used in connection with the ability to reconfigure ISS while it is still running, you can use this keyword dynamically to turn trace on and off. See "Step 4: Reconfiguring ISS While it is Running" on page 154.

The syntax is:

```
MONITOR_TRACE
```

This keyword is optional.

NAME
Defines the name users can specify to connect to the pool of machines. A **NAME** entry in the configuration file marks the start of the configuration parameters for a new pool. Therefore, the **NAME** line must be the first non-comment line in the definition of a pool. The syntax is:

```
NAME pool_name
```

where *pool_name* is the fully qualified name of the pool. For example, you can specify the pool as:

```
dept_pool.sp.site.corp.com
```

In this example, **site.corp.com** is the sub-domain over which the nameserver has authority.

The **NAME** keyword is required and must appear first when defining a pool. Pools names can be any string that you choose, but must conform to the requirements for valid TCP/IP domain names, such as not containing whitespace or certain punctuation characters.

NAMED_DATAFILE
Defines the path and filename of the datafile which contains the **A** type resource record for the name of the pool in which this definition is located in the **lss_config** file. This keyword must be provided in order for ISS to locate the nameserver data that it requires. However, all pools apart from the first may omit this keyword in which case they inherit the value that was defined (or inherited) by the previous pool in the **lss_config file**. The syntax is:

```
NAMED_DATAFILE path/filename
```

The datafile must contain the pool name as it appears in the **NAME** keyword.

NAMED_REVERSEFILE
Defines the path and filename of the datafile which contains the **PTR** type resource record for the name of the pool in which this definition is located in the **lss_config** file. This keyword must be provided if reverse datafile updates are required. The omission of this keyword from a pool definition indicates to ISS that reverse updates are not required for the pool. The syntax is:

```
NAMED_REVERSEFILE path/filename
```

The reverse file must contain the pool name as it appears in the **NAME** keyword.

POLICY
Indicates whether the metric function specified in a **METRIC CUSTOM** entry is to be minimized or maximized. The syntax is:

POLICY { MIN | MAX }

For example, a metric based on the number of active users would regard the smallest value as the best and would have the following entry:

```
POLICY MIN
```

A metric function based on CPU idle time, on the other hand, would regard the maximum value as best and, therefore, the policy would be specified as the following:

```
POLICY MAX
```

If **METRIC** is set to **LOADLEVELER** or **ROUNDROBIN**, this keyword is ignored. When **METRIC** is set to **CUSTOM**, this keyword is required.

POOL_ALARM

Specifies some action that you want taken if a pool runs out of available servers. This applies to all types of metric.

Servers may be unavailable because they do not respond when a heartbeat is sent to them, or (in the case of a CUSTOM pool) because they have reached the load cap specified in **LIMIT**.

The syntax is:

POOL_ALARM *string*

For example, to get ISS to send you a mail message when the pool becomes full or unavailable, you could enter the following:

```
POOL_ALARM echo "mypool not available at `date`" | mail root@admin.site.com
```

PREFERENCES

This keyword is only used in conjunction with the LoadLeveler metric. The value assigned to it must be a string containing valid LoadLeveler preferences, and allows you to bias login sessions to those machines that have the preferred characteristics in the LoadLeveler configuration. Refer to “Job Command File Keywords” on page 40 for information on how creating a Boolean expression containing preferred characteristics.

If you do not set METRIC to LOADLEVELER, this keyword is ignored. The syntax is:

PREFERENCES *Boolean_expression*

REQUIREMENTS

This keyword is only used in conjunction with the LoadLeveler metric. The value assigned to it must be a string containing valid LoadLeveler requirements. It allows you to restrict login sessions to those machines that have the required characteristics in the LoadLeveler configuration. Refer to “Job Command File Keywords” on page 40 for information on how to create a Boolean expression containing required characteristics.

If you do not set **METRIC** to **LOADLEVELER**, this keyword is ignored. The syntax is:

REQUIREMENTS *Boolean_expression*

SERVERS

Marks the start of a list of machine names or internet addresses (in dotted decimal notation), specifying the machines which you want to include in the pool and which are to be accessed via the pool name given in the preceding **NAME** entry. Where possible, you should use machine names because they

make administration simpler but you can use machine names and internet addresses interchangeably. Separate the list with blanks and continue the list on subsequent lines if required. If you use multiple lines, only issue the keyword on the first line.

The syntax is:

```
SERVERS server1 [server2...]
```

For example, for a pool of three machines the **SERVERS** entry might look like this:

```
SERVERS 1.2.3.10 deptserver 1.2.3.12
```

There is no requirement for machines to be within the same internet subnet groups. It is only necessary that the machines are accessible from the client and the nameserver machines. You can define overlapping pools, such that the same machine name or internet address may appear on the **SERVERS** line of more than one pool defined in the configuration file.

If you accidentally put a server into the same list twice (for example, because you enter the server once using the domain name and again as using the internet address) ISS eliminates the duplication and tells you how many distinct servers are in the pool.

This keyword is required for **CUSTOM** and **ROUNDROBIN**. It is ignored when used with the **LOADLEVELER** metric.

Configure the TCP/IP nameserver

This section describes how to set up the TCP/IP nameserver part of the ISS system.

Select an appropriate configuration: First, you need to designate a machine to act as the nameserver for the domain which contains the names of the ISS pools. The choice of machine and the definition of the domain depend on whether there is an existing nameserver in use and the size of the domain which it serves. Refer to the following table to help you decide which machine in your network to use to act as the nameserver:

If these circumstances apply:	Then:
Your organization already has a nameserver in use	You can add the pool servers to a domain served by this existing nameserver or you can add them as an additional sub-domain which that machine will serve.
Your organization either does not have an existing nameserver in use or wants to use a different machine to act as the nameserver. You might want to use this alternative machine in order to isolate the ISS nameserver from the site nameserver.	You can add the pool servers as a new sub-domain to this new nameserver and modify the client machines so that they go in the first instance to the new nameserver. You could also add the pool servers as a new sub-domain to this new nameserver and have an existing nameserver pass on requests for the new sub-domain. This approach is best suited for large sites because if your site has a large number of users, it may not be appropriate to add the ISS pools to the site nameserver because ISS causes the nameserver to reload its database quite frequently. For large named data files the overhead of doing this could be prohibitive. In such sites the pool servers should be added to a separate sub-domain, and this sub-domain should be administered by a separate nameserver machine. This does not necessarily mean that you have to modify the many client workstations' **/etc/resolv.conf** files, since the site nameserver can be configured to pass on anything for the newly introduced ISS sub-domain. This allows the client workstations to remain unchanged, directing all requests for name resolution to the site nameserver.

"Configure a New Nameserver" describes how to set up a sub-domain for the pool names and how to introduce a machine to act as a nameserver for that sub-domain. In addition, these instructions illustrate how to modify the client workstations so that they use the new nameserver and how to set up the new nameserver to pass requests for names outside the new sub-domain to an existing nameserver.

If you do not want to use this approach to setting up a pool server, you can still follow these instructions but you can ignore the section on modifying the client machines.

If you are using an existing nameserver, you do not have to set up a sub-domain, and you can add the pool names to the existing nameserver's data file.

If you will be adding the pool servers as a new sub-domain to a new nameserver and you want to use an existing nameserver to pass on requests for the new sub-domain, you also need to add an entry to the existing nameserver so that it passes on requests for the new sub-domain to the new nameserver.

Configure a New Nameserver: The TCP/IP nameserver is set up to have authority over a sub-domain of the main domain to which the server machines, the nameserver machine, and the client machines belong. For example, you might use the sub-domain **sp.site.corp.com** to denote the domain of authority of the ISS system, with all client machines and the nameserver machine and the pool machines all being configured as part of the main domain **site.corp.com**. Note that the pool machines are not designated as being part of the **sp.site.corp.com** sub-domain. It is not necessary to change the server machines in order for ISS to provide load-leveling over those machines.

There are three files which determine the behavior of the TCP/IP nameserver:

- Boot File

This file is normally called the **named.boot** file. You can supply the name of this file as a command line parameter when the **named** daemon is started. If you don't supply the name, then the **named** daemon uses the /**etc**/**named.boot** file. Use the root id to copy the ISS sample file provided to you and modify it. Take care not to destroy any existing **named.boot** file. There are three essential lines in this file which you have to configure according to the particular environment in which you will use the ISS system:

```
domain  sp.site.corp.com
primary sp.site.corp.com        /u/loadl/named.data
primary in-addr.arpa            /u/loadl/named.rev
```

The **sp.site.corp.com** string is the sub-domain that we are creating for the server machines. You should modify the pathnames to the files **named.data** and **named.rev** to match the path that has been chosen for installing the ISS files. For this example, the LoadLeveler home directory, /**u**/**loadl**, has been used. The sub-domain name string should be set correctly, with an appropriate sub-domain name, and the main domain to which the client and server machines belong.

- Reverse Data File

  This file is normally called the **named.rev** file and it is identified in the boot file. You should copy the reverse data file to the chosen directory (for example, /**u**/**loadl**). Its full pathname must appear in the boot file.

  The main purpose of this file is to ensure that reverse name resolution requests (dotted decimal to machine name conversion) which are outside the domain of the ISS nameserver are passed on to the main site nameserver for resolving. For example, you might want anything which begins with “1.2.”, but which has not been resolved by the ISS nameserver, to be resolved by the local site nameserver; so you would have the following line:

  ```
  2.1     9999999 IN      NS      nameserver.site.corp.com.
  ```

  In the Start of Authority (SOA) stanza of this file, the ISS nameserver machine name is specified, including the sub-domain name. Note that the ISS nameserver machine is not itself configured (in its TCP/IP configuration) as part of the sub-domain, we are just telling the **named** daemon that it is for the benefit of ISS. You should change this line as appropriate to match the local environment.

- Data File

  This file is normally called the **named.data** file. You should copy this file to the chosen directory (for example, /**u**/**loadl**). Its full pathname must appear in the boot file.

  The purpose of this file is to tell the **named** daemon what dotted-decimal internet address to return in response to a request to convert a machine name into an internet address (forward resolution). This is the file in which you declare the names for the pools over which ISS is presiding. ISS searches only the data file for the names of ISS pools. ISS does not search any other files specified by **$INCLUDE** statements in the data file.

  The data file also contains the information to permit hierarchical name resolution. Any name which the ISS name daemon cannot resolved is passed to the site nameserver.

  You need to modify several lines in this file. In this example, the name of the machine on which the nameserver and ISS are running is `poolserver`:

```
@         9999999 IN      SOA     poolserver.sp.site.corp.com.
                                  root.poolserver.sp.site.corp.com. (
```

You should adjust the entries in the SOA stanza to match the local environment, and the name of the ISS nameserver machine and the sub-domain name.

```
            9999999 IN      NS      poolserver
```

This Name Server (NS) stanza has the machine name of the ISS nameserver machine. Update this stanza if necessary to reflect the correct name.

The next two stanzas indicate the way that unresolved name resolution requests are passed up to the nameserver with authority over the domain of which the ISS system is a sub-domain.

```
site.corp.com.              9999999 IN   NS   nameserver.site.corp.com.
nameserver.site.corp.com.   9999999 IN   A    1.2.3.4
```

The following lines declare the names of the pools associated with the groups of servers over which ISS has control. The names are given without their sub-domain name and the internet address given should initially be set to the address of one of the machines in the pool.

```
tpserver     9999999    IN  A    1.2.3.4
dbclust      9999999    IN  A    1.2.3.4
```

Once you edit all of these files, you can start the name daemon on the ISS nameserver machine with the following command (issued by the root user):

```
/etc/named
```

or:

```
startsrc -s named
```

Configure the Client Machines

If you are not setting up a sub-domain for the pool servers and you don't want to introduce a new machine to act as a nameserver for that sub-domain and if you do not need to modify client machines to interact with the new nameserver, you can ignore this section. If your selected approach dictates that you need to configure the client machines, follow the instructions presented in this section. These instructions illustrate how to set up the TCP/IP configuration on the the client machines so that the ISS nameserver is the main nameserver and the site nameserver is the backup nameserver. You can do this using SMIT for AIX platforms. For other platforms, use any appropriate configuration tool. Using SMIT, select these choices:

- Communications Applications and Services
- TCP/IP
- Further Configuration
- Name Resolution
- Domain Nameserver (**/etc/resolv.conf**)
- Add a Nameserver.

When you are finished, ensure that the **/etc/resolv.conf** has the nameserver entries in the following order:

1. The ISS nameserver machine should be first.
2. The site nameserver machine should be second.

The ISS nameserver is used for name resolution under normal conditions. If the ISS nameserver does not respond, then the backup nameserver is used to perform name resolution. Note that when the backup nameserver is being used performance may be affected.

To test the operation of the ISS nameserver, make a name resolution request to the nameserver from a client machine. For example, for the name **myserver**, type:

```
host myserver
```

This returns the dotted decimal internet address for myserver specified in the **named.data** file.

Configure an ISS Backup

You can run two instances of ISS, with one serving as a backup. The site nameserver can use both instances of ISS. Under normal circumstances, the Site Nameserver uses the first instance of ISS. If the first instance of ISS fails for any reason, the site nameserver uses the second ISS instance. When the first instance of ISS recovers, the site nameserver switches to the first instance.

It is recommended that you use primary servers for both instances rather than one primary server and one secondary server. Each instance of ISS should be run on a separate machine. If you are running ROUNDROBIN or LOADLEVELER configurations, no changes are required to the ISS configuration files.

However, if you are running a CUSTOM configuration, some changes are necessary. **issgent** is exclusive and therefore cannot be run twice on the same server. The two instances of **issagent** must use different names and different port numbers.

For example, suppose you are running two instances of ISS, called **ISSone** and **ISStwo**. The configuration for **ISSone** should include: **ISS_PORT 10001**. The configuration for **ISStwo** should include: **ISS_PORT 100021**. In addition, the **/etc/services** and **/etc/inetd.conf** files need to be altered as follows:

```
/etc/services  :  issagent1    10001/tcp
                  issagent2    10002/tcp
/etc/inetd.conf:
        issagent1 stream tcp nowait root path/issagent1 issagent1
        issagent2 stream tcp nowait root path/issagent2 issagent2
```

issagent1 and **issagent2** can be symbolically linked to the original **issagent**, or you can copy the executable.

Step 4: Reconfiguring ISS While it is Running

To reconfigure ISS while it is still running, edit the configuration file that was used when ISS was started and send a SIGHUP signal to the **issmonitor** process, as follows:

```
kill -HUP issmonitor_pid
```

where *issmonitor_pid* is the process ID of the **issmonitor** process, which can be determined by issuing the following command:

```
ps -ef | grep issmonitor
```

ISS then re-reads the configuration file and alters its behavior in accordance with the new pool definitions provided. This causes ISS to temporarily suspend leveling of TCP/IP connections but does not prevent new connections to pools from being established. All such requests received during reconfiguration are resolved to the machine address which was current for each pool when the above command was issued. Reconfiguration is very quick and should not be noticeable to users.

Pool names which are deleted from the configuration file during reconfiguration (with ISS either running or stopped) will still be present in the nameserver datafile but will not be actively leveled. They will always resolve to the machine address which was current for that pool when the configuration was changed.

Step 5: Starting ISS

You can run ISS in the foreground or background and you can run it using `nohup` if desired. You can start ISS after ensuring that the TCP/IP **named** daemon is running and is correctly resolving names from the client machines. If it isn't started then ISS will produce an error message and exit. If this occurs check the **named** daemon configuration, start the **named** daemon and try again. To start ISS, as root, enter the following:

issmonitor *filename*

where *filename* is the name of the ISS configuration file. ISS prints an opening message. If it does not find any errors in the ISS configuration file, it begins to transparently level TCP/IP connections across the pools defined in the configuration file. If there are problems with the pool definitions in the configuration file, a message is generated and ISS exits. If this occurs, modify the configuration file and try again. Refer to the sample configuration file for guidance.

Step 6: Establishing an Interactive Session

An interactive session can be established with any standard AIX TCP/IP application such as **telnet, rsh, ftp, or rlogin**. When the application expects you to provide the machine name or internet address of the remote host to which you want to connect, enter instead the name of the pool of machines to which you want to connect. For example:

`telnet` *dept_pool*

where *dept_pool* is the name of the ISS defined pool. By using *pool name* the user is automatically connected to a least-loaded node in that pool, using the leveling metric defined in the configuration file.

Applications that specify TCP/IP machine names or internet addresses for client server connections should specify the *pool name* in their TCP/IP configuration files.

Depending on the way your site nameserver is set up and the relationship between the TCP/IP domains in which the client and server machines are located, you may need to specify the pool name more fully. For example, *dept_pool.sp* or even *dept_pool.sp.site.corp.com.*. This may be necessary if your client is served by AFS.

Step 7: Ending an Interactive Session

Interactive sessions are ended in the same manner as they would be in the absence of ISS. For example, to end an ftp session, type **quit** at the ftp prompt or to end a telnet session type the escape character followed by **quit**.

Step 8: Stopping ISS

To stop ISS, issue a SIGINT signal. For example:

```
kill -INT issmonitor_pid
```

where *issmonitor_pid* is the process ID of the **issmonitor** process, which can be determined by issuing the following command:

```
ps -ef | grep issmonitor
```

It is recommended that you shut down using SIGINT (as shown above) rather than using SIGKILL via **kill -9**. When you stop ISS by issuing **kill -9**, the monitor cannot perform a graceful clean up, and the agents must wait for their periodic check on the status of the monitor before they can clean up. This is because agents minimize the traffic they send to the monitor if the load on a server is stable. The agents still check periodically that the monitor is up, but this checking is done less frequently when loads are stable.

You can also kill an **issagent** (either gracefully with **kill -INT** or with **kill -9**), but if the server is configured to a CUSTOM pool, the monitor, if still running, will start a new instance of the **issagent** on the server.

When ISS is not running, access requests directed to the pool names will be resolved to the machine address which was current for each pool when ISS was terminated.

Troubleshooting

Using the ISS trace facilities

If ISS does not appear to be working as you would expect, then try running it with its trace facilities enabled.

Monitor Trace

You can enable ISS tracing on the monitor by inserting the **MONITOR_TRACE** keyword into the ISS configuration file. The position of the keyword in the file is not important, except that it should be on a line by itself and should be at the start of that line. Tracing will start as soon as ISS encounters the keyword in the file, so if you wish to see what happens when ISS parses your configuration file, put it at the top. If you just want tracing of events following the parsing of the configuration file then you could put it at the bottom.

Setting this keyword in the configuration file causes ISS to write trace messages to stdout. Examine the trace that is produced and compare it with the problems described in this section.

An example of the use of the **MONITOR_TRACE** keyword in the configuration file follows:

```
MONITOR_TRACE

NAME    My_Pool
METRIC  LOADLEVELER
CLASS   No_Class
 ..etc..
```

Note that ISS can be reconfigured while running using SIGHUP. Therefore, you can add the **MONITOR_TRACE** keyword to the configuration file and issue:

```
kill -HUP issmonitor_pid
```

in order to turn ISS trace on. You can turn it off by removing **MONITOR_TRACE** from the configuration file and issuing a further SIGHUP.

Agent Trace

If you are using **CUSTOM** metrics in any of your pools and you suspect that something is wrong with such a pool, then you can turn on ISS tracing at the servers by inserting the **AGENT_TRACE** keyword into the ISS configuration file. The position of the keyword in the file is not important, except that it should be on a line by itself and should be at the start of that line. Tracing will start at the agents when ISS configures them into a pool. This occurs soon after the agent is started. Refer to the descriptions of **AGENT_TRACE** and **AGENT_LOG** for details of how to capture the agent trace output, and how to limit its overall size.

Setting this keyword in the configuration file causes issagent to write trace messages to a log file you specify. Examine the trace that is produced and compare it with the problems described in this section.

An example of the use of the **AGENT_TRACE** keyword in the configuration file follows:

```
AGENT_TRACE
AGENT_LOG /tmp/myagent.log

NAME    My_Pool
METRIC  CUSTOM
SERVERS first second third
 ..etc..
```

Note that ISS can be reconfigured while running using SIGHUP. Therefore, you can add the **AGENT_TRACE** keyword to the configuration file and issue:

```
kill -HUP issmonitor_pid
```

on the monitor in order to turn ISS agent tracing on. You can turn it off by removing **AGENT_TRACE** from the configuration file and issuing a further SIGHUP on the monitor.

Problem: All interactive sessions are assigned to one machine

If this situation occurs, it may be for one of the following reasons:

- One machine genuinely has a consistently light load relative to other machines in the pool. This may be because other machines in the pool are supporting some other activity, which is not under the control of LoadLeveler or ISS.
- There may be a problem with ISS configuration or file permissions, which is preventing ISS from operating satisfactorily.

If this condition does arise, try measuring the loads on the servers manually, and verify that ISS is failing to pick the best one.

Check that the load on the machines is not skewed to the extent that the metric being used to level load across the pool would genuinely result in the same machine being nominated every time. You can do this by looking at the amount of idle time each CPU is experiencing (using **vmstat** or **sar**), or by looking at the list of active processes (using **ps**). You can test a CUSTOM metric by manually running it on the server. For a LoadLeveler pool, you can use **llstatus** or **xloadl** to test the availability of machines which support the Class, Requirements and Preferences defined for the pool.

Also, turn on **MONITOR_TRACE** to determine what ISS is doing. See “Using the ISS trace facilities” on page 156 for details.

Problem: llinitiate: configuration error - no default group

This error message occurs if there is no stanza for the **loadl** user in the **LoadL_admin** file, or if there is a stanza but it has no default group assignment.

Ensure that such a user stanza exists and that the default group is set.

Problem: ISS trace shows "llinitiate()" and nothing further happens

This problem is caused by LoadLeveler not responding to ISS. This can happen if, for example, the **LoadL_admin** file has been changed. The LoadLeveler configuration needs to be refreshed by using the **llctl reconfig** command.

Problem: ISS trace shows "Cannot rename 'file 1' to 'file 2'. Permission denied"

Ensure that ISS is being run by root, and check the permissions on the files named in the error message.

Problem: Agents do not get started on servers.

If you are running a **CUSTOM** metric, (which is the only type that uses agents), and you don't see agents running on the servers, or see messages in the MONITOR_TRACE output that indicate that the monitor is having to repeatedly try and start agents, then check that the servers on which the agents fail to start are configured correctly.

Check that the **/etc/services** and **/etc/inetd.conf** files contain the information described in the description of the **METRIC CUSTOM** keyword, and ensure that **inetd** has been refreshed. You can do this, as root, by typing:

```
refresh -s inetd
```

Check that you can connect to the server's port configured in **/etc/services**, and check that the metric can be run on the server. Turn on **AGENT_TRACE** output and look at the log file it produces. Note that the agents are only informed of the location of the log file when the agents have been successfully started. So, the log file only shows what happens after the servers are told which pools they are in and what the merics are. The agent log file is therefore useful for the latter stages of agent start up. Refer to the **MONITOR_TRACE** output to investigate why agents are stopping. In the **MONITOR_TRACE** output, look for the reason code each

agent sends back to the monitor when the agent stops. One such reason code is "Agent could not run metric".

ISS Configuration File

The following is the ISS configuration file called **Iss_config** which is found in the install directory /**usr**/**lpp**/**LoadL**/**iss**:

```
# Sample Iss_config file
# ======================
#
# The following keywords and values control the overall behaviour of ISS.
# They affect all pools. MONITOR_TRACE turns on trace level output on the
# monitor machine if present. All output on the monitor machine is produced
# via stdout. AGENT_TRACE turns on trace level output on all servers that are
# in custom pools, and thus run an agent. If AGENT_TRACE is specified then
# AGENT_LOG must be present and provide the name of a log file on the agent
# machines. There is an optional second parameter to AGENT_LOG which limits
# the size to which the log file can grow. When the log file reaches that
# size it is moved to <filename>.old, and a new log file is created. ISS_PORT
# specifies the port number on which the servers in custom pools (which will
# run agents) have issagent configured.

MONITOR_TRACE
AGENT_TRACE
AGENT_LOG /tmp/issagent.log 10000
ISS_PORT 10001


# pool1 is a custom pool whose domain name is stored in /etc/named.data and
# /etc/named.rev. It has three servers, whose domain names are first..third.
# It measures the number of processes running on each of these servers once
# every 2*HEARTBEAT_INTERVAL seconds, and HEARTBEAT_INTERVAL is configured
# to be 30 secs during most of the day, but only 10 secs between 8am and 10am.
# Once every HEARTBEAT_INTERVAL seconds ISS checks that the server whose
# address is currently recommended is functional at its ICMP layer. The
# servers in the list provided are load capped at 75 (processes) and if all
# servers first..third reach that limit then the address of server fourth
# is used. If at any time there are no servers available, the name pool1
# will be removed from the DNS files and the pool alarm triggered, which in
# this case makes an entry into a file called /tmp/issalarm.log
NAME                    pool1.sp.site.corp.com
NAMED_DATAFILE          /etc/named.data
NAMED_REVERSEFILE       /etc/named.rev
METRIC                  CUSTOM ps -ef | wc -l
POLICY                  MIN
SERVERS                 first second third
HEARTBEAT_INTERVAL      0(30) 8(10) 10(30)
HEARTBEATS_PER_UPDATE   2
LIMIT                   75 fourth
POOL_ALARM echo "pool1 ran out of servers at `date`" >> /tmp/issalarm.log
```

```
# pool2 is a roundrobin pool whose domain name is stored in /etc/named.data
# and /etc/named.rev. It has six servers, whose domain names are first..sixth.
# It cycles the address associated with the domain name "pool2" once per
# minute, and every 20 seconds checks that the server whose address is
# currently recommended is functional at its ICMP layer. If at any time there
# are no servers available, the name pool2 will be removed from the DNS files
# and the pool alarm triggered, which in this case makes an entry into a file
# called /tmp/issalarm.log
NAME                    pool2.sp.site.corp.com
NAMED_DATAFILE          /etc/named.data
NAMED_REVERSEFILE       /etc/named.rev
METRIC                  ROUNDROBIN
SERVERS                 first second third fourth fifth sixth
HEARTBEAT_INTERVAL      20
HEARTBEATS_PER_UPDATE   3
POOL_ALARM echo "pool2 ran out of servers at `date`" >> /tmp/issalarm.log


# pool3 is a loadleveler pool whose domain name is stored in /etc/named.data
# and /etc/named.rev. It specifies a loadleveler class Iss_Class, which
# should be configured on all servers that the pool may include. It also
# specifies requirements and preferences as configured into loadleveler to
# allow the mapping of pool3 onto servers to be controlled. It ascertains
# the best server to use (by asking LoadL) once every minute, and every 30
# seconds checks that the server whose address is currently recommended is
# functional at its ICMP layer. If at any time there are no servers available,
# the name pool3 will be removed from the DNS files and the pool alarm
# triggered, which in this case makes an entry into a file called
# /tmp/issalarm.log
NAME                    pool3.sp.site.corp.com
NAMED_DATAFILE          /etc/named.data
NAMED_REVERSEFILE       /etc/named.rev
METRIC                  LOADLEVELER
CLASS                   Iss_Class
REQUIREMENTS            (Arch == "R6000")
PREFERENCES             (Feature == "data_server"}
HEARTBEAT_INTERVAL      30
HEARTBEATS_PER_UPDATE   2
POOL_ALARM echo "pool3 ran out of servers at `date`" >> /tmp/issalarm.log

# end of sample config file
```

Chapter 7. 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 following keywords in the configuration file:

JOB_ACCT_Q_POLICY = *number*
where *number* is the amount of time in seconds that determines how often the startd daemon updates the schedd daemon with accounting data of running jobs. This controls the accuracy of the **llq -x** command. The default is 300 seconds.

JOB_LIMIT_POLICY = *number*
where *number* is an amount of time in seconds. The smaller of **JOB_LIMIT_POLICY** and **JOB_ACCT_Q_POLICY** is used to control how often the **startd** daemon collects resource consumption data on running jobs, and how often the **job_cpu_limit** is checked. The default for **JOB_LIMIT_POLICY** is **POLLING_FREQUENCY** multiplied by **POLLS_PER_UPDATE**.

Collecting Job Resource Data Based on Machines

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

To enable recording of resources by machine, you need to specify the following in the LoadLeveler configuration file.

ACCT = A_ON A_DETAIL

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 78.

© Copyright IBM Corp. 1993, 1996

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 capture -g eventname
```

eventname is any string of continuous characters (no whitespace 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 capture -g graveyard
```

Using this command allows you to obtain a snapshot of all data up to and including that moment in time. If you wanted to capture this type of information on a regular basis, you could set up a crontab entry to invoke this command regularly. For example:

```
#  sample crontab for accounting
#  shift crontab 94/8/5
#
# Set up three shifts, first, second, and graveyard shift.
#  Crontab entries indicate the end of shift.
#
#M  H d m day command
#
00 08 * *  * /u/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 193. For more information on the collection of accounting records, see "llq - Query Job Status" on page 207.

Collecting Job Resource Information Based on User Accounts

If your installation is interested in keeping track of resources used on an account basis, you can require all users to specify an account number in their job command files. They can specify this account number with the **account_no** keyword which is explained in detail in "Job Command File Keywords" on page 40.

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

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

- User name
- User's login group name
- Account number specified on the Job

- 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 7: Define Job Accounting” on page 106.
- 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.YYMMDDHHmm*. You may specify the directory in which to place the file when you invoke the **llacctmrg** command or you can specify the directory with the **GLOBAL_HISTORY** keyword in the configuration file. The default value set up in the sample configuration file is the local spool directory:

GLOBAL_HISTORY = *$(SPOOL) (optional)*

Accounting Reports

You can produce three types of reports using either the local or global history file. These reports are called the *short*, *long*, and *extended* versions. As their names imply, the short version of the report is a brief listing of the resources used by LoadLeveler jobs. The long version provides more comprehensive detail with summarized resource usage and the extended version of the report provides the comprehensive detail with detailed resource usage. If you do not specify a report type, you will receive the default short version.

The short report displays the number of jobs along with the total CPU usage according to user, class, group, and account number. The extended version of the

report displays all of the data collected for every job. See the **llsummary** command, "llsummary - Return Job Resource Information for Accounting" on page 221, for examples of the short and extended versions of the report.

For information the accounting Application Programming Interfaces, refer to Appendix C, "Sample Files" on page 313.

Sample Job Accounting Scenario

The following sample scenario walks you through the process of collecting account data. You can perform all of the steps or just the ones that apply to your situation.

Task 1: Update the Configuration File

Edit the configuration file according to the following table:

Edit this keyword:	To:
GLOBAL_HISTORY	Specify a directory in which to place the global history files.
ACCT	Turn accounting and account validation on and off and specify detailed accounting.
ACCT_VALIDATION	Specify the account validation routine.
Note: See "Step 7: Define Job Accounting" on page 106 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**: eee "llacctmrg - Collect machine history files" on page 186 for the syntax of this command.
- Using the graphical user Interface:

  Select A machine from the Machines window

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

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

  Press **OK**

  ▲ The window closes and you return to the main window.

Task 3: Report Job Information on all the Jobs in the History File

You can accomplish this step using either the **llsummary** command or the graphical user interface:

- Using **llsummary**: see "llsummary - Return Job Resource Information for Accounting" on page 221 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 "Job Command File Keywords" on page 40 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 the machines, specify the following keyword in the machine's **machine stanza** in the administration file:

speed = *number*
where *number* defines the weight. The higher numbers correspond with a greater weight. The default weight is 1.0.

Chapter 8. Error Logging Facility

LoadLeveler performs error recording by using the AIX error logging facility. **You must run LoadLeveler on AIX Release 4.1 in order to use this facility.**

In addition, LoadLeveler daemons and processes also keep log files. For more information on these LoadLeveler log files, see "Step 10: Record and Control Log Files" on page 110.

Error Logging Overview

Error logging is the writing of information to persistent storage to be used for debugging purposes. This type of logging is for subsystems that perform a service or function on behalf of an end user. The subsystem does not communicate directly with the end user and, therefore, needs to log events to some storage location. The events that are logged are primarily error events.

LoadLeveler error logging uses AIX error log facilities to report events on a per machine basis. The intent is to have the AIX error log be the starting point for diagnosing system problems. For more information on AIX error log facilities, see *IBM AIX Version 4.1 Problem Solving Guide and Reference*, SC23-2606, and *IBM General Concepts and Procedures for RISC System/6000*, GC23-2202. For more information on Error Log facilities available with the SP, see *IBM RS/6000 Scalable POWERparallel SP Diagnosis and Messages Guide*, SC23-3866.

Error log entries include a "`DETECTING MODULE`" string that identifies software component, module name, module level, and line of code or function that detected the event that was logged. The information is formatted depending on the logging facility the user is viewing. For example the AIX Error Log facility information appears:

```
DETECTING MODULE
LPP=LPP name Fn=filename SID_level_of_the_file L#=Line number
```

How Error Log Events Are Classified

The following table shows the mapping of LoadLeveler error log suffixes to syslog priorities and AIX Error Log error types.

Table 8. LoadLeveler Error Label Suffixes Mapped to AIX Error Log

Error Label Suffix	AIX Error Log Error Type	AIX Error Log Description
EM	PEND	The loss of availability of a device is imminent.
ER	PERM	No recovery from this condition. A permanent error occurred.
ST	UNKN	It is not possible to determine the severity of the error.
TR	UNKN	It is not possible to determine the severity of the error.
RE	TEMP	Condition was recovered after several unsuccessful attempts.
DE	UNKN	It is not possible to determine the severity of the error.

© Copyright IBM Corp. 1993, 1996

What Events are Recorded

The following events are recorded by the LoadLeveler error logging facility:

- Errors occurring during the initial start up of LoadLeveler daemons
- Installation and customization errors
- The restarts per hour limit being exceeded
- **LoadL_master** detecting the crash of a daemon
- Changes in the state of a daemon
- LoadLeveler truncating the log files

Viewing LoadLeveler Error Log Reports

Enter the following command to view the LoadLeveler error reports for a machine:

```
errpt -a -N LoadLeveler
```

Clearing All LoadLeveler Error Log Entries

Enter the following command to clear all LoadLeveler entries from the error logs of a machine:

```
errclear -N LoadLeveler 0
```

Error Notification

You can be notified of a LoadLeveler error when it occurs by using the AIX Error Notification Facility. This facility will perform an ODM method defined by the administrator when a particular error occurs or a particular process fails. *IBM General Concepts and Procedures for RISC System/6000* (GC23-2202) explains how to use the AIX Error Notification Facility.

Sample Error Log

The following is a sample LoadLeveler error report:

```
LABEL:          LL_TRUNCATE_ST
IDENTIFIER:     05D868E9

Date/Time:       Fri Mar 31 18:07:43
Sequence Number: 271
Machine Id:      000001801000
Node Id:         ll2
Class:           S
Type:            UNKN
Resource Name:   LoadLeveler

Description
SOFTWARE

Probable Causes
APPLICATION PROGRAM

User Causes
FILE NEEDS REORGANIZATION

        Recommended Actions
        NO ACTION NECESSARY

Detail Data
DETECTING MODULE
LPP=LoadL,Fn=dprintf_config.c,SID=1.2,L#=392,
DIAGNOSTIC EXPLANATION
Logfile "/u/loadl/log/StartLog" was truncated
```

Figure 31. Sample LoadLeveler Error Log

Possible Causes for LoadLeveler Errors

Table 9 describes the possible causes for some common problems reported by the error logging facility:

Table 9 (Page 1 of 6). Possible Causes for LoadLeveler Failure

Label: LL_INIT_ER
Error Type: PERM
Diagnostic Explanation: LoadL_master must be run as root

Explanation: The ownership and the permissions of **LoadL_master** file are incorrect

Cause: This error ocurrs for one of the following reasons:

- The **LoadL_master** file is not owned by root
- The **LoadL_master** file does not have Set User ID permission (the “s” bit)

Action: Perform the following actions:

- Issue **chown root LoadL_master**
- **chmod u+s LoadL_master**
- Restart LoadLeveler

Table 9 (Page 2 of 6). Possible Causes for LoadLeveler Failure

Label: LL_INIT_ER
Error Type: PERM
Diagnostic Explanation: master: could not set the real uid to ROOT, rc=nn

Explanation: Unable to set the real Uid to 0 (root).
Cause: The setuid system call failed for an authorized caller.
Action: Contact the system administrator

Label: LL_INIT_ER
Error Type: PERM
Diagnostic Explanation: Error setting group id to nn. errno = nn.

Explanation: **LoadL_master** was unable to set the group ID.
Cause: The setguid() call failed.
Action: Verify that the following are true:

- **LoadLGroupid** is defined correctly in **/etc/LoadL.cfg**
- The group ID reported in the error description is defined on the system.

Label: LL_START_ER
Error Type: PERM
Diagnostic Explanation: LoadLeveler cannot continue; this host not in machine list

Explanation: You attempted to start LoadLeveler on a machine that is not defined in the **LoadL_admin** file, and MACHINE_AUTHENTICATE = TRUE is set in the **LoadL_config** file.
Cause: See "Explanation."
Action: Do one of the following:

- Define the machine in the **LoadL_admin** file
- Set MACHINE_AUTHENTICATE = FALSE in the **LoadL_config** file

Then, restart LoadLeveler on the machine where the problem occurred.

Label: LL_START_ER
Error Type: PERM
Diagnostic Explanation: Cannot start LoadLeveler. Must be the designated LoadLeveler administrator

Explanation: A user other than the user defined as the LoadLeveler administrator attempted to start LoadLeveler.
Cause: The **LOADL_ADMIN** keyword does not contain the user that attempted to start LoadLeveler.
Action: Add the user to the **LoadL_admin** file.

Table 9 (Page 3 of 6). Possible Causes for LoadLeveler Failure

Label: LL_START_ER
Error Type: PERM
Diagnostic Explanation: START_DAEMONS flag was set to *value*. Exiting.

Explanation: **LoadL_master** could not start the LoadLeveler daemons.
Cause: The **START_DAEMONS** keyword was not defined or was set to False.
Action: Set START_DAEMONS = true.

Label: LL_START_ER
Error Type: PERM
Diagnostic Explanation: Cannot get address of *hostname* from nameserver; LoadLeveler not starting.

Explanation: LoadLeveler could not resolve the hostname of the machine defined as the central manager.
Cause: gethostbyname() failed.
Action: Follow standard procedures for resolving nameserver problems.

Label: LL_INSTL_CUST_ER
Error Type: PERM
Diagnostic Explanation: *keyword* not specified in config file. Admin file not specified in config! Cannot continue.

Explanation: The specified keyword is not defined correctly.
Cause: This error occurs for one of the following reasons:
- The keyword value is equal to a blank
- The keyword is misspelled
- The keyword is commented out
- The keyword is missing

Action: Define the keyword correctly.

Label: LL_INSTL_CUST_ER
Error Type: PERM
Diagnostic Explanation: *keyword*=*value* can not execute

Explanation: LoadLeveler was unable to execute the specified keyword (binary).
Cause: This error occurs for one of the following reasons:
- The binary does not have execute permission
- The binary name was misspelled
- The binary does not exist

Action: Correct the problem and reissue the keyword.

Table 9 (Page 4 of 6). Possible Causes for LoadLeveler Failure

Label: LL_INSTL_CUST_ER
Error Type: PERM
Diagnostic Explanation: Cannot open file *filename*

Explanation: LoadLeveler was unable to open the specified file.
Cause: This error occurs for one of the following reasons:
- Incorrect permissions are set for the file
- The file does not exist

Action: Correct the problem with file.

Label: LL_RESTART_ER
Error Type: PERM
Diagnostic Explanation: Exceeded nn restarts / hour

Explanation: **LoadL_master** exceeded the maximum allowed restarts for a daemon, as defined by the **RESTARTS_PER_HOUR** keyword.
Cause: The **LoadL_master** daemon died after exceeding the **RESTARTS_PER_HOUR** limit.
Action: Check the LoadLeveler error logs and the administrator mail for more information.

Label: LL_CRASH_ER
Error Type: PERM
Diagnostic Explanation: The *daemon_name* (process *pid*) died

Explanation: The specified daemon died.
Cause: This error occurs for one of the following reasons:
- The daemon received a signal
- The daemon experienced an unrecoverable error

Action: Check the LoadLeveler error logs for more information.

Label: LL_INFO_ST
Error Type: UNKN
Diagnostic Explanation: Started *daemon_name*, pid and pgroup=*pid*

Explanation: The specified daemon started.
Action: None. This is an informational message.

Table 9 (Page 5 of 6). Possible Causes for LoadLeveler Failure

Label: LL_INFO_ST
Error Type: UNKN
Diagnostic Explanation: Got SHUTDOWN command. Got RECONFIG command.

Explanation: **LoadL_master** received the specified commands.
Cause: The **llctl** command was issued with “stop” or “reconfig”.
Action: None. This is an informational message.

Label: LL_INFO_ST
Error Type: UNKN
Diagnostic Explanation: Alternate central manager *cent_manager* will become active central manager. Switching central manager to *cent_manager*.

Explanation: LoadLeveler performed central manager recovery.
Cause: The backup central manager was unable to contact the primary central manager.
Action: Check the LoadLeveler error logs and the administrator mail for more information.

Label: LL_INFO_ST
Error Type: UNKN
Diagnostic Explanation: Alternate central manager *cent_manager* will return to stand-by state.

Explanation: LoadLeveler performed central manager recovery.
Cause: The backup central manager gave control back to the primary central manager.
Action: None. This is an informational message.

Label: LL_INFO_ST
Error Type: UNKN
Diagnostic Explanation: *machine* is unable to serve as alternate central manager.

Explanation: The machine specified cannot be the alternate central manager because the appropriate negotiator keyword is not defined for this machine, or is defined incorrectly.
Cause: The specified machine was defined as the central manager.
Action: Define the appropriate keywords.

Table 9 (Page 6 of 6). Possible Causes for LoadLeveler Failure

Label: LL_TRUNCATE_ST
Error Type: UNKN
Diagnostic Explanation: Logfile *filename* was truncated.

Explanation: LoadLeveler truncated the specified log file because the file exceeded the size defined in the **LoadL_config** file.
Action: None. This is an informational message.

Chapter 9. Routing Jobs to NQS Machines

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

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

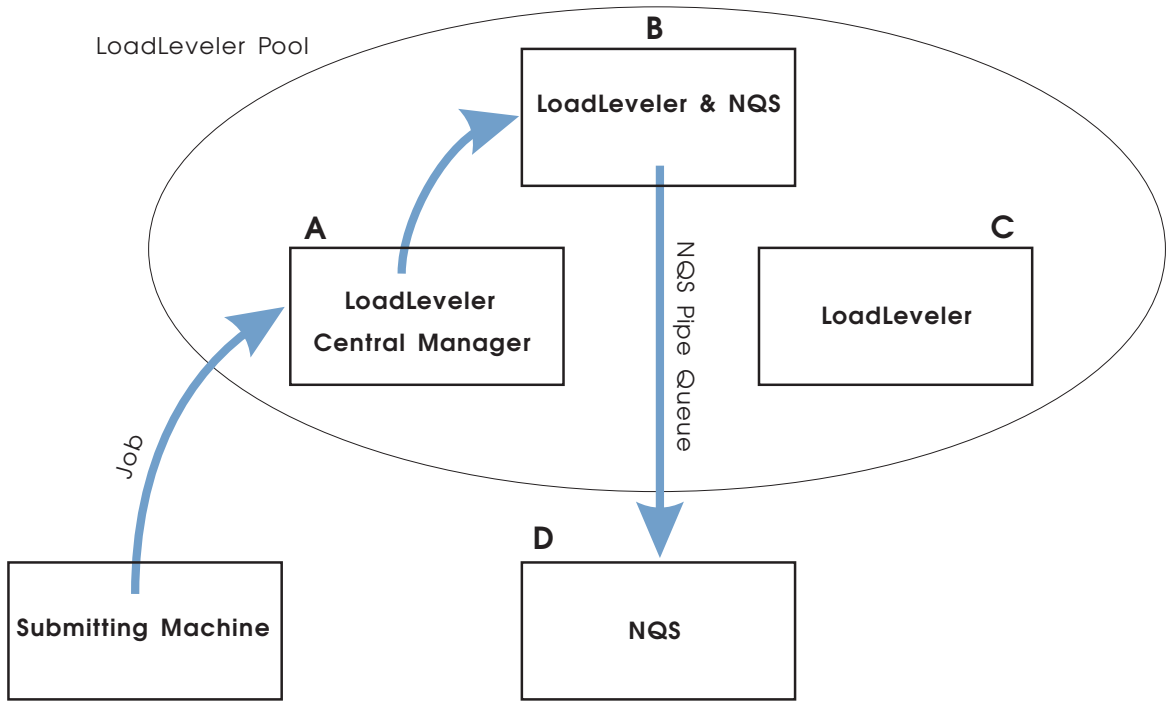


Figure 32. Environment illustrating jobs being routed to NQS machines.

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

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

Setting Up the NQS Environment

Setting up the NQS environment involves the following:

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

© Copyright IBM Corp. 1993, 1996

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**.

Sample Routing Jobs to NQS Machines Scenario

The following example walks you through the process of setting up your environment to route jobs to machines that run NQS.

Assume Figure 32 on page 177 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.

Task 1: Modify the Administration File

After setting up your NQS environment, modify the **LoadL_admin** file by defining the class **NQS** including the following stanzas:

```
NQS:
 type = class
 NQS_class = true
 NQS_submit = pipe_a
 NQS_query = queue@chevy.kgn.ibm.com
```

Task 2: Modify the Configuration File

Modify the **LoadL_config.local** on the machine(s) that you want to accept this class of jobs. In this example, you would modify machine B's **LoadL_config.local** file. To do this, add a class statement similar to:

```
CLASS = {"NQS" "a" "b" ....}
```

where `NQS` is the name of the class of jobs that will be routed to the machines that run NQS, and `a` and `b` are names of additional classes.

Task 3: Submit the Jobs

After you perform the previous tasks, users can route their jobs to machines running NQS using the **llsubmit** command. The job command file must specify the **class** keyword. For example:

```
class = NQS
```

The job command file must also contain the shell script to be submitted to the NQS node. NQS accepts only shell scripts, binaries are not allowed. All options in the command file pertaining to scheduling the job will be used by LoadLeveler to schedule the job. When the job is dispatched to the node running the specified NQS class, the LoadLeveler options pertaining to the runtime environment are converted to NQS options and the job is submitted to the specified NQS queue.

LoadLeveler command file options are used as follows:

arguments	error message generated and job not submitted
checkpoint	error message generated and job not submitted
class	used only for LoadLeveler scheduling
core_limit	converted to **-lc** option
cpu_limit	converted to **-lt** option
data_limit	converted to **-ld** option
environment	if COPY_ALL is specified, the option is converted to **-x**, otherwise error message generated and job not submitted
error	converted to **-e**
executable	error message generated and job not submitted
file_limit	converted to **-lf** option
hold	used only for LoadLeveler scheduling
image_size	error message generated and job not submitted
initialdir	error message generated and job not submitted
input	error message generated and job not submitted
notification	If the option specified is
	always converted to **-mb**and **-me** options
	error converted to **-me** option
	start converted to **-mb** option
	never ignored
	complete converted to **-me** option

notify_user	converted to **-mu** option
output	converted to **-o** option
preferences	used only for LoadLeveler scheduling
queue	places one copy of job in the LoadLeveler queue
requirements	used only for LoadLeveler scheduling
restart	If the option specified is
	yes ignored
	no converted to **-nr** option
rss_limit	converted to **-lw** option
shell	converted to **-s** option
stack_limit	converted to **-ls** option
start_date	used only for LoadLeveler scheduling
user_priority	used only for LoadLeveler scheduling

Users can also submit an NQS script. In this case, any NQS options in the script are used to schedule the job and once dispatched by LoadLeveler, the file is sent to NQS unmodified.

LoadLeveler schedules these jobs the same as it schedules other jobs. When the job is dispatched, LoadLeveler determines whether or not it is running in an NQS class. If it is, an NQS command **qsub** is issued.

LoadLeveler monitors the job by periodically invoking a **qstat** command. A **qstat** command is first issued for the pipe queue on the local host. If the request id is not found, a **qstat** is issued for each queue listed in the NQS_query class keyword. If the request id is still not found, starter marks the job as complete.

When a job is sent to an NQS class, **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 either queue. If LoadLeveler loses contact with NQS, a message is written to the shadow log file and the jobs associated with those queues will be considered complete.

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 the **NQS_submit** and **NQS_query** keywords.

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
ls	stack_limit
lt	cpu_limit
lT	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

Command Reference

© Copyright IBM Corp. 1993, 1996

Chapter 10. LoadLeveler Commands

LoadLeveler provides two types of commands: those that are available to all users of LoadLeveler, and those that are reserved for LoadLeveler administrators. (Administrators are identified by the **LOADL_ADMIN** keyword in the configuration file. Refer to "Step 1: Define LoadLeveler Administrators" on page 97 for more information.)

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

Summary of LoadLeveler Commands

The following table summarizes the LoadLeveler commands:

Command	Description	Who Can Issue?	For More Information
llacctmrg	Collects all individual machine history files together into a single file.	Administrators	See page 186
llcancel	Cancels a submitted job.	Users and Administrators	See page 188
llclass	Returns information about classes.	Users and Administrators	See page 190
llctl	Controls daemons on one or more machines in the LoadLeveler cluster.	Administrators	See page 193
llfavorjob	Raises one or more jobs to the highest priority, or restores original priority.	Administrators	See page 197
llfavoruser	Raises job(s) submitted by one or more users to the highest priority, or restores original priority.	Administrators	See page 199
llhold	Holds or releases a hold on a job.	Users and Administrators	See page 200
llinit	Initializes a new machine as a member of the LoadLeveler cluster.	Administrators	See page 203
llprio	Changes the user priority of a submitted job step.	Users and Administrators	See page 205
llq	Queries the status of LoadLeveler jobs.	Users and Administrators	See page 207
llstatus	Queries the status of LoadLeveler machines.	Users and Administrators	See page 214
llsubmit	Submits a job.	Users and Administrators	See page 219
llsummary	Returns resource information on completed jobs.	Administrators	See page 221

© Copyright IBM Corp. 1993, 1996

llacctmrg - Collect machine history files

Purpose

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

Syntax

llacctmrg [**-H**] [**-v**] [**-h** *hostlist*] [**-d** *directory*]

Flags

-H Provides a short usage message.

-v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

-h *hostlist* A blank delimited list of machines from which to collect data. The default is all machines listed in the **LoadL_admin** file.

-d *directory* 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 **LoadL_admin** file. To override the default, specify a machine or a list of machines using the **-h** flag.

When the **llacctmrg** command ends, accounting information is stored in a file called **globalhist.***YYMMDDHHmm*. Information such as the amount of resources consumed by the job and other job-related data is stored in this file. In this file:

YY indicates the year
MM indicates the month
DD indicates the day
HH indicates the hour
mm indicates the minute.

You can use this file as input to the **llsummary** command. For example, if you created the file **globalhist.9644301050**, you can issue **llsummary globalhist.9644301050** to record information on all machines.

Data on processes which fork child processes will be included in the file only if the parent process waits for the child process to end. Therefore, complete data may not be collected for jobs which are not composed of simple parent/child processes. For example, if a LoadLeveler job invokes an **rsh** command to execute some function on another machine, the resources consumed on the other machine will not be collected as part of the accounting data.

Examples

The following example collects data from machines named `mars` and `pluto`.

```
llacctmrg -h mars pluto
```

The following example collects data from the machine named `mars` and places the data in an existing directory called **merge**.

```
llacctmrg -h mars -d merge
```

Results

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

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

llcancel - Cancel a Submitted Job

Purpose

Cancels one or more jobs from the LoadLeveler queue.

Syntax

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

Flags

-H Provides a short usage message.

-v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

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

-u *userlist* Is a blank-delimited list of users. When used with the **-h** option, only the user's jobs monitored on the machines in the *hostlist* are cancelled. When used alone, only the user's jobs monitored by the machine issuing the command are cancelled.

-h *hostlist* Is a blank-delimited list of machine names. All jobs monitored on machines in this list are cancelled. 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 machine to which the job was submitted (delimited by dot). The default is the local machine.
- *jobid* is the job ID assigned to the job when it was submitted using the **llsubmit** command. The *jobid* is required.
- *stepid* (delimited by dot) is the step ID assigned to the job when it was submitted using the **llsubmit** command. The default is to include all members 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 cancelled.

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 cancelled.

Group administrators and class administrators cannot use the **-h** flag or the **-u** flag.

Description

When you issue **llcancel**, the command is sent to the negotiator. You should then use the **llq** command to verify your job was cancelled. A job state of RM (Removed) indicates the job was cancelled. A job state of RP (Remove Pending) indicates the job is in the process of being cancelled.

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.

Examples

This example cancels the job step 3 that is part of the job 18 that runs on the machine named bronze:

```
llcancel bronze.18.3
```

This example cancels all the job steps that are a part of job 8 that are running on 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 central manager on "gold"
```

llclass - Query Class Information

Purpose

Returns information about classes.

Syntax

llclass [**-H**] [**-v**] [**-l**] [*classlist*]

Flags

-H Provides a short usage message.

-v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

-l Specifies that a long listing be generated for each class for which status is requested. If **-l** is *not* specified, then the standard listing is generated.

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

If you have more than a few classes configured for LoadLeveler, consider redirecting the output to a file when you use the **-l** flag.

Examples

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

```
llclass -l silver gold
```

Results

The Standard Listing: The standard listing is generated when you do *not* specify **-l** with the **llclass** command. The following is sample output from the **llclass silver** command, where there are five `silver` classes configured in the cluster, with one `silver` class job currently running.

```
Name                 MaxJobCPU     MaxProcCPU  Free   Max Description
                    d+hh:mm:ss     d+hh:mm:ss Slots Slots

silver              0+00:02:00     0+00:10:00     4     5 silver grade jobs
```

The standard listing includes the following fields:

MaxJobCPU The CPU limit for all the processes in a job of this class.

MaxProcCPU The CPU limit for processes in this class.

Free Slots The number of free slots (available classes) on this machine.

Max Slots The total number of slots (configured classes) on this machine.

Description The description of this class.

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 silver** command, where there are five `silver` classes configured in the cluster, with one `silver` class job currently running.

```
=============== Class silver ==========
                 Name: silver
             priority: 50
                admin: brownap
            NQS_class: F
           NQS_submit:
            NQS_query:
      max_processors: -1
              maxjobs: -1
       class_comment: silver grade jobs
    wall_clock_limit:   0+00:24:00, -1
       job_cpu_limit:   0+00:02:00, -1
           cpu_limit:   0+00:10:00, -1
          data_limit: -1, -1
          core_limit: -1, -1
          file_limit: -1, -1
         stack_limit: -1, -1
           rss_limit: -1, -1
                nice: 0
                free: 4
             maximum: 5
```

The long listing includes these fields:

Name The name of the class

Priority The system priority of this class relative to other classes.

admin The list of administrators of this class.

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

NQS_submit The NQS queue where the job will be submitted.

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

max_processors The maximum number of processors than can be used for parallel jobs.

maxjobs The maximum number of jobs the class can run at any time.

class_comment The text supplied by the administrator describing this class.

wall_clock_limit The hard and soft wall clock limits (the elapsed time for which the job can run).

job_cpu_limit The hard and soft CPU limits for all processes in a job of this class.

cpu_limit The hard and soft CPU limitsfor all processes in this class.

data_limit The hard and soft limits for the data area used for processes in this class.

core_limit The hard and soft core size limits.

file_limit The hard and soft file size limits.

stack_limit The hard and soft stack size limits.

rss_limit	The hard and soft rss size limits.
nice	The *nice* value of jobs in this class.
free	The number of classes available to new jobs.
maximum	The total number of configured classes on this machine.

Related Information

Each machine periodically updates the central manager with a snapshot of its environment. Since the information returned by **llclass** is a collection of these snapshots, all taken at varying times, the total picture may not be completely consistent.

llctl - Control LoadLeveler Daemons

Purpose

Controls LoadLeveler daemons on all members of the LoadLeveler cluster.

Syntax

llctl [**-H**] [**-v**] [**-q**] [**-g** | **-h** *host*] *keyword*

Flags

-H Provides a short usage message.

-v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

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

-g Indicates that the command should apply globally to all machines in the **LoadL_admin** file.

-h *host* Indicates that the command should apply to only this machine in the LoadLeveler cluster. If neither **-h** nor **-g** is specified, the default is the machine on which the **llctl** command is issued.

keyword Must be specified after all flags and can be the following:

recycle
Stops all LoadLeveler daemons and restarts them.

reconfig
Forces all daemons to reread the configuration files.

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

stop
Stops the LoadLeveler daemons on the specified machine.

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 inadvertantly 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.

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) jobs in the Starting or Running state in the schedd queue are allowed to continue running, and (3) jobs in the Idle state in the schedd queue are drained, meaning they will not get dispatched. When you issue **drain startd**, the following happens: (1) the startd machine accepts no more LoadLeveler jobs to be run, and (2) jobs 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 redispatch (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 cancelled if it does not complete within a five minute period.

suspend
Suspends all jobs on this machine. This is not supported for parallel jobs.

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

version
Displays version and release data at the screen.

capture *eventname*
Captures accounting data for all jobs running on the designated machines and associates the name *eventname* with the data. *eventname* must be a character string containing no blanks.

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:

- 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 parallel job is running on a machine that receives the **llctl recycle** command, or the **llctl stop** and **llctl start** commands, the running job is terminated. You can restart the job by resubmitting the job or by specifying the **restart=yes** option in the job command file.

  If a serial job is running on a machine that receives the **llctl recycle** command, or the **llctl stop** and **llctl start** commands, the running job is terminated. You can restart the job by resubmitting the job or by enabling checkpointing and specifying the **restart=yes** option in the job command file.

- If you find that the **llctl -g start** command or the **llctl -g *** command is taking a long time to complete, you should consider using the SP **dsh** command to send **llctl** commands to multiple nodes in a parallel fashion. For more information on **dsh**, see *IBM RS/6000 Scalable POWERparallel Systems: Administration Guide*, (SH26-2486).

Examples

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

```
llctl -h iron stop
```

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

```
llctl -g start
```

This example causes the LoadLeveler daemons on machine *iron* to re-read the 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 medium large
```

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 164.

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.

```
5/20 10:50:21 Sent command to mars
```

llfavorjob - Reorder System Queue by Job

Purpose

Sets specified jobs to a higher system priority than all jobs that are not favored.

This command also *unfavors* previously favored job(s), restoring the original priority, when you specify the **-u** flag.

Syntax

llfavorjob [**-H**] [**-v**] [**-u**] *joblist*

Flags

-H Provides a short usage message.

-v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

-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 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 jobs 12.4 on the machine *iron* and 8.2 on *zinc* the highest priorities in the system, with the jobs ordered by the **SYSPRIO** expression.

```
llfavorjob iron.12.4 zinc.8.2
```

This example unfavors jobs 12.4 on the machine *iron* and 8.2 on the machine *zinc*.

```
llfavorjob -u iron.12.4 zinc.8.2
```

llfavoruser - Reorder System Queue by User

Purpose

Sets a user's job(s) to the highest priority in the system, regardless of the current setting of the job priority, or SYSPRIO. Jobs already running are not affected. This command also *unfavors* the user's job(s), restoring the original priority, when you specify the **-u** flag.

Syntax

llfavoruser [**-H**] [**-v**] [**-u**] *userlist*

Flags

-H Provides a short usage message.

-v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

-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 negotiator is restarted, any favor applied to users is revoked.

The user's jobs still remain ordered by **UserPrio** (which may cause jobs for the user to swap **SYSPRIO**). If more than one user is affected by this command, the jobs of favored users are ordered by **SYSPRIO** and are scanned before the jobs of not favored users. However, jobs of favored users which do not match job requirements with available machines may run after jobs of not favored users.

Examples

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

```
llfavoruser ellen fred
```

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

```
llfavoruser -u ellen fred
```

Although this example includes the list of users, it does not have to.

llhold - Hold or Release a Submitted Job

Purpose

Places jobs in user hold or system hold and releases jobs from both types of hold. Users can only move their own jobs into and out of user hold. Only LoadLeveler administrators move jobs into and release them from system hold.

Syntax

llhold [**-H**] [**-v**] [**-q**] [**-s**] [**-r**] [**-u** *userlist*] [**-h** *hostlist*] [*joblist*]

Flags

-H Provides a short usage message.

-v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

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

-s Puts job(s) in system hold. Only a system 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 an 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 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 schedd machine that sent the job to the negotiator.
- *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 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

This command does not affect a job step that is running until 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 fron 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.

```
hold: Hold command has been sent to central manager on "bronze"
```

llinit - Initialize Machines in the LoadLeveler Cluster

Purpose

Initializes a new machine as a member of the LoadLeveler hardware resource cluster

Syntax

llinit [**-H**] [**-quiet**] [[**-prompt**] [**-local** *pathname*] [**-release** *pathname*] [**-cm** *machine*] [**-debug**]

Flags

-H Provides a short usage message.

-quiet Quiet mode. Information messages do not display. Error messages always display.

-prompt Prompts or leads you through a set of questions that help you to complete the **llinit** command.

-local *pathname* Where *pathname* is the local directory on which to create the spool, execute, and log sub-directories. 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* Where *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/nfs** directory.

-cm *machine* Where *machine* is the central manager machine, where the negotiator daemon runs.

-debug Displays a large amount of messages, tracing the path through **llinit** during execution. This is intended for debugging purposes only.

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 **LoadL_config** and **LoadL_admin** files are copied from LoadLeveler's release directory to LoadLeveler's home directory. **LoadL_config.local** is copied from LoadLeveler's release directory to LoadLeveler's local directory.

llinit initializes a new machine as a member of the LoadLeveler resource 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_admin** and the **LoadL_config** files from the release directory samples subdirectory into the loadl home directory.
- 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).
- Create symbolic links from the home directory to the bin, lib, man, samples, and include subdirectories in the release directory.
- For AIX platforms: creates symbolic links for the **llcc** and **llxlf** commands 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**/**nfs** as the release directory, and the machine named **bronze** as the central manager.

```
llinit -local /var/loadl -release /usr/lpp/LoadL/nfs -cm bronze
```

Results

The following is sample output from this command:

```
llinit -local /home/ll_admin -release /usr/lpp/LoadL/nfs -cm mars
```

```
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/nfs/bin"
llinit: creating symbolic link "/home/ll_admin/lib -> /usr/lpp/LoadL/nfs/lib"
llinit: creating symbolic link "/home/ll_admin/man -> /usr/lpp/LoadL/nfs/man"
llinit: creating symbolic link "/home/ll_admin/samples -> /usr/lpp/LoadL/nfs/samples"
llinit: creating symbolic link "/home/ll_admin/include -> /usr/lpp/LoadL/nfs/include"
llinit: ==> symbolic link "/usr/lpp/LoadL/nfs/bin/llcc -> /bin/cc" already exists
llinit: ==> symbolic link "/usr/lpp/LoadL/nfs/bin/llxlf -> /bin/xlf" already exists
llinit: program complete.
```

llprio - Change the User Priority of Submitted Job Steps

Purpose

Changes the user priority of one or more job steps in the LoadLeveler queue. You can adjust the priority by supplying a + (plus) or – (minus) immediately followed by an *integer* value. **llprio** does not affect a job step that is running, even if its priority is lower than other jobs steps, unless the job step goes into the Idle state.

Syntax

llprio [**-H**] [**-v**] [**-q**] [+*integer* | –*integer* | **-p** *priority*] *joblist*

Flags

-H Provides a short usage message.

-v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

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

+/– *integer* Operates on the current priority of the job step, making it higher (closer to execution) or lower (further from execution) by adding or subtracting the value of *integer*.

-p *priority* Is the new absolute value for priority. The valid range is 0–100 (inclusive) where 0 is the lowest possible priority and 100 is highest.

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

- *host* is the name of the 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 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 system 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 29 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.

```
prio: Prio command has been sent to central manager on "silver"
```

llq - Query Job Status

Purpose

Returns information about jobs that have been dispatched.

Syntax

llq [**-H**] [**-v**] [**-x**] [**-s**] [**-l**] [*joblist*] [**-u** *userlist*] [**-h** *hostlist*] [**-c** *classlist*]

Flags

-H Provides a short usage message.

-v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

-x Provides extended information about the selected job. If the **-x** flag is used with the **-h**, **-s**, or **-u** 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 macro in the LoadLeveler configuration file.

Normally, **llq** connects with the central manager to obtain job information. When you specify **-x**, **llq** connects to the schedd machine that received the specified job to get extended job information.

When specified without **-l**, CPU usage for active jobs is reported in the short format. Using **-x** can produce a very long report and can cause excess network traffic.

-s Provides information on why a selected list of jobs remain in the Hold, 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 job(s) cannot run. If a list of machines is not provided, the default is the list of machines in the LoadLeveler cluster. For each job, **llq** determines why the job remains in one of the given states instead of Running.

-l Specifies that a long listing be generated for each job for which status is requested. Detailed information can only be displayed for jobs belonging to the user issuing the **llq** command. If status is requested for a job which does not belong to the user, an error message is generated and no further output is displayed. Administrators can always display detailed information about any job. Fields included in the long listing are shown in “Results” on page 208. Only the owner of a job and the administrator can use this option.

If **-l** is *not* specified, then the standard listing is generated as shown in “Results” on page 208.

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

- *host* is the name of the 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 members of the cluster.

-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. If the **-s** flag is specified, this value is ignored. When used with **-h**, only those jobs monitored on the machines in the *hostlist* are queried. When used alone, only the classes configured on the schedd machine are queried.

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** command, where the machine `mars` has two jobs running and one job waiting.

```
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
mars.498.0               brownap     5/20 11:31 R  100 silver       mars
mars.499.0               brownap     5/20 11:31 R  50  No_Class     mars
mars.501.0               brownap     5/20 11:31 I  50  silver

3 jobs in queue 1 waiting, 0 pending, 2 running, 0 held.
```

The standard listing includes the following fields:

Id job identifier presented as *host.jobid.stepid* number. The job ID may be shortened if the job ID is a name which would be too long to fit into the standard format. The name is shortened by removing the domain name from the host portion of the job ID In this case, a dash (-) is added to the shortened name to indicate that it was shortened. 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
- **D** Deferred
- **H** User Hold
- **I** Idle
- **NR** Not Run
- **NQ** Not Queued
- **P** Pending
- **R** Running
- **RM** Removed
- **RP** Remove Pending
- **S** System Hold
- **SH** User Hold and System Hold
- **ST** Starting
- **V** Vacated
- **XP** Reject Pending

For a detailed explanation of these job states, see “LoadLeveler Job States” on page 18.

PRI user priority of the job, 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 205.

Class job class.

Running On if running, the machine the job is running on. This is blank when the job is not running. For parallel jobs, only the first machine is shown.

The Long Listing: The long listing is generated when you specify the **-l** option with the **llq** command. The following is sample output for the **llq -l mars.502.0** command:

```
=============== Job Step mars.502.0 ===============
        Job Step Id: mars.502.0
           Job Name: mars.502
          Step Name: 0
  Structure Version: 9
              Owner: brownap
         Queue Date: Mon May 20 11:36:40 1996
             Status: Running
      Dispatch Time: Mon May 20 11:36:43 1996
         Start Time:
    Completion Date:
    Completion Code:
      User Priority: 50
       user_sysprio: 0
      class_sysprio: 50
      group_sysprio: 0
    System Priority: 4495906
          q_sysprio: 4495906
      Notifications: Complete
        User Number: 1
 Virtual Image Size: 8 kilobytes
          Checkpoint:
             Restart: yes
     Hold Job Until:
                 Cmd: c1
                Args: 30
                 Env:
                  In: /dev/null
                 Out: c1x.502.0.out
                 Err: c1x.502.0.err
Initial Working Directory: /u/brownap/loadl_jobs
          Dependency:
       Requirements: (Arch == "R6000") && (OpSys == "AIX41")
        Preferences:
           Job Type: Serial
     Min Processors:
     Max Processors:
     Allocated host: mars
    Submitting host: mars
        Notify User: brownap@mars
              Shell: /bin/ksh
           LL_group: No_Group
              Class: silver
     Cpu Hard Limit: 600 seconds
     Cpu Soft Limit: -1
    Data Hard Limit: -1
    Data Soft Limit: -1
    Core Hard Limit: -1
    Core Soft Limit: -1
    File Hard Limit: -1
    File Soft Limit: -1
   Stack Hard Limit: -1
   Stack Soft Limit: -1
     Rss Hard Limit: -1
     Rss Soft Limit: -1
 Job Cpu Hard Limit: 120 seconds
 Job Cpu Soft Limit: -1
Wall Clock Hard Limit: 1440 seconds
Wall Clock Soft Limit: -1
            Comment:
            Account:
         Unix_group: usr
   NQS Submit Queue:
   NQS Query Queues:
Negotiator Messages:
```

The long listing includes these fields:

Job Step ID job step identifier.

Job Name name of the job.

Step Name name of the job step

Structure Version internal version identifier.

Owner userid of the job submitter.

Queue Date date and time job was received by LoadLeveler.

Status status (state) of the job. Job status can be:

Completed
Deferred
Idle
NotQueued
Not Run
Pending
Reject Pending
Removed
Remove Pending
Running
Starting
System Hold
System User Hold
User Hold
Vacate

For a detailed explanation of these job states, see "LoadLeveler Job States" on page 18.

Dispatch Time the time the job was dispatched.

Start Time the time the starter process was executed

Completion Date date and time job was completed.

Completion Code the job completion code.

User Priority priority of the job, as specified by the user in the job command, or changed by the **llprio** command.

user_sysprio user system priority of the job, where the value is defined in the administration file.

class_sysprio class priority of the job, where the value is defined in the administration files.

group_sysprio group priority of the job, where the value is defined in the administration files.

System Priority overall system priority of the job, where the value is defined by the SYSPRIO expression in the configuration file.

q_sysprio adjusted system priority of the job. (See "How Does a Job's Priority Affect Dispatching Order?" on page 29.)

Notifications notification status for the job, where:

always notification is sent through mail for all four job status categories below.

complete notification is sent through mail only when job completes.

error	notification is sent through mail only when job terminates abnormally.
never	notification is never sent.
start	notification is sent through mail only upon starting or restarting job.

Virtual Image Size	of the executable that was submitted.
Checkpoint	checkpoint status (yes or no)
Restart	restart status (yes or no)
Hold Job Until	job is deferred until this date and time.
Cmd	name of the executable that was submitted.
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	file to be used for stdin.
Out	file to be used for stdout.
Err	file to be used for stderr.
Init Working Directory	directory from which the job is run. The relative directory from which the stdio files are accessed, if appropriate.
Dependency	job requirements as specified when the job was submitted.
Requirements	job requirements as specified when the job was submitted.
Preferences	job preferences as specified when the job was submitted.
Job Type	type of job (serial or parallel).
Min Processors	minimum number of processors needed for this job.
Max Processors	maximum number of processors needed for this job.
Allocated Hosts	the machines that have been allocated for this job.
Submitting Host	name of machine to which job is submitted the job.
Notify User	user to be notified by mail of job status.
Shell	shell to be used when job is run.
LL_Group	the LoadLeveler group associated with the job.
Class	job class as specified when job was submitted.
CPU Hard Limit	CPU hard limit as specified when job was submitted.
CPU Soft Limit	CPU soft limit as specified when job was submitted.
Data Hard Limit	Data hard limit as specified when job was submitted.
Data Soft Limit	Data soft limit as specified when job was submitted.
Core Hard Limit	Core hard limit as specified when job was submitted.
Core Soft Limit	Core soft limit as specified when job was submitted.

File Hard Limit	File hard limits as specified when job was submitted.
File Soft Limit	File soft limit as specified when job was submitted.
Stack Hard Limit	Stack hard limit as specified when job was submitted.
Stack Soft Limit	Stack soft limit as specified when job was submitted.
Rss Hard Limit	RSS hard limit as specified when job was submitted.
Rss Soft Limit	RSS soft limit as specified when job was submitted.
Job Cpu Hard Limit	Job CPU hard limit as specified when job was submitted.
Job Cpu Soft Limit	Job CPU soft limit as specified when job was submitted.
Wall Clock Hard Limit	Wall clock hard limit as specified when job was submitted.
Wall Clock Soft Limit	Wall clock soft limit as specified when job was submitted.
NQS Submit Queue	The name of the NQS pipe queue to which the NQS job will be routed.
NQS Query Queue	The NQS queue names you can use to monitor the job.
Comment	The comment specified in the job command file.
Account	account number specified in the job command file.
UNIX Group	effective UNIX group name.
Negotiator Messages	informational message for jobs in the Idle or NotQueued state.

Other fields found when issuing **llq -x -l** are:

maxrss	maximum resident set sized utilized.
ixrss	amount of shared memory used.
idrss	amount of unshared memory used.
isrss	Integral unshared stack used.
minflt	# Page faults (re-claimed).
majflt	# Page faults (I/O required).
nswap	# times swapped out.
inblock	# times file system performed input.
oublock	# times file system performed output.
msgsnd	# of IPC messages sent.
msgrcv	# of IPC messages received.
nsignals	# of signals delivered.
nvcsw	# of context switches due to voluntarily giving up processor.
nivcsw	# of involuntary context switches.

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**] [**-v**] [**-l**] [*hostlist*]

Flags

-H Provides a short usage message.

-v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

-l Specifies that a long listing be generated for each machine for which status is requested. If **-l** is *not* specified, the standard list, described below, is generated.

hostlist Is a blank-delimited list of machines for which status is requested.

Description

If no *hostlist* is specified, all machines are queried.

If you have more than a few machines configured for LoadLeveler, consider redirecting the output to a file when using the **-l** flag.

Each machine periodically updates the central manager with a snapshot of its situation. Since the information returned by using **llstatus** is a collection of such snapshots, all taken at varying times, the total picture may not be completely consistent.

Examples

This example requests a long status listing for machines named silver and gold.

```
llstatus -l silver gold
```

Results

The Standard Listing: The standard listing is generated when you do *not* specify the **-l** option with the **llstatus** command. The following is sample output from the **llstatus** command, where there are two nodes in the cluster.

```
Name                      Schedd  InQ Act Startd Run LdAvg Idle Arch      OpSys
k10n09.ppd.pok.ibm.com    Avail     3   3 Run      1 2.72     0 R6000     AIX41
k10n12.ppd.pok.ibm.com    Avail     0   0 Idle     0 0.00   365 R6000     AIX41

R6000/AIX41            2 machines   3 jobs   1 running
                       2 machines   3 jobs   1 running

The Central Manager is defined on k10n09.ppd.pok.ibm.com

All machines on the machine_list are present
```

The standard listing includes the following fields:

Name hostname of the machine.

Schedd state of the schedd daemon, which can be one of the following:

Down
Drned (Drained)
Drning (Draining)
Avail (Available)

For a detailed explanation of these states, see "The schedd Daemon" on page 14.

InQ number of jobs in the queue that were scheduled from this machine.

Act number of jobs that the schedd has dispatched.

Startd state of the startd daemon, which can be:

Busy
Down
Drned (Drained)
Drning (Draining)
Flush
Idle
None
Resrvd (Reserved)
Run (Running)
Suspnd (Suspend)

For a detailed explanation of these states, see "The startd Daemon" on page 15.

Run number of jobs running on this machine.

LdAvg Berkeley one-minute load average on this machine.

Idle number of seconds since keyboard or mouse activity in a login session was detected. Highest number displayed is 9999.

Arch hardware architecture of machine as listed in configuration file.

OpSys operating system on this machine.

The Long Listing: The long listing is generated when you specify the **-l** option with the **llstatus** command. The following is sample output from the **llstatus -l k10n09** command:

```
name: "k10n09.ppd.pok.ibm.com"
machine_context:
Running             = 0
ScheddAvail         = 1
StartdAvail         = 1
State               = Idle
OpSys               = AIX41
Arch                = R6000
Machine             = k10n09.ppd.pok.ibm.com
START               = T
SUSPEND             = F
CONTINUE            = T
VACATE              = F
KILL                = F
SYSPRIO             = ((((ClassSysprio *  100) + (UserSysprio *  10)) +  (GroupSysprio *  1)) -  QDate)
MACHPRIO            = (0 -  LoadAvg)
VirtualMemory       = 46220
EnteredCurrentState = Mon May 20 11:36:49 1996
Disk                = 513252
Tmp                 = 10384
KeyboardIdle        = 6
LoadAvg             = 2.006492
AvailableClasses    = { "No_Class" "No_Class" "a" "a" "b" "c" "i" "silver" "silver" "silver" "silver" }
DrainingClasses     = { }
DrainedClasses      = { }
ConfiguredClasses   = { "No_Class" "No_Class" "a" "a" "b" "c" "i" "silver" "silver" "silver" "silver" }
Feature             = { }
Memory              = 256
Max_Starters        = 5
ConfigTimeStamp     = Mon May 20 11:36:35 1996
Cpus                = 1
Speed               = 1.000000
MasterMachPriority  = 0.000000
Subnet              = 129.40.161
CustomMetric        = 1
ScheddRunning       = 3
Pending             = 0
Starting            = 0
Idle                = 0
Unexpanded          = 0
Held                = -1
Removed             = 0
RemovePending       = 0
Completed           = 19
DependantNotRun     = 0
TotalJobs           = 3
time_stamp: Mon May 20 11:39:20 1996
```

The long listing includes these fields:

Name hostname of the machine.

Running number of jobs running on this machine.

ScheddAvail flag indicating if machine is running a schedd daemon (0=no, 1=yes).

StartdAvail flag indicating if machine is running a startd daemon (0=no, 1=yes).

State state of the startd daemon, which can be:

Busy
Down
Drain
Flush
Idle
None
Reserved
Running
Suspend

For a detailed explanation of these states, see "The startd Daemon" on page 15.

OpSys operating system on this machine.

Arch hardware architecture of machine as listed in configuration file.

Machine fully qualified name of 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.

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.

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.

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.

KILL 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 killed on this machine.

SYSPRIO actual expression that determines overall system priority of the job, defined in the configuration file.

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

Virtual Memory available swap space, in kilobytes, on this machine.

Entered Current State date and time when machine state was set.

Disk	available space, in kilobytes (less 512KB) in LoadLeveler's execute directory on this machine.
Keyboard Idle	number of seconds since last keyboard or mouse activity.
LoadAvg	Berkely one-minute load average on machine.
AvailableClasses	set of currently available classes.
DrainingClasses	set of names of classes which are currently being drained on this machine.
DrainedClasses	set of names of classes which have been drained on this machine and are therefore unavailable.
ConfiguredClasses	set of all classes supported on this machine, both those in use and those not in use, as defined in the configuration file.
Pool	the identifier of the pool where this startd machine is located.
Adapter	set of all adapters on this machine.
Feature	set of all features on this machine.
Memory	physical memory, in megabytes, on this machine.
Max_Starters	maximum number of jobs that can run simultaneously on this machine.
Config Time Stamp	date and time of last (re)configuration.
Cpus	number of CPUs on this machine.
Speed	speed associated with the machine.
MasterMachPriority	machine priority for the parallel master node.
Subnet	TCP/IP subnet that this machine resides on.
CustomMetric	number that indicates the order of the machines for scheduling purposes.
ScheddRunning	number of jobs submitted to this machine that are running somewhere in the LoadLeveler cluster.
Pending	number of jobs in this state on this schedd machine.
Starting	number of jobs in this state on this schedd machine.
Idle	number of jobs in this state on this schedd machine.
Held	number of jobs in this state on this schedd machine.
Removed	number of jobs in this state on this schedd machine.
Remove Pending	number of jobs in this state on this schedd machine.
Completed	number of jobs in this state on this schedd machine.
DependentNotRun	number of jobs in this state on this schedd machine.
Total Jobs	number of total jobs submitted to this schedd machine.
ScheddState	state of the schedd on this schedd machine.
time stamp	date and time the central manager last received a status update from this schedd machine.

llsubmit - 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

llsubmit [**-H**] [**-v**] [**-q**] [*cmdfile* | **–**]

Flags

-H Provides a short usage message.

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

-v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

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 **llsubmit** command.
- When a LoadLeveler job ends, you may receive UNIX mail notification indicating the job exit status. For example, you could get the following mail message:

  ```
  Your LoadLeveler job
       myjob1
  exited with status 139.
  ```

  The return code 139 is from the user's job, and is not a LoadLeveler return code.
- For information on writing a program to filter job scripts when they are submitted, see “Filtering a Job Script” on page 281.

Examples

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

```
llsubmit qtrlyrun.cmd
```

Results

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

```
submit: The job "earth.505" has been submitted.
```

Note that `505` is the job ID generated by LoadLeveler.

llsummary - Return Job Resource Information for Accounting

Purpose

Returns job resource information on completed jobs for accounting purposes.

Syntax

llsummary [**-H**] [**-v**] [**-x**] [**-l**] [**-s** *MM/DD/YY* **to** *MM/DD/YY*]
[**-e** *MM/DD/YY* **to** *MM/DD/YY*][**-u** *user*] [**-c** *class*] [**-g** *group*] [**-G** *unixgroup*]
[**-a** *allocated*] [**-r** *report*] [**-j** *host.jobid*] [**-d** *section*] [*filename*]

Flags

-H Provides a short usage message

-v Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

-x Provides extended information. Using **-x** can produce a very long report. This option is meaninful 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. For more information, see “Step 7: Define Job Accounting” on page 106.

-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/YY* (where *MM* is month, *DD* is day, and *YY* is year), or is a string of digits representing the number of seconds since 1970. The default is to report 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/YY* (where *MM* is month, *DD* is day, and *YY* is year), or is a string of digits representing the number of seconds since 1970. The default is to report 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.

-g *group* Specifies the LoadLeveler group for which accounting data reported is reported.

-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 can choose one or more of the following:

resource	Provides CPU usage.
avgthroughput	Provides average queue time, run time, and CPU time
maxthroughput	Provides maximum queue time, run time, and CPU time
minthroughput	Provides minimum queue time, run time, and CPU time
throughput	Selects all thorughput 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.

filename The file containing the accounting data. If not specified, the default is the local history file on the machine from which the command was issued. You can use the **llacctmrg** command to produce such a file.

Examples

The following example requests summary reports (standard listing) of all the jobs submitted on your machine between the days of September 12, 1994 and October 12, 1994:

```
llsummary -s 09/12/94 to 10/12/94
```

Results

The Standard Listing: The standard listing is generated when you do not specify **-l**. **-r**, or **-d** with **llsummary**. This sample report includes summaries of the following data:

- Number of jobs, Total CPU usage, per user.
- Number of jobs, Total CPU usage, per class.
- Number of jobs, Total CPU usage, per group.
- Number of jobs, Total CPU usage, per account number.

The following is an example of the standard listing:

```
       Name   Jobs   Steps        Job Cpu    Starter Cpu     Leverage
    krystal     15      36     0+00:09:50     0+00:00:10         59.0
     lixin3     18      54     0+00:08:28     0+00:00:16         31.8
      TOTAL     33      90     0+00:18:18     0+00:00:27         40.7

      Class   Jobs   Steps        Job Cpu    Starter Cpu     Leverage
      small      9      21     0+00:01:03     0+00:00:06         10.5
      large     12      36     0+00:13:45     0+00:00:11         75.0
       osl2      3       9     0+00:00:27     0+00:00:02         13.5
   No_Class      9      24     0+00:03:01     0+00:00:06         30.2
      TOTAL     33      90     0+00:18:18     0+00:00:27         40.7

      Group   Jobs   Steps        Job Cpu    Starter Cpu     Leverage
   No_Group     12      30     0+00:09:32     0+00:00:09         63.6
  chemistry      7      18     0+00:04:50     0+00:00:05         58.0
engineering     14      42     0+00:03:56     0+00:00:12         19.7
      TOTAL     33      90     0+00:18:18     0+00:00:27         40.7

    Account   Jobs   Steps        Job Cpu    Starter Cpu     Leverage
      33333     16      39     0+00:05:54     0+00:00:11         32.2
      22222     15      45     0+00:12:05     0+00:00:13         55.8
      99999      2       6     0+00:00:18     0+00:00:01         18.0
      TOTAL     33      90     0+00:18:18     0+00:00:27         40.7
```

The standard listing includes the following fields:

Name User ID submitting jobs.

Class Class specified or defaulted for the jobs.

Group User's login group.

Account Account number specified for the jobs.

Jobs Count of the total number of jobs submitted by this user, class, group, or account.

Steps Count of the total number of job steps submitted by this user, class, group, or account.

Job CPU Total CPU time consumed by user's jobs.

Starter CPU Total CPU time consumed by LoadLeveler starter processes on behalf of the user jobs.

Leverage Ratio of job CPU to starter CPU.

The Long Listing: When you specify the **-x** option in conjunction with the **-l** option on the **llsummary** command, the long report resembles the following:

llsummary

```
================== Job ll1.kgn.ibm.com 772 =================
             Job Id: ll1.kgn.ibm.com 772
           Job Name: ll1.kgn.ibm.com.772
  Structure Version: 8
              Owner: jeffli
         Unix_group: staff
    Submitting host: ll1.kgn.ibm.com
  Submitting Userid: 17212
 Submitting Groupid: 100
    Number of steps: 2
-------------------------- Step ll1.kgn.ibm.com 772.0 -----------------
                  Job Step Id: ll1.kgn.ibm.com 772.0
                    Step Name: 0
                   Queue Date: Wed Sep  6 09:43:38 1995
          Job Step Dependency: NONE
                       Status: Completed
               Completion Date: Wed Sep  6 10:27:23 1995
               Completion Code: 0
                   Start Count: 1
                 User Priority: 50
                  user_sysprio: 0
                 class_sysprio: 0
                 group_sysprio: 0
                   User Number: 1
                 Notifications: Complete
           Virtual Image Size: (not recorded)
                    Checkpoint: no
                       Restart: yes
               Hold Job Until: (no hold)
                          Cmd: job1.cmd
                         Args: NONE
                          Env: LOADL_CORESIZE = 1024
                           In: /dev/null
                          Out: job1.ll1.772.0.out
                          Err: job1.ll1.772.0.err
    Initial Working Directory: /u/jeffli/regress
                 Requirements: (Arch == "R6000") && (OpSys == "AIX32")
                  Preferences: NONE
                     Job Type: Serial
          Allocated host name: ll1.kgn.ibm.com
                  Notify User: jeffli@ll1.kgn.ibm.com
                        Shell: /bin/ksh
                     LL_group: No_Group
                        Class: No_Class
           Job Cpu Hard Limit: 0 seconds
           Job Cpu Soft Limit: 0 seconds
                      Comment: NONE
                      Account: NONE
             NQS Submit Queue: NONE
             NQS Query Queues: NONE
            Job tracking exit: NONE
             Job tracking arg: NONE
------------------------------------ Detail for ll1.kgn.ibm.com.772.0 ------
                           Running host: ll1.kgn.ibm.com
                          Machine speed: 1.000000
                                  Event: System
                             Event name: completed
                          Time of Event: Wed Sep  6 10:27:23 1995
                        Starter User Time:   0+00:00:00.240000
                      Starter System Time:   0+00:00:00.390000
                       Starter Total Time:   0+00:00:00.630000
                           Starter maxrss: 828
                           Starter ixrss: 8388
                           Starter idrss: 6896
```

```
                        Starter isrss: 0
                       Starter minflt: 202
                       Starter majflt: 0
                        Starter nswap: 0
                      Starter inblock: 0
                      Starter oublock: 0
                       Starter msgsnd: 12
                       Starter msgrcv: 11
                     Starter nsignals: 1
                        Starter nvcsw: 79
                       Starter nivcsw: 0
                        Job User Time:   0+00:00:00.810000
                      Job System Time:   0+00:00:01.500000
                       Job Total Time:   0+00:00:02.310000
                           Job maxrss: 712
                            Job ixrss: 15540
                            Job idrss: 14296
                            Job isrss: 0
                           Job minflt: 1443
                           Job majflt: 0
                            Job nswap: 0
                          Job inblock: 0
                          Job oublock: 0
                           Job msgsnd: 5
                           Job msgrcv: 4
                         Job nsignals: 14
                            Job nvcsw: 70
                           Job nivcsw: 0
-------------------------- Step ll1.kgn.ibm.com 772.1 -----------------
                  Job Step Id: ll1.kgn.ibm.com 772.1
                     Step Name: 1
                    Queue Date: Wed Sep  6 09:43:38 1995
          Job Step Dependency: NONE
                       Status: Completed
              Completion Date: Wed Sep  6 10:27:26 1995
              Completion Code: 0
                  Start Count: 1
                User Priority: 50
                 user_sysprio: 0
                class_sysprio: 0
                group_sysprio: 0
                  User Number: 1
                Notifications: Complete
           Virtual Image Size: (not recorded)
                   Checkpoint: no
                      Restart: yes
               Hold Job Until: (no hold)
                          Cmd: job1.cmd
                         Args: NONE
                          Env: LOADL_CORESIZE = 1024
                           In: /dev/null
                          Out: job1.ll1.772.1.out
                          Err: job1.ll1.772.1.err
    Initial Working Directory: /u/jeffli/regress
                 Requirements: (Arch == "R6000") && (OpSys == "AIX32")
                  Preferences: NONE
                     Job Type: Serial
          Allocated host name: ll1.kgn.ibm.com
                  Notify User: jeffli@ll1.kgn.ibm.com
                        Shell: /bin/ksh
                     LL_group: No_Group
                        Class: small
           Job Cpu Hard Limit: 0 seconds
           Job Cpu Soft Limit: 0 seconds
```

llsummary

```
                      Comment: NONE
                      Account: NONE
             NQS Submit Queue: NONE
             NQS Query Queues: NONE
            Job tracking exit: NONE
             Job tracking arg: NONE
------------------------------------ Detail for ll1.kgn.ibm.com.772.1 ------
                         Running host: ll1.kgn.ibm.com
                        Machine speed: 1.000000
                                Event: System
                           Event name: completed
                         Time of Event: Wed Sep  6 10:27:26 1995
                    Starter User Time:   0+00:00:00.240000
                  Starter System Time:   0+00:00:00.400000
                   Starter Total Time:   0+00:00:00.640000
                       Starter maxrss: 828
                        Starter ixrss: 8424
                        Starter idrss: 7312
                        Starter isrss: 0
                       Starter minflt: 203
                       Starter majflt: 3
                        Starter nswap: 0
                      Starter inblock: 0
                      Starter oublock: 0
                       Starter msgsnd: 12
                       Starter msgrcv: 11
                     Starter nsignals: 1
                        Starter nvcsw: 60
                       Starter nivcsw: 0
                         Job User Time:   0+00:00:00.740000
                       Job System Time:   0+00:00:01.460000
                        Job Total Time:   0+00:00:02.200000
                            Job maxrss: 696
                             Job ixrss: 14060
                             Job idrss: 13260
                             Job isrss: 0
                            Job minflt: 1450
                            Job majflt: 1
                             Job nswap: 0
                           Job inblock: 0
                           Job oublock: 0
                            Job msgsnd: 5
                            Job msgrcv: 4
                          Job nsignals: 13
                             Job nvcsw: 64
                            Job nivcsw: 0
```

The Graphical User Interface

© Copyright IBM Corp. 1993, 1996

Chapter 11. Graphical User Interface Overview

This chapter provides some introductory information on the LoadLeveler graphical user interface (GUI). This section provides neither complete nor detailed instructions on using either the LoadLeveler GUI or any other graphical user interface. 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.

This chapter also discusses how to customize your graphical user interface by modifying the **Xloadl** and **Xloadl_so** files and provides a discussion of the **skel.cmd** file.

Note that LoadLeveler provides an installation with two types of graphical user interfaces. One interface is for LoadLeveler users whose machines are interacting fully with LoadLeveler. The second interface is available to users whose machines are only participating on a limited basis. This second type of machine is called a submit-only machine.

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 246 for more information.

The LoadLeveler Main Window

LoadLeveler's main window has three sub-windows, titled Jobs, Machines, and Messages, as shown in Figure 33 on page 230. Each of these sub-windows has its own menu bar.

© Copyright IBM Corp. 1993, 1996

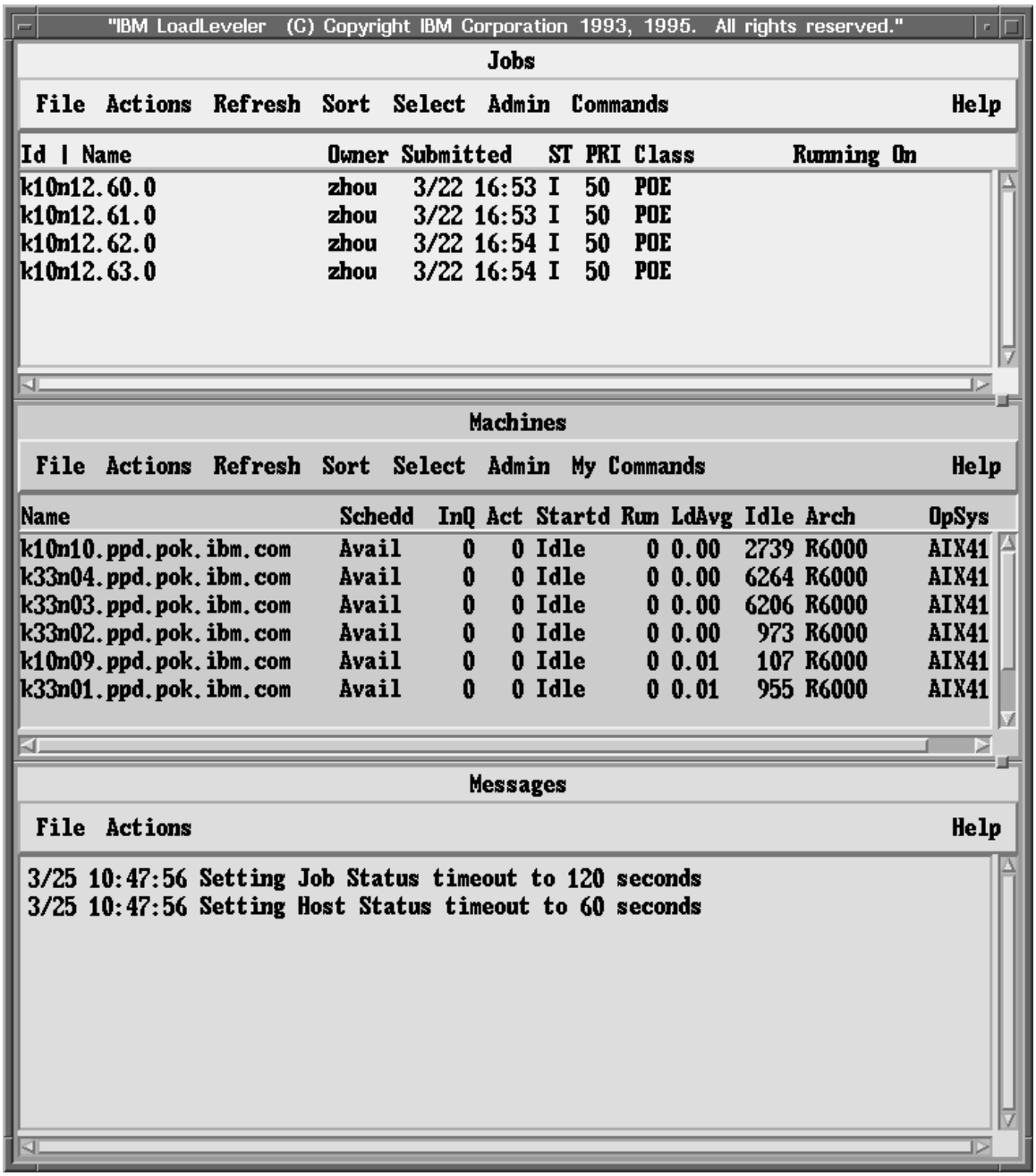


Figure 33. Main Window of the LoadLeveler GUI

The menu bar on the Jobs window relates to actions you can perform on jobs. The menu bar on the Machines window relates to actions you can perform on machines. Similarly, the menu bar on the Messages window displays actions you can perform related to LoadLeveler generated messages.

When you select an item from a menu bar, a pull-down menu appears. You can select an item from the pull-down menu to carry out an action or to bring up another pull-down menu originating from the first one.

Getting Help Using the Graphical User Interface

You can get help when using the GUI by pressing the Help key. This key is function key 1 (F1) on most keyboards. To receive help on parts of LoadLeveler, place the cursor over the area or field on which you want help and press Help. A help screen appears describing that area. You can also get help by using the Help pulldown menu and the Help push buttons available in pop-up windows.

Before you invoke the GUI, make sure your PATH statement includes the directory containing the LoadLeveler executable. Otherwise, some GUI functions may not work correctly.

Differences Between LoadLeveler's Graphical User Interface and Other Graphical User Interfaces

LoadLeveler's GUI contains many items common to other GUIs. There are, however, some differences that you should be aware of. These differences are:

- Accelerators or mnemonics do not appear on the menu bars.
- Submerged windows do not necessarily rise to the top when refreshed.

Building and Submitting Jobs Using the Graphical User Interface

This chapter explains how to build and submit a job to LoadLeveler using the GUI. In addition, you will learn how to perform other job related tasks. You can accomplish these same tasks by using the LoadLeveler commands. For information on these commands, refer to “Command Reference” on page 183.

This manual presents step-by-step instructions for performing tasks. For each step in a task, a user action and a system response to the action are included. User actions appear in uppercase boldface type, for example **PRESS**. The system response to an action follows a ▲. For example:

▲ The main window appears.

An action is sometimes represented by itself. For example:

PRESS OK

Other actions can require a selection or decision. Selection and decision actions are presented in tables.

Selection tables list all possible selections in the left column of the table. The following is an example of a selection table:

To	Do This
Submit a job	Refer to “Step 3: Submit a Job Command File” on page 239.
Cancel a job	Refer to “Step 9: Cancel a Job” on page 242.

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

Did the job you submitted complete processing?	
Yes	Submit another job.
No	Check the status of the job.

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

Task Scenario Using the Graphical User Interface

The tasks described in this chapter are those that you, as a user might be interested in accomplishing and are presented in a typical step-by-step scenario. You do not have to follow the steps shown here and may perform certain tasks before others without any difficulty. 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 the editor.

Step 1: Build a Job

From the Jobs window:

SELECT File → Build a Job

▲ The dialog box shown in Figure 34 appears:

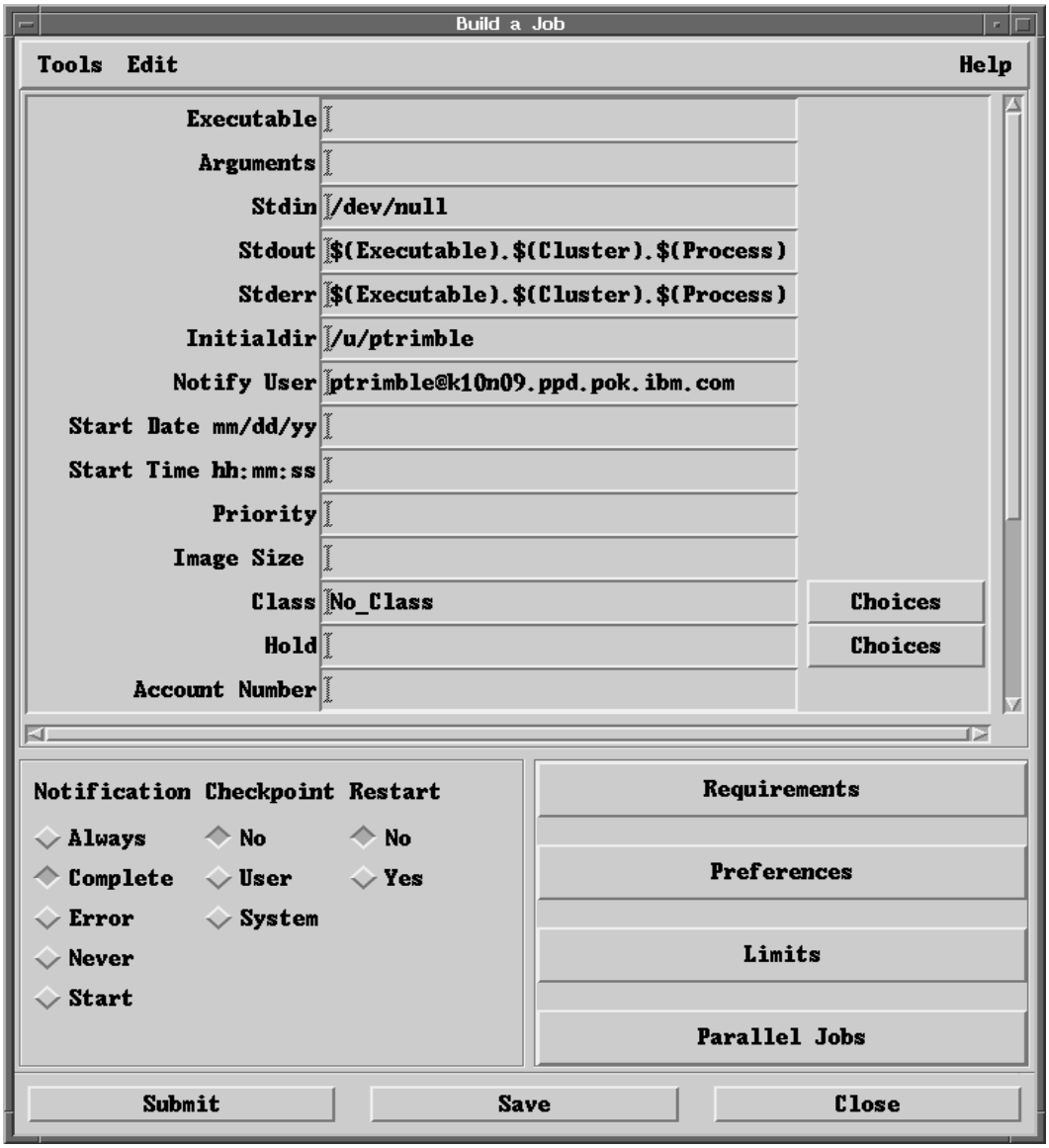


Figure 34. LoadLeveler Build a Job Window

Complete those fields for which you want to override what is currently specified in your **skel.cmd** defaults file. A sample **skel.cmd** file is found in /**usr**/**LoadL**/**nfs**/**samples**. You can update this file to define defaults for your site, and then update the ***skelfile** resource in **Xloadl** to point to your new **skel.cmd** file. If you want a personal defaults file, copy **skel.cmd** to one of your directories, edit the file, and update the ***skelfile** resource in **.Xdefaults**.

Field	Input
Executable	Name of the program to run. It must be an executable file. Optional. If omitted, the command file is executed as if it were a shell script.
Arguments	Parameters to pass to the program. Required only if the executable requires them.
Stdin	Filename to use as standard input (stdin) by the program. Optional. The default is /**dev**/**null**.
Stdout	Filename to use as standard output (stdout) by the program. Optional. The default is /**dev**/**null**.
Stderr	Filename to use as standard error (stderr) by the program. Optional. The default is /**dev**/**null**.
Initialdir	Initial directory. LoadLeveler changes to this directory before running the job. Optional. The default is your current working directory.
Notify User	User id of person to notify regarding status of submitted job. Optional. The default is your userid.
StartDate*	Month, day, and year in the format mm/dd/yy. The job will not start before this date. Optional. The default is to run the job as soon as possible.
StartTime*	Hour, minute, second in the format hh:mm:ss. The job will not start before this time. Optional. The default is to run the job as soon as possible.
Priority	Number between 0 and 100, inclusive. Optional. The default is 50. This is the user priority. For more information on this priority, refer to “Setting and Changing the Priority of a Job” on page 29.
Image size	Number in kilobytes that reflects the maximum size you expect your program to grow to as it runs. Optional.
Class	Class type. The job will only run on machines that support the specified class type. Your system administrator defines the class types. Optional. You can press the Choices button to get a list of available classes. Press the Details button under the class list to verify your permissions.
Hold	Hold status of the submitted job. Permitted values are: **user** user hold **system** system hold** **usersys** user and system hold** Optional. The default is a no-hold state. You can press the **choice** button to get a list of available hold types.
Account Number	Number identifying the job. For use with the llacctmrg and llsummary 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 user belongs. Optional.
Step Name	The name of this job step. Optional.

Field	Input
Dependency	A Boolean expression defining the relationship between the job steps. Optional.
Comments	Comments associated with the job. These comments help to distinguish one job from another job. Optional.

Note:

* If you specify *StartTime* but not *StartDate*, the default *StartDate* is the current day. If you specify *StartDate* but not *StartTime*, the default *StartTime* is 00:00:00. This means that the job will start as soon as possible on the specified date.

** Only the system administrator can specify this type of hold.

Note:

The fields that appear in this table are what you see when viewing the Build a Job window. The text in these fields do not necessarily correspond with the keywords listed in "Job Command File Keywords" on page 40.

See "Job Command File Keywords" on page 40 for information on the defaults associated with these keywords.

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 Checkpoint option

Your choices are:

No	Do not checkpoint the job. This is the default.
User	Yes, checkpoint the job. This type of checkpointing requires that you call the LoadLeveler **ckpt()** function from your program.
System	Yes, checkpoint the job. LoadLeveler automatically checkpoints the program.

Not checkpointing a job is the default.

Refer to "Checkpointing a Job Step" on page 29 for information on checkpointing a job.

SELECT a Restart option

Your choices are:

No	Do not restart the job.
Yes	Yes, restart the job.

PRESS Requirements

▲ The Requirements dialog box appears.

Complete those fields for which you want to specify requirements. Defaults are used for those fields that you leave blank. LoadLeveler

dispatches your job only to one of those machines with resources that matches the requirements you specify.

Field	Input
Architecture	Machine type. The job will not run on any other machine type. Optional. The default is the architecture of your current machine.
Operating System	Operating system. The job will not run on any other operating system. Optional. The default is the operating system of your current machine.
Disk	Amount of disk space in the execute directory. The job will only run on a machine with at least this much disk space. Optional. The default is defined in your local configuration file.
Memory	Amount of memory. The job will only run on a machine with at least this much memory. Optional. The default is defined in your local configuration file.
Machine(s)	Machine name(s). The job will only run on the specified machines. Optional
Feature(s)	Features. The job will only run on machines with specified features. Optional
Adapter	Adapter. Specifies the adapter on the SP system to use when processing a parallel job step. If you specify a value for this keyword, your job must run on the SP system. The default is Ethernet.
Pool	Pool. Specifies the number associated with the SP system parallel pool to use. If you specify a value for this keyword, your job must run on the SP system. The default is to select nodes from any pool.
Requirement	Requirements. The job will only run if these requirements are met.
Hint: If you type in 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, ensure 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.	
Note: 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.	

PRESS Close to return to the Build a Job dialog box.

PRESS Preferences

▲ The Preferences dialog box appears.

This dialog box, except for the last field, is the same as the Requirements dialog box. Complete the fields for those parameters that you want to specify. These parameters are not binding. For any preferences that you specify, LoadLeveler attempts to find a machine that matches these preferences along with your requirements. If it cannot find the machine, LoadLeveler chooses the first machine that matches the requirements.

PRESS Close to return to the Build a Job dialog box.

PRESS Limits

▲ The Limits dialog box appears.

Complete the fields for those limits that you want to impose upon your job. If you type *copy* in any field, the limits in effect on the submit machine are used. If you leave any field blank, the default limits in effect for your userid on the machine that runs the job are used.

Field	Input
CPU Limit	Maximum amount of CPU time that the submitted job can use. Express the amount as: `[hours:[minutes:][seconds][.fraction]` For example, 12:56:21 is 12 hours, 56 minutes, and 21 seconds. Optional
Data Limit	Maximum amount of the data segment that the submitted job can use. Express the amount as: `integer[.fraction][units]` where integer and fraction represent strings of up to eight digits. Optional
Core Limit	Maximum size of a core file. Optional
RSS Limit	Maximum size of the resident set size. It is the largest amount of physical memory a user's process can allocate. Optional
File Limit	Maximum size of a file that is created. Optional
Stack Limit	Maximum size of the stack. Optional
Job CPU Limit	Maximum amount of CPU a single job step can use per processor. Optional
Wall Clock Limit	Maximum amount of elapsed time for which a job can run. Optional

PRESS Close to return to the Build a Job dialog box.

PRESS Parallel Jobs

▲ The Parallel 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 parallel job. Optional. The default is one.
Max # of Processors	Maximum number of processors required for running the parallel job. Optional. The default is one.
Job Type	The type of job that you want to process. Press the **Choices** button to get the following list of job types from which to choose: • Serial • PVM3 • Parallel Optional. The default is Serial.
Parallel Path	The directory that defines where the PVM3 executables are located.

PRESS Close to return to the Build a Job dialog box.

Step 2: Edit the Job Command File

There are several ways that you can edit the job command file that you just built:

1. Using the Jobs window:

   SELECT **File → Submit a Job**

   ▲ The Submit a Job dialog box appears.

   SELECT the job file you want to edit from the file column.

   PRESS **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** pulldown menus on the Build a Job window:

   Using the Edit pulldown menu, you can modify the job command file. Your choices appear in the following table:

To	Select
Add a step to the job command file	Add a Step
Delete a step from the job command file	Delete a Step
Clear the fields in the Build a Job window	Clear Fields
Select defaults to use in the fields	Set Field Defaults
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** pulldown menu, you can modify the job command file. Your choices appear in the following table:

To	Select
Name the job	Set Job Name
Open a window where you can enter a script file	Append Script
Fill in the fields using another file	Restore from File
View the job command file in a window	View Entire Job
Determine which step you are viewing	What is step #
Start a new job command file	Start a new job

To	Do This
Save the information you entered into a file which you can submit later	**PRESS** **Save** ▲ A window appears prompting you to enter a job filename. **TYPE IN** a job filename in the text entry field. **PRESS** **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	**PRESS** **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.

PRESS **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.

PRESS 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.

For more information on these states, see "llq - Query Job Status" on page 207.

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:

Action	Select Sort →	Type of Sort
Sort jobs by the machine from which they were submitted	**Sort by Submitting Machine →**	[Primary|Secondary]
Sort by owner	**Sort by Owner →**	[Primary|Secondary]
Sort by the time the jobs were submitted	**Sort by Submission Time →**	[Primary|Secondary]
Sort by the state of the job	**Sort by State →**	[Primary|Secondary]
Sort jobs by their user priority (last job listed runs first)	**Sort by Priority →**	[Primary|Secondary]
Sort by the class of the job	**Sort by Class →**	[Primary|Secondary]
Sort by the group associated with the job	**Sort by Group →**	[Primary|Secondary]
Sort by the machine running the job	**Sort by Running Machine →**	[Primary|Secondary]
Sort by dispatch order	**Sort by Dispatch Order →**	[Primary|Secondary]
Not specify a sort	**No Sort**	[Primary|Secondary]

Each sorting option contains a cascading window which allows you to select this option as either a Primary or Secondary sorting option. For example, suppose you select Sort by Owner as the primary sorting option and Sort by Class as the secondary sorting option. The Jobs window is sorted by owner and, within each owner, by class.

Go to the next step.

Step 6: Change Priorities of Jobs in a Queue

If your job has not yet begun to run and is still in the queue, you can change the priority of the job in relation to your other jobs in the queue that belong to the same class. This only affects the user priority of the job. For more information on this priority, refer to “Setting and Changing the Priority of a Job” on page 29. 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.

PRESS **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**

▲ A warning dialog box appears prompting you to confirm your cancellation request. Once you confirm your request, LoadLeveler cancels the job and the job information disappears from the Jobs window.

Go to the next step.

Step 10: 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.

PRESS **OK**

▲ The window closes and the value you specified takes effect.

Go to the next step.

Step 11: 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:

Action	Select Sort →	Sort Type
Sort by machine name	**Sort by Name →**	[Primary|Secondary]
Sort by schedd state	**Sort by Schedd →**	[Primary|Secondary]
Sort by total number of jobs scheduled	**Sort by InQ →**	[Primary|Secondary]
Sort by number of running jobs scheduled by this machine	**Sort by Act →**	[Primary|Secondary]
Sort by startd state	**Sort by Startd →**	[Primary|Secondary]
Sort by the number of jobs running on this machine	**Sort by Run →**	[Primary|Secondary]
Sort by load average	**Sort by LdAvg →**	[Primary|Secondary]
Sort by keyboard idle time	**Sort by Idle →**	[Primary|Secondary]
Sort by hardware architecture	**Sort by Arch →**	[Primary|Secondary]
Sort by operating system type	**Sort by OpSys →**	[Primary|Secondary]
Not specify a sort	**No Sort**	[Primary|Secondary]

Each sorting option contains a cascading window which allows you to select this option 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 by hardware architecture, and within each architecture type, by machine name.

Go to the next step.

Step 12: 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 13: 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 14: Specify Which Jobs Appear in the Jobs Window

Normally, only your jobs appear in the Jobs window. You can, however, specify which jobs you want to appear by using the Select pull-down menu on the Jobs window.

To Display	Select Select →
All jobs in the queue	**All**
All jobs belonging to a specific user (or users)	**By User** ▲ A window appears prompting you to enter the user IDs whose jobs you want to view.
All jobs submitted to a specific machine (or machines)	**By Machine** ▲ A window appears prompting you to enter the machine names on which the jobs you want to view are running.
All jobs belonging to a specific group (or groups)	**By Group** ▲ A window appears prompting you to enter the LoadLeveler group names to which the jobs you want to view belong.
All jobs having a particular ID	**By Job Id** A dialog box prompts you to enter the id of the job you want to appear. This ID appears in the left column of the Jobs window. Type in the ID and press OK.
Note: When you choose By User, By Machines, or By Group, you can use a UNIX regular expression enclosed in parentheses. For example, you can enter **(^k10)** to display all machines beginning with the characters "k10".	

Select **Select → Show Selection** to show the selection parameters.

Go to the next step.

Step 15: 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 pulldown menu appears prompting you to select the state of the machines that you want to view.

Select **Select → Show Selection** to show the selection parameters.

Go to the next step.

Step 16: 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.

PRESS **OK**

▲ The window closes.

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

Both of these files are located in the **/usr/lib/X11/app-defaults** directory. 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. To see all the resources, or to make changes to the resources, edit the **Xloadl** file.

Note:

Another way to customize LoadLeveler is to place the resource names you want to modify, with their new values, in your **.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 10. Window Indentifiers in the Xloadl File	
Window	**Identifier**
Jobs	job_status
Machines	machine_status
Messages	message_area
Build a Job	builder
Submit a Job	submit
Requirements	requirements
Preferences	preferences
Limits	limits
Account Report Data	reporter
Parallel	parallel
Script	script

The following table lists the resource variables for all the windows and the buttons along with a description of each resource variable. Use the information in this table to modify your graphical user interface by changing the values of desired resource variables. The values of these resource variables depend upon Motif requirements.

Resource Variable	Description
geometry	The location of the object
foreground	The foreground color of the object
background	The background color of the object
width	The width of the object
height	The height of the object
labelString	The text associated with the object

Creating Your Own Pulldown Menus

You can add a pulldown menu to both the Jobs window and the Machines window.

To add a pulldown menu to the Jobs window, in the **Xloadl** file:

1. Set **userJobPulldown** to **True**
2. Set **userJob.labelString** to the name of your menu.
3. Fill in the appropriate information for your first menu item, **userJob_Option1**
4. To define more menu items, fill in the appropriate information for **userJob_Option2**, **userJob_Option3**, and so on. You can define up to ten menu items.

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

To add a pulldown menu to the Machines window, in the **Xloadl** file:

1. Set **userMachinePulldown** to **True**
2. Set **userMachine.labelString** to the name of your menu.

3. Fill in the appropriate information for your first menu item, **userMachine_Option1**
4. To define more menu items, fill in the appropriate information for **userMachine_Option2**, **userMachine_Option3**, and so on. You can define up to ten menu items.

Example – Creating a New Pulldown: 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.

```
*userJobPulldown: True
*userJob.labelString: Commands
*userJob.Option1: True
*userJob.Option1_command: ping -c1
*userJob_Option1.labelString: ping
*userJob_Option1_parameter: True
*userJob_Option1_output: Window
```

Figure 35. Creating a New Pulldown Menu

The **Xloadl** definitions shown in the Figure 35 create a menu bar item called "Commands". The first item in the Commands pulldown menu is called "ping". When you select this item, the command **ping -c1** is executed, with the machine you selected on the Machines window passed to this command. Your output is displayed in an informational window.

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

Customizing Fields on the Jobs Window and the Machines Window

You can control which fields are displayed and which fields are not displayed on the Jobs window and the Machine window by changing the **Xloadl** file. Look in the **Xloadl** file for "Resources for specifying lengths of fields displayed in the Jobs and Machines windows".

In most cases, you can remove a field from a window by setting its associated resource value to 0. To remove the Arch field from the Machines window, enter the following:

```
*mach_arch_len : 0
```

Note that the Job ID, Running On, and Machine Name fields must always be displayed and therefore cannot be set to 0.

All fields have a minimum length value. If you specify a smaller value, the minimum is used.

Modifying Help Panels

Help panels have the same characteristics as all of the windows plus a few unique ones:

Resource Variable	Values	Description
help*work_area.width	Any integer*	The width of the help panel.
help*work_area.height	Any integer*	The height of the help panel.
help*scrollHorizontal	[true|false] The default is False.	Sets the scrolling option on or off.
help*wordWrap	[true|false] The default is True.	Sets word wrapping on or off.
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 97.

You can access the administrative commands that act on LoadLeveler through the **Admin** pulldown menu on both the Jobs and the Machines pane of the GUI. The **Admin** pulldown menu on the Jobs window corresponds to the command options available in the **llhold**, **llfavoruser**, and **llfavorjob** commands. The **Admin** pulldown menu on the Machines window corresponds to the command options available in the **llctl** command.

The main window of the GUI, as shown in Figure 33 on page 230, 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** pulldown menu in the jobs pane of the GUI.

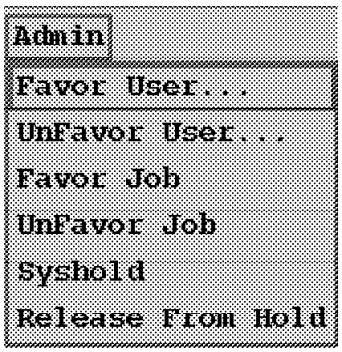


Figure 36. Administrator Commands: Job Pulldown

You can perform the following tasks with this menu:

Favor Users Allows you to favor users. This means that you can select one or more users whose jobs you want to move up in the job queue. This corresponds to the **llfavoruser** command.

Select **Admin** from the Jobs window

Select **Favor User**

▲The **Order by User** window appears.

Type in the name of the user for whom you want to favor their jobs.

Press OK

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 Jobs**

▲The selected jobs are favored.

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 Jobs**

▲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** pulldown 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** pulldown menu from the Jobs window

Select **Release From Hold** to release the system hold on the job.

Machine Related Administrative Actions

You access the administrative commands that act on machines through the **Admin** pulldown menu in the machines pane of the GUI.

Using the GUI pulldown menu, you can perform the tasks described in this section.

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

Select **Admin** from the Machines window.

Select **Start All**

Start LoadLeveler Allows you to start LoadLeveler on selected machines.

Select one or more machines on which you want to start LoadLeveler.

Select **Admin** from the Machines window.

Select **Start LoadLeveler**

Stop LoadLeveler Allows you to stop LoadLeveler on selected machines.

Select on or more machines on which you want to stop LoadLeveler.

Select **Admin** from the Machines window.

Select **Stop LoadLeveler**.

Stop All Stops LoadLeveler on all machines listed in machine stanzas. Use this option when specifying alternate central managers.

Select **Admin** from the Machines window.

Select **Stop All**

reconfig forces all daemons to reread the configuration files.

Select the machine on which you want to operate.

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**.

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 started.

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.

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**.

Application Programming Interfaces

© Copyright IBM Corp. 1993, 1996

Chapter 12. LoadLeveler Application Programming Interfaces

LoadLeveler provides several Application Programming Interfaces (API) that you can use. LoadLeveler's APIs are interfaces that allow application programs written by customers to use specific data or functions that are a part of LoadLeveler. This appendix describes these APIs.

Warning: These APIs are not *thread safe*; that is, they should not be linked to a threaded application.

Accounting APIs

LoadLeveler's job accounting function has two APIs. One is for account validation and the other is for extracting accounting data.

Account Validation API

LoadLeveler provides a program called **llacctval** which performs account validation. You must specify **A_VALIDATE** on the **ACCT** keyword in the configuration file in order to use **llacctval**. Your installation may choose to replace this routine with your own accounting routine by specifying the name of the program in the **ACCT_VALIDATION** keyword in the configuration file. The program is invoked from the **llsubmit** command.

Syntax

```
program user_name user_group user_acct# acct1 acct2 ...
```

Where:

program
: Is the name of the program to perform account validation which is the value specified in the configuration file in the **ACCT_VALIDATION** keyword.

user_name
: Is the name of the user to validate.

user_group
: Is the login group name of the user to validate.

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.

If the validation succeeds, the exit status must be zero. If it does not succeed, the exit status must be a non-zero number.

© Copyright IBM Corp. 1993, 1996

Report Generation API

Programmers may write C programs to process records in either a local or the global history file. These C programs can call the function **GetHistory** which will obtain an accounting record from the history file and invoke a caller supplied routine, passing the routine the address of an **LL_job** structure. This structure defines a history file record. The **GetHistory** function reads every record of the file, specified by the argument called **file**. Each record is read into an **LL_job** structure and passed to the function passed as the second argument.

An include file called **llapi.h** and a LoadLeveler library called **libllapi.a** are used to compile programs which call **GetHistory** and link them with **GetHistory**. **libllapi.a** can be found in the **lib** subdirectory and **llapi.h** can be found in the **include** subdirectory of the release directory.

You may need to reference the following fields found in the **ll_event_usage** structure of **libllapi.h**:

event
Is the event identifier. This is an interger whose value is one of the following:

1 For a LoadLeveler-generated event
2 For an installation-generated event

name
Is a character string identifying the event. You can record any event name using the command **llctl capture** *eventname*.

LoadLeveler-generated events can be the following:

started
checkpoint
vacated
completed
rejected
removed

Syntax for GetHistory

```
#include <stdio.h>
#include <sys/resource.h>
#include <sys/types.h>
#include <sys/time.h>
#include "llapi.h"

void
GetHistory(file, function, version)
char *file;
int function();
int version;
```

Where:

file Is the name of the history file to process.

function
Points to the function to call to process each history record. The syntax to specify this function is:

```
#include <stdio.h>
#include "llapi.h"

void
function(job)
LL_job *job;
```

Where:

job
Is a pointer to the **LL_job** structure representing a record in the history file.

version
Is an integer indicating the version of the history file being used. This argument should be set to **LL_JOB_VERSION** which is defined in the **llapi.h** include file.

Sample Accounting API

A sample program called **llpjob** is provided in the **samples**/**llphist** subdirectory of the release directory. **llpjob** invokes **GetHistory** to print every record in the history file. In order to compile **llpjob**, you need to update the sample **Makefile** to reflect your release directory if not /**usr**/**lpp**/**LoadL**/**nfs**. The syntax for running **llpjob** is:

```
llpjob file
```

Where *file* is a local or global history file.

Prolog and Epilog APIs

An administrator can write *prolog* and *epilog* programs 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 authentification (if AFS is installed). You must configure 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 enviroment prolog and epilog settings on the executing machine are ignored.

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

```
env id = value
```

Where:

id Is the name of the environment variable

value Is the value (setting) of the environment variable

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

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

Prolog Programs

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

Syntax

```
prolog_program
```

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

No arguments are passed to the program but the following environment variables are set:

LOADL_ACTIVE	LoadLeveler version
LOADL_STEP_ID	Job step id
LOADL_STEP_OWNER	Job step owner
LOADL_STEP_COMMAND	Command
LOADL_STEP_ARGS	Arguments
LOADL_STEP_IN	File for stdin
LOADL_STEP_OUT	File for stdout
LOADL_STEP_ERR	File for stderr
LOADL_STEP_INITDIR	Initial working directory
LOADL_STEP_CLASS	Class
LOADL_STEP_ACCT	Account number
LOADL_STEP_GROUP	Group name
LOADL_STEP_TYPE	Type (SERIAL, PARALLEL, PVM3, NQS)
LOADL_PROCESSOR_LIST	Blank-delimited list of hostnames allocated for the step.

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

Return Code Values

0 The job will begin.

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

Sample Prolog Programs

Sample of a Prolog Program for Korn Shell:

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

Sample of a Prolog Program for C Shell:

```
#!/bin/csh
#
# Set up environment
source /u/loadl/.login
#
setenv PATH  "${PATH}:/loctools/lladmin/bin"
setenv LOG "/tmp/${LOADL_STEP_OWNER}.${LOADL_JOB_ID}.prolog"
#
# Do set up based upon job step class
#
switch ($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.

The following environment variables are set:

LOADL_ACTIVE	LoadLeveler version
LOADL_STEP_ID	Job step id
LOADL_STEP_OWNER	Job step owner
LOADL_STEP_COMMAND	Command
LOADL_STEP_ARGS	Arguments
LOADL_STEP_IN	File for stdin
LOADL_STEP_OUT	File for stdout
LOADL_STEP_ERR	File for stderr
LOADL_STEP_INITDIR	Initial working directory
LOADL_STEP_CLASS	Class
LOADL_STEP_ACCT	Account number
LOADL_STEP_GROUP	Group name
LOADL_STEP_TYPE	Type (SERIAL, PARALLEL, PVM3, NQS)
LOADL_PROCESSOR_LIST	Blank-delimited list of hostnames allocated for the step.

Notes

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

Sample Epilog Programs

Sample of an Epilog Program for Korn Shell:

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

Checkpointing API

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. “Step 13: Enable Checkpointing” on page 113 describes how to checkpoint your jobs in various ways including system-initiated and user-initiated. This section describes the API used for user-initiated checkpointing.

To activate user-initiated checkpointing, code the LoadLeveler **ckpt()** function call in the FORTRAN or C program. Whenever this call is invoked, a checkpoint of the program will be taken.

An include file called **llapi.h** and a LoadLeveler library called **libllapi.a** are used to compile programs and link them. **libllapi.a** can be found in the **lib** subdirectory and **llapi.h** can be found in the **include** subdirectory of the release directory.

C Syntax

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

FORTRAN Syntax

```
call ckpt()
```

The Submit API

Normally, the **llsubmit** command is used to submit jobs to LoadLeveler for scheduling. However, you may also write a program using these APIs to submit jobs for scheduling.

llsubmit API

The function to submit a job is called **llsubmit**.

Note: **llsubmit** is the name of the LoadLeveler command used to submit jobs as well as the function described here.

Syntax

```
int llsubmit(char *job_cmd_file, char *monitor_program, char *monitor_arg,
             LL_job *job_info, int job_version)
```

Where:

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. It is set to NULL if a monitoring program is not provided.

monitor_arg
Is a pointer to a string which is stored in the job proc 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. The string is 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 function **llfree_job_info**.

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.

Return Code Values

0 Job has been submitted.
-1 Job has not been submitted. Error messages are written to stderr.

LoadLeveler must be installed and configured correctly on the machine on which the submit application is run.

The uid and gid in effect when **llsubmit** is invoked is the uid and gid used when the job is run.

Syntax for the llfree_job_info Function:

```
void llfree_job_info(LL_job *job_info, int job_version);
```

Where:

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.

The Monitor API

You can create a monitor program that will monitor jobs that were submitted using the **llsubmit** API. 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.

Syntax

```
monitor_program job_id user_arg state exit_status
```

Where:

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 cancelled 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.

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 APIs. The supported parallel operating environments have already been coded to interface with the APIs described in this section. If you have another application environment that you want to use, you need to use the APIs described here to interface with LoadLeveler.

There are two APIs you can use to allow users to submit parallel jobs, one to acquire the list of LoadLeveler selected parallel nodes and the other is to start the parallel task under the LoadLeveler starter. An include file called **llapi.h** and a LoadLeveler library called **libllapi.a** are used to compile and link programs. **libllapi.a** can be found in the **lib** subdirectory of the release directory and **llapi.h** can be found in the include subdirectory of the release directory.

Before reviewing the parallel APIs, review the following section that describes how parallel job submission works. Understanding this will help you to better understand the APIs.

Interaction Between LoadLeveler and the Parallel APIs

Program applications which use the parallel APIs to interface with LoadLeveler Release 1.3.0 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 the nodes based on the user's requested **min_processors**, **max_processors**, and **requirements** keywords. 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**. This machine will be a resource manager configured machine if one exists. The Master Starter machine is one of the selected parallel nodes. The Master Starter will try to allocate the machines through the resource manager when configured to do so. If allocation fails, the job will be rejected and returned to the queue. All **Starter Slaves** will be terminated.

The Master Starter will start the **executable** specified in the job command file. The executable is your API application and is referred to from now on as the **Parallel Master**. A LOADLBATCH environment variable will be 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 selected 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.

Once all Parallel Slaves have been started, the job is marked Running. Once the Parallel Master has exited, 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** API.
- A SIGCONT, followed by a SIGTERM, is received. A SIGTERM is issued to all slaves. Possible sources include:
  - LoadLeveler receives a job cancel
  - LoadLeveler receives a stop LoadLeveler daemons command
- A SIGCONT, followed by a SIGUSR1, is received. SIGINT is issued to all slaves and the job is retruned to the queue to be redispatched. Possible sources include:
  - The Master Starter has encountered a bad return code while monitoring the resource manager socket.
  - One of the Parallel Slaves terminates abnormally.

When the Parallel Master exits before the Parallel Slaves, LoadLeveler issues a SIGINT to terminate all the Parallel Slaves. A SIGKILL is issued to any process (Master or Slave) which does not exit within five minutes of receiving a termination signal.

ll_get_hostlist API

This API is used to obtain a list of machines from the Master Starter machine so that the Parallel Master can start the Parallel Slaves. The Parallel Master is the LoadLeveler executable specified in the job command file and the Parallel Slaves are the processes started by the Parallel Master through the **ll_start_host** API.

Syntax

```
ll_get_hostlist(struct JM_JOB_INFO* jobinfo)
```

Where **jobinfo** is a pointer to a resource manager compatible structure available in **llapi.h**. No fields are required to be filled in. **ll_get_hostlist** allocates storage for an array of JM_NODE_INFO and returns the pointer in the jm_min_node_info pointer. It is the caller's responsibility to free this storage. **ll_get_hostlist** will update the fields marked in bold. The rest of the fields are used by LoadLeveler when communicating with the resource manager and will not be updated.

```
   struct JM_JOB_INFO {
   int  jm_request_type;
   char  jm_job_description[50];
   enum  JM_ADAPTER_TYPEjm_adapter_type;
   int  jm_css_authentication;
   int jm_min_num_nodes;
   struct JM_NODE_INFO *jm_min_node_info;
};
   struct JM_NODE_INFO {
   char 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;
};
```

Where:

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 in the high-performance switch partition. The second entry is mapped to task 1, and so on. For more information, refer to the *IBM RS/6000 Scalable POWERparallel Systems: Administration Guide*.

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 the High Performance Switch jobs. Otherwise, it is set to -1.

Return Code Values

0 No errors

-2 Cannot get LoadLeveler jobid 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

The Parallel Master must:

- Issue error messages as appropriate.
- Exit when the **ll_get_hostlist** API returns with a negative return value. The Parallel Master exit status is included in the job mail returned to the user.

ll_start_host API

This API starts a task on a selected machine. This function must be invoked for all the machines returned from the **ll_get_hostlist** API once and only once by the Parallel Master. Acquiring the **start_cmd** is the responsibility of the Parallel Master. The user may pass this information through the **arguments** or **environment** keywords in the job command file.

Syntax

```
ll_start_host(char *node_name, char *start_cmd)
```

Where:

node_name
Is the name of the node.

start_cmd
Is the actual command to execute on the node, including flags and arguments.

Return Code Values

fd The socket connected to the Parallel Slave's standard I/O (stdio)

-1 Remote host cannot fork new process

-2 Cannot get LoadLeveler jobid from environment

-3 Cannot get my hostname

-4 Nameserver cannot resolve host

-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

-7 Cannot send PASS_OPEN_SOCKET command to remote startd

The Parallel Master must:

- Issue error messages as appropriate.
- Exit when the **ll_start_host** API returns with a negative return value. The Parallel Master exit status is included in the job mail returned to the user.

Sample Parallel API

A sample program called **para_api.c** is provided in the **samples/llpara** subdirectory of the release directory, usually /**usr/lpp/LoadL/nfs**.

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/nfs/samples/llpara**. For example:

   ```
   char *startCmd = "/home/user/para_api -s";
   ```

3. Issue **make** to create the executable **para_api**.
4. Update your job command file as follows:

```
#!/bin/ksh
# @ initialdir     = /home/user
# @ executable     = para_api
# @ output         = para_api.$(cluster).$(process).out
# @ error          = para_api.$(cluster).$(process).err
# @ job_type       = parallel
# @ min_processors = 2
# @ max_processors = 2
# @ queue
```

5. Submit the job command file to LoadLeveler.

   The syntax to invoke the Parallel Master is:

   ```
   para_api
   ```

   The syntax to invoke the Parallel Slave is:

   ```
   para_api -s
   ```

   The Parallel Master does the following:

   - Acquires the hostlist through the **ll_get_hostlist** API and prints out the returned fields.
   - Starts a Parallel Slave task by executing the command specified in the **StartCmd** variable on all hosts returned in the hostlist.
   - Acquires the socket connected to the Parallel Slave's standard I/O (stdio).
   - Writes a command over the socket to verify stdin.
   - Reads acknowledgments over the socket to verify stderr and stdout.
   - Prints out host names and acknowledgments.

   Example output follows:

```
num_nodes=2

name=host1.kgn.ibm.com address=9.115.8.162 switch_number=-1

name=host2.kgn.ibm.com address=9.115.8.164 switch_number=-1

Connected to host1.kgn.ibm.com at sock 3
Received acko "8000" and acke "10000" from host 0

Connected to host2.kgn.ibm.com at sock 4
Received acko "8001" and acke "10001" from host 1

<Master Exiting>
```

The Parallel Slave does the following:

- Reads command from stdin.
- Writes acknowledgment to stdout and stderr.

Scheduling API

This API allows you to disable the native LoadLeveler scheduling algorithm and "plug in" an external scheduler. The scheduling API consists of two calls: **ll_start_job** and **ll_terminate_job**. **ll_terminate_job** is available to LoadLeveler administrators and to users. **ll_start_job** is available only to LoadLeveler administrators. To use **ll_start_job**, you must specify the following keyword in the global LoadLeveler configuration file:

```
SCHEDULER_API = YES
```

Specifying **YES** disables the native LoadLeveler scheduling algorithm and enables you to use **ll_start_job**. You can then use an external scheduler, in conjunction with **ll_start_job**, to make scheduling decisions. When you disable the LoadLeveler scheduling algorithm, jobs do not start unless requested to do so by **ll_start_job**.

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 this API in conjuction with the query API, which collects information regarding which machines are available and which jobs need to be scheduled. See "Query API" on page 277 for more information.

You can switch between the LoadLeveler scheduling algorithm and an external scheduler in the following ways. If you are running the LoadLeveler scheduler, you can switch to an external scheduler by doing the following:

1. In the configuration file, set **SCHEDULER_API = YES**
2. On the central manager machine, issue the **llctl** command with the **reconfig** option

If you are running an external scheduler, you can re-enable the LoadLeveler scheduling algorithm by doing the following:

1. In the configuration file, set **SCHEDULER_API = NO**
2. On the central manager machine, issue the **llctl** command with the **reconfig** option

Note that the scheduling API automatically connects to an alternate central manager if the API cannot contact the primary central manager.

To use the scheduling API you must include a header file called **llapi.h** in the scheduling application program to allow the program to access structures such as **LL_job** and **LL_node**. The LoadLeveler library **libllapi.a** should be linked to your applications. **llapi.h** is found in the **include** subdirectory, and **libllapi.a** is found in the **lib** subdirectory.

ll_start_job API

This API, available to LoadLeveler administrators only, tells the LoadLeveler negotiator to start a job. To use this API, you must specify **SCHEDULER_API = YES** in the configuration file.

Syntax

```
int ll_start_job(LL_start_job_info *ptr)
```

A pointer to the structure **LL_start_job_info** is required as the input argument. The **LL_start_job_info** structure is defined as follows:

```
typedef struct LL_start_job_info {
   int          version_num;
   LL_STEP_ID   *StepId;
   char         **nodeList;
} LL_start_job_info;
```

Where:

version_num
Should be set to LL_PROC_VERSION

StepId
Is a pointer to **LL_STEP_ID**, which contains the ID of the job to be started.

nodeList
Is a pointer to an array of node names where the job should be started. The first node specified in the array is the parallel master node. The last element in the array is NULL.

Return Code Values

ll_start_job includes the following return codes:

0 The start job request was accepted by the negotiator. However, a return code of 0 does not necessarily imply the job started. You can use the **llq** command to verify the job started.

-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 An error ocurred 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 occured.

-7 The API cannot authorize the action because you are not a LoadLeveler administrator.

-8 The proc version number is incorrect.

-9 The *StepId* is not in the Idle state.

-10 One of the machines specified is not available to run the job.

-11 One of the machines specified does not have an available initiator for the class of the job.

-12 For one of the machines specified, the requirements statement does not satisfy the job requirements.

-13 The number of machines specified was less than the minimum or more than the maximum requested by the job.

-14 The LoadLeveler native scheduler is enabled; that is, **SCHEDULING_API=NO**.

-15 The same machine was specified twice in **ll_start_job** *nodeList*.

ll_terminate_job API

This API can be used by LoadLeveler administrators or the user who submitted the job. It tells the negotiator to cancel a job.

Syntax

```
int ll_terminate_job(LL_terminate_job_info *ptr)
```

A pointer to the structure **LL_terminate_job_info** is required as the input argument. The **LL_terminate_job_info** structure is defined as follows:

```
typedef struct LL_terminate_job_info {
   int          version_num;
   LL_STEP_ID   *StepId;
} LL_terminate_job_info;
```

Where:

version_num
Should be set to LL_PROC_VERSION

StepId
Is a pointer to **LL_STEP_ID**, which contains the ID of the job to be cancelled.

Return Code Values

ll_terminate_job includes the following return codes:

0 The terminiate job request was accepted by the negotiator. However, a return code of 0 does not necessarily imply the negotiator cancelled the job. Use the **llq** command to verify the job was cancelled.

-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 An error ocurred reading parameters from the administration or the configuration file.

-6 A data transmission failure occured.

-7 The API cannot authorize the action because you are not a LoadLeveler administrator or you are not the user who submitted the job.

-8 The proc version number is incorrect.

Sample Scheduling API

A sample program called **sch_api.c** is provided in the **samples**/**llsch** subdirectory of the release directory, which is usually /**usr**/**lpp**/**LoadL**/**nfs**. In order to run this sample you need to do the following:

1. Copy the sample **Makefile** and the sample program called **sch_api.c** to your home directory.
2. Issue **make** to create the executable **sch_api**.

Usage Notes

It is important to know how LoadLeveler keywords and commands behave when you disable the 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 proc or the machine which are used by the LoadLeveler scheduler have their values maintained as before and passed to the query API.
- Keywords used only by the LoadLeveler scheduler have no effect.

The following sections discuss some specific keywords and commands and how they behave when you disable the 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. procs 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 proc 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 29). 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 calulated 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 procs recalulated.

Query API

This API provides information about the jobs and machines in the LoadLeveler cluster. You can use this in conjuction with the scheduling API, since the scheduling API requires you to know which machines are available and which jobs need to be scheduled. See “Scheduling API” on page 273 for more information.

The query API consists of the following calls:

ll_get_jobs API

This API, available to any user, queries LoadLeveler's job queue. Note that 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.

Syntax

```
int ll_get_jobs(LL_get_jobs_info *ptr)
```

A pointer to the structure **LL_get_jobs_info** is required as the input argument. **LL_get_jobs_info** is defined as follows:

```
typedef struct LL_get_jobs_info {
   int          version_num;
   int          numJobs;
   LL_job       **JobList;
} LL_get_jobs_info;
```

Where:

version_num
Returns the version number sent from the negotiator.

numJobs
Returns the number of entries in the array.

JobList
Returns the pointer to an array of **LL_job** structures, provided the return code is 0 and there are jobs in the system. It is the caller's responsibility to call **ll_free_jobs** to free storage allocated for the **LL_job** structures in **ll_get_jobs**. The **LL_job** structure is defined in **llapi.l**.

Return Code Values

ll_get_jobs includes the following return codes:

0 This API call was successful.

-1 There is an error in the input parameter.

-2 The API cannot connect to the central manager.

-3 The API cannot allocate memory.

-6 A data transmission failure occured.

ll_free_jobs API

This API, available to any user, frees storage that was allocated by **ll_get_jobs**.

Syntax

```
int ll_free_jobs(LL_get_jobs_info *ptr)
```

A pointer to the structure **LL_get_jobs_info** is required as the input argument. In **LL_get_jobs_info**, which is described in "ll_get_jobs API" on page 277:

JobList
Is a pointer to the array of **LL_job** structures to be freed.

numJobs
Is the number of **LL_job** structures to be freed.

Return Code Values

ll_free_jobs includes the following return codes:

0 This API call was successful.

8 The proc version is incorrect.

ll_get_nodes API

This API, available to any user, queries the status of nodes known by the negotiator daemon.

Syntax

```
int ll_get_nodes(LL_get_nodes_info *ptr)
```

A pointer to the structure **LL_get_nodes_info** is required as the input argument. **LL_get_nodes_info** is defined as follows:

```
typedef struct LL_get_nodes_info {
   int          version_num;
   int          numNodes;
   LL_node      **NodeList;
} LL_get_nodes_info;
```

Where:

version_num
Returns the version number sent from the negotiator.

nodeNames
Returns the number of entries in the array.

NodeList
Returns the pointer to an array of **LL_node** structures, provided the return code is 0 and there are nodes known by the negotiator. It is the caller's responsibility to call **ll_free_nodes** to free storage allocated for the **LL_node** structures in **ll_get_nodes**.

LL_node is defined as follows:

```
typedef struct LL_node {
   char         *nodename;       /* Name of this node */
   int          version_num;     /* Version number */
   int          configtimestamp; /* Date and time of last reconfig */
   int          virtual_memory;  /* Available swap space in KBs */
   int          memory;          /* Physical memory */
   int          disk;            /* Available space (KB) in execute */
   float        loadavg;         /* Berkely one-minute load average */
   float        speed;           /* Speed associated with this node */
   int          max_starters;    /* Maximum number of jobs allowed */
   int          pool;            /* Pool number for this node */
   int          cpus;            /* Number of CPUs on the node */
   int          *state;          /* State of the node */
   int          keywordidle;     /* No. of seconds since last activity */
   char         **adapter;       /* Name of adapter on this node */
   char         **feature;       /* Set of all features on this node */
   int          totaljobs;       /* Number of jobs submitted to the node */
   char         *arch;           /* Hardware architecture */
   char         *opsys;          /* Operating system used */
   char         **job_class;     /* Job classes allowed on the node */
   LL_STEP_ID   *steplist;       /* procs allocated to run on this node */
} LL_node;
```

Return Code Values

ll_get_nodes includes the following return codes:

0 This API call was successful.

-1 There is an error in the input parameter.

-2 The API cannot connect to the central manager.

-3 The API cannot allocate memory.

-6 A data transmission failure occured.

ll_free_nodes API

This API, available to any user, frees storage that was allocated by **ll_get_nodes**.

Syntax

```
int ll_free_nodes(LL_get_nodes_info *ptr)
```

A pointer to the structure **LL_get_nodes_info** is required as the input argument. In **LL_get_nodes_info**, which is described in "ll_get_nodes API" on page 278:

NodeList
Is a pointer to an array of **LL_node** structures to be freed.

numNodes
Is the number of **LL_node** structures in the array to be freed.

Return Code Values

ll_free_nodes includes the following return codes:

0 This API call was successful.

8 The proc version is incorrect.

User Exits

This section discusses separate user exits for handling an AFS token, filtering a job script, and overriding the default mail notification method.

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 started and can be used to revive a token which expired while the job was being queued.

The program uses the following structures:

ktc_principal-service
ktc_principal-client
ktc_token-token

The ktc_principal and ktc_token structures are defined in the AFS header files **afs/auth.h** and **rx/rxkad.h**.

The following environment variables are set when the you invoke this program:

LOADL_STEP_OWNER The owner of the job

LOADL_STEP_LIMIT	The number of seconds the job is scheduled to run.

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 cancelled.

The following environment variables are set when the program is invoked:

LOADL_ACTIVE	LoadLeveler version
LOADL_CPU_LIMIT	CPU limit
LOADL_STEP_COMMAND	Job command file name
LOADL_STEP_OWNER	Job step owner
LOADL_STEP_CLASS	Class
LOADL_STEP_ID	Job step ID

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.

Appendixes

© Copyright IBM Corp. 1993, 1996

Appendix A. 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 289 for more information.
  - **The central manager is not operating.** See "What Happens if the Central Manager Isn't Operating?" on page 290 for more information.
  - **Miscellaneous questions.** See "Other Questions" on page 292 for more information.
- "Helpful Hints" on page 293, which contains tips on running LoadLeveler, including some productivity aids.
- "Getting Help from IBM" on page 297, 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 10: Record and Control Log Files" on page 110 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:

© Copyright IBM Corp. 1993, 1996

Why Your Job May Not Be Running:	Possible Soulution
Job requires specific machine, operating system, or other resource.	• Does the resource exist in the LoadLeveler cluster? If yes, wait until it becomes available. Check the GUI to compare the job requirements to the machine details, especially **Arch**, **OpSys**, and **Class**. Ensure that the spelling and capitalization matches.
Job requires specific job class	• Is the class defined in the administration file? Use **llclass** to determine this. If yes, • Is there a machine in the cluster that supports that class? If yes, you need to wait until the machine becomes available to run your job.
The maximum number of jobs are already running on all the eligible machines	Wait until one of the machines finishes a job before scheduling your job.

The start expression evaluates to false.	Examine the configuration files (both **LoadL_config** and **LoadL_config.local**) to determine the **START** control function expression used by LoadLeveler to start a job. As a problem determination measure, set the START and SUSPEND values, as shown in this example:
	`START: T`
	`SUSPEND: F`

The priority of your job is lower than the priority of other jobs.	You cannot affect the system priority given to this job by the negotiator daemon but you can try to change your user priority to move this job ahead of other jobs you previously submitted using the **llprio** command or the GUI.
The information the central manager has about machines and jobs may not be current.	Wait a few minutes for the central manager to be updated and then the job may be dispatched. This time limit (a few minutes) depends upon the polling frequency and polls per update set in the **LoadL_config** file. The default polling frequency is five minutes.
You do not have the same user ID on all the machines in the cluster.	To run jobs on any machine in the cluster, you have to have the same user ID and the same uid number on every machine in the pool. If you do not have a userid on one machine, your jobs will not be scheduled to that machine.

You can use the **llq** command to query the status of your job or the **llstatus** command to query the status of machines in the cluster. Refer to Chapter 10, "LoadLeveler Commands" on page 185 for information on these commands.

Why Won't My Parallel Job Run?

If you submitted your parallel job and it is in the LoadLeveler queue but has not run, issue **llq -s** first to help diagnose the problem. If issuing this command does not help, refer to the previous table and to the following table for more information:

Why Your Job May Not Be Running	Possible Solution
The minimum number of processors requested by your job is not available.	Sufficient resources must be available. Specifying a smaller number of processors with the **min_processors** keyword may help if your job can run with fewer resources.
The pool in your **requirements** statement specifies a pool which is invalid or not available.	The specified pool must be valid and available.
The adapter specified in your **requirements** statement specifies an adapter which is invalid or not available.	The specified adapter must be valid and available.
PVM3 is not installed	PVM3 must be installed on any machine you wish to use for pvm. The PVM3 system itself is not supplied with LoadLeveler.
You are already running a PVM3 job on one of the LoadLeveler machines.	PVM3 restrictions prevent a user from running more than one pvm daemon per user per machine. If you want to run pvm3 jobs on LoadLeveler, you must not run any pvm3 jobs outside of LoadLeveler control on any machine being managed by LoadLeveler.
The **parallel_path** keyword in your job command file is incorrect.	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.
The **pvm_root** keyword in the administration file is incorrect.	This keyword corresponds to the pvm **ep** keyword and is required to tell LoadLeveler where the pvm system is installed.
The file /**tmp**/**pvmd.***userid* exists on some LoadLeveler machine but no PVM jobs are running.	If PVM3 exits unexpectedly, it will not properly clean up after itself. Although LoadLeveler attempts to clean up after pvm, some situations are ambiguous and you may have to remove this file yourself. Check all the systems specified as being capable of running PVM3, and remove this file if it exists.

Common Set Up Problems with Parallel Jobs

This section presents a list of common problems found in setting up parallel jobs:

- If jobs that need resource manager adapters remain in an Idle state:
  - Make sure LoadLeveler machines do *not* specify **rm_host=false**
  - If LoadLeveler machines specify **rm_host=true**:
    - LoadLeveler may not have been globally reconfigured since you added **rm_host=true**
    - Check that the Resource manager is running (use **jm_status -P**)
    - Check that machines are specified as general pool members or as batch pool members in **jmd_config** (use **jm_status -P**)
- 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*

Common Submit Problems with Parallel Jobs

- For POE:
  - Specify the POE partition manager as the executable. Do *not* specify the parallel job as the executable.
  - Pass the parallel job as an argument to POE.
  - The parallel job must exist and must be specified as a full path name.
  - If the job runs in user space, specify the flag **-euilib us**.
  - Specify the correct adapter (when needed).
  - Specify a POE job only once in the job command file.
  - Compile only with the supported level of POE.
  - Specify only **parallel** as the *job_type*.

- For PVMe:
  - Do *not* invoke the PVMe daemon from the **executable** keyword.
  - Pass the parallel job to PVMe as an **-exec** argument.
  - The parallel job must exist and must be specified as a full path name.
  - You must specify a user space adapter.
  - Do *not* specify adapter **hps_ip**.
  - Specify a PVMe job only once in the job command file.
  - Compile only with the supported level of PVMe.
  - 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 cancelled.

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:

- 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 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
```

For other submit-only requirements, see the submit-only section on page 82.

Why Does a Job Stay in the Pending (or Starting) State?

If a job appears to stay in the Pending or Starting state, it is possible the job is continually being dispatched and rejected. Check the setting of the **MAX_JOB_REJECT** keyword. If it is set to the default, -1, the job will be rejected an unlimited number of times. Try resetting this keyword to some finite number. Also, check the setting of the **ACTION_ON_MAX_REJECT** keyword. These keywords are described in "Additional Configuration File Keywords" on page 119.

Can the startd Daemon be Temporarily Unavailable?

When you start a host machine, you may see that startd is listed as "unavailable" for a short period of time even though schedd is "available".

This delay may occur when you have **rm_host** set to **true** for this host in the **LoadL_admin** file (when the Resource Manager isn't running on the host). To solve this problem, set **rm_host** to **false**.

What Happens to Running Jobs When a Machine Goes Down?

Both the startd daemon and the schedd daemon maintain persistent states of all jobs. Both daemons use a specific protocol to ensure that the state of all jobs is consistent across LoadLeveler. In the event of a failure, the state can be recovered. Neither the schedd nor the startd daemon discard the job state information until it is passed onto and accepted by another daemon in the process.

If	Then
The network goes down but the machines are still running	If the network goes down but the machines are still running, when LoadLeveler is restarted, it looks for all jobs that were marked running when it went down. On the machine where the job is running, the startd daemon searches for the job and if it can verify that the job is still running, it continues to manage the job through completion. On the machine where schedd is running, schedd queues a transaction to the startd to re-establish the state of the job. This transaction stays queued until the state is established. Until that time, LoadLeveler assumes the state is the same as when the system went down.
The network partitions or goes down.	All transactions are left queued until the recipient has acknowledged them. Critical transactions such as those between the schedd and startd are recorded on disk. This ensures complete delivery of messages and prevents incorrect decisions based on incomplete state information.
The machine with startd goes down.	Because job state is maintained on disk in startd, when LoadLeveler is restarted it can forward correct status to the rest of LoadLeveler. In the case of total machine failure, this is usually "JOB VACATED", which causes the job to be restarted elsewhere. In the case that only LoadLeveler failed, it is often possible to "find" the job if it is still running and resume management of it. In this case LoadLeveler sends JOB RUNNING to the schedd and central manager, thereby permitting the job to run to completion.

If	Then
The central manager machine goes down.	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. However, on a SP machine where the central manager is also the control workstation with nodes having **rm_host** set to **true**, loss of the control workstation results in running jobs being vacated.
The machine running the job goes down	The central manager thinks that the job is still running. When the machine comes back up, the job gets put back on the queue to be run again. This happens only if **restart** is set to yes.
The schedd machine goes down	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. 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. During the time the schedd is down, the central manager will not be able to start new jobs that were submitted to that schedd
The **llsubmit** machine goes down	schedd gets its own copy of the executable so it does not matter if the **llsubmit** machine goes down.

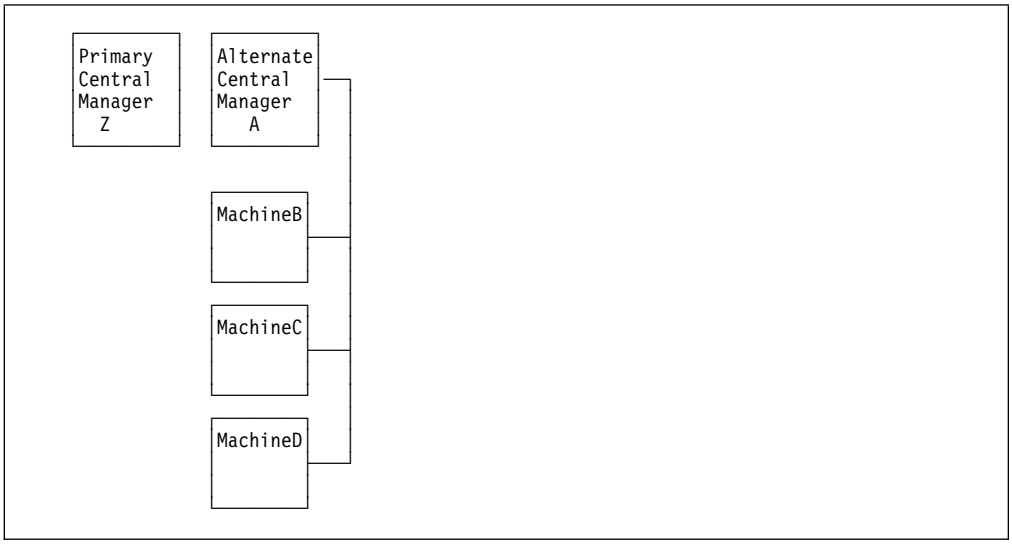
Why Does llstatus Indicate that a Machine is Down when llq Indicates a Job is Running on The Machine?

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.

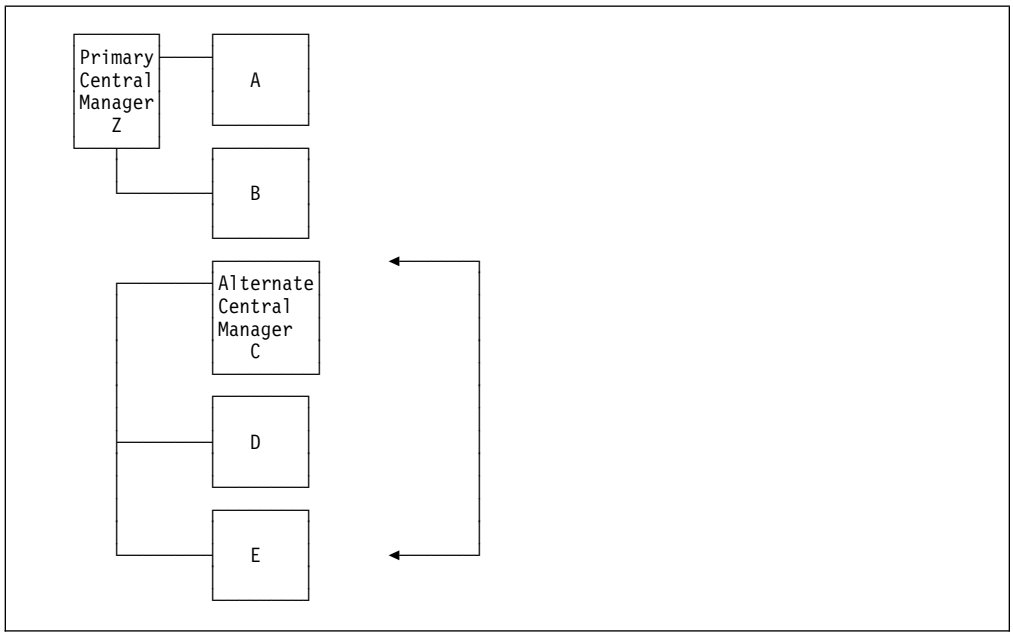
Once an alternate central manager takes control, it starts up its negotiator daemon and notifies all of the other machines in the LoadLeveler cluster that a new central manager has been selected. The following diagram illustrates how a machine can become the alternate central manager:



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:

- 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 78

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 non-checkpointing batch job to LoadLeveler, the operating system will execute your **.profile** script before executing the batch job if your login shell is the Korn shell. On the other hand, if your login shell is the Bourne shell, on most operating systems (including AIX), the **.profile** script is not executed. Similarly, if your login shell is the C shell then AIX will execute your **.login** script before executing your LoadLeveler batch job but some other variants of UNIX may not invoke this script.

The reason for this discrepancy is due to the interactions of the shells and the operating system. To understand the nature of the problem, examine the following C program that attempts to open a login Korn shell and execute the "ls" command:

```
#include <stdio.h>
main()
{
  execl("/bin/ksh","-","-c","ls",NULL);
}
```

UNIX documentations in general (SunOS, HP-UX, AIX, IRIX) give the impression that if the second argument is "-" then you get a login shell regardless of whether the first argument is /bin/ksh or /bin/csh or /bin/sh. In practice, this is not the case. Whether you get a login shell or not is implementation dependent and varies depending upon the UNIX version you are using. On AIX you get a login shell for /bin/ksh and /bin/csh but not the Bourne shell.

If your login shell is the Bourne shell and you would like the operating system to execute your **.profile** script before starting your batch job, add the following statement to your job command file:

```
# @ shell = /bin/ksh
```

LoadLeveler will open a login Korn shell to start your batch job which may be a shell script of any type (Bourne shell, C shell, or Korn shell) or just a simple executable.

Helpful Hints

This section contains tips on running LoadLeveler, including some productivtiy aids.

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:   0 01:08:29
     Job Step User Time:   0 00:30:15
   Job Step System Time:   0 00:12:55
    Total Job Step Time:   0 00:43:10

      Starter User Time:   0 00:00:00
    Starter System Time:   0 00:00:00
     Total Starter Time:   0 00:00:00
```

This mail tells you the following:

Submitted at The time you issued the **llsubmit** command or the time you submitted the job with the graphical user interface.

Started at The time the starter process executed the job.

Exited at The actual time your job completed.

Real Time The wall clock time from submit to completion.

Job Step User Time The CPU time the job consumed executing in user space.

Job Step System Time The CPU time the system (AIX) consumed on behalf of the job.

Total Job Step Time The sum of the two fields above.

Starter User Time The CPU time consumed by the LoadLeveler starter process for this job, executing in user space. Time consumed by the starter process is the only LoadLeveler overhead which can be directly attributed to a user's job.

Starter System Time The CPU time the system (AIX) consumed on behalf of the LoadLeveler starter process running for this job.

Total Starter Time The sum of the two fields above.

You can also get the starting time by issing **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 8:00AM 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: Running, Pending, or Starting. These keywords, which are described in detail in "Step 2: Specify User Stanzas" on page 83, are:

maxqueued
Controls the number of jobs in any of these states: Idle, Running, Pending, or Starting.

maxjobs
Controls the number of jobs in any of these states: Running, Pending, or Starting; thus it controls a subset of what **maxqueued** controls. **maxjobs** effectively controls the number of jobs in the Running state, since Pending and Starting are usually temporary states.

maxidle
Controls the number of jobs in any of these states: Idle, Pending, or Starting; thus it controls a subset of what **maxqueued** controls. **maxidle** effectively controls the number of jobs in the Idle state, since Pending and Starting are usually temporary states.

What Happens When You Submit a Job: For a user's job to be allowed into the job queue, the total of other jobs (in the Idle, Pending, Starting and Running states) for that user must be less than the **maxqueued** value for that user. Also, the total idle jobs (those in the Idle, Pending, and Starting states) must be less than the **maxidle** value for the user. If either of these constraints are at the maximum, the job is placed in the Not Queued state until one of the other jobs changes state. If the user is at the **maxqueued** limit, a job must complete, be cancelled, 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 cancelled, 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 cancelled. (An exception to this is parallel jobs which do not accumulate sufficient machines in a given time period. These jobs are moved to the Deferred state, meaning they must vie for the queue when their Deferred period expires.)

Once a job is in the queue, the job will run unless the **maxjobs** limit for the user is at a maximum.

Note the following restrictions for using these keywords:

- If **maxqueued** is greater than (**maxjobs** + **maxidle**), the **maxqueued** value will never be reached.
- If either **maxjobs** or **maxidle** is greater than **maxqueued**, then **maxqueued** will be the only restriction in effect, since **maxjobs** and **maxidle** will never be reached.

Sending Output from Several Job Steps to One Output File

You can use dependencies in your job command file to send the output from many job steps to the same output file. For example:

```
# @ step_name = step1
# @ executable = ssba.job
# @ output = ssba.tmp
# @ ...
# @ queue
#
# @ step_name = append1
# @ dependency = (step1 != CC_REMOVED)
# @ executable = append.ksh
# @ output = /dev/null
# @ queue
# @
# @ step_name = step2
# @ dependency = (append1 == 0)
# @ executable = ssba.job
# @ output = ssba.tmp
# @ ...
# @ queue
# @
# @ step_name = append2
# @ dependency = (step2 != CC_REMOVED)
# @ executable = append.ksh
# @ output = /dev/null
# @ queue
#
# ...
```

Then, the file **append.ksh** could contain the line **cat ssba.tmp >> ssba.log**. All your output will reside in **ssba.log**. (Your dependecies can look for different return values, depending on what you need to accomplish.)

You can achieve the same result from within **ssba.job** by appending your output to an output file rather than writing it to **stdout**. Then your output statement for each step would be /**dev**/**null** and you wouldn't need the append steps.

Hints for Using Machines

Setting Up a Single Machine To Have Multiple Job Classes

You can define a machine to have multiple job classes which are active at different times. For example, suppose you want a machine to run jobs of Class A any time, and you want the same machine to run Class B jobs between 6 p.m. and 8 a.m.

You can combine the **Class** keyword with a user-defined macro (called **Off_shift** in this example).

For example:

```
Off_Shift = ((tm_hour >= 18) || (tm_hour < 8))
```

Then define your START statement:

```
START : (Class == "A") || ((Class == "B") && $(Off_Shift))
```

Make sure you have the parenthesis around the **Off_Shift** macro, since the logical OR has a lower precedence than the logical AND in the START statement.

Also, to take weekends into account, code the following statements. Remember that Saturday is day 6 and Sunday is day 0.

```
Off_Shift = ((tm_wday == 6) || (tm_wday == 0) || (tm_hour >=18) \
            || (tm_hour < 8))

Prime_Shift = ((tm_wday != 6) && (tm_wday != 0) && (tm_hour >= 8) \
              && (tm_hour < 18))
```

Reporting the Load Average on Machines

You can use the /**usr**/**bin**/**rup** command to report the load average on a machine. The **rup** *machine_name* command gives you a report that looks similar to the following:

```
   localhost    up 23 days, 10:25,    load average: 1.72, 1.05, 1.17
```

You can use this command to report the load average of your local machine or of remote machines. Another command, /**usr**/**bin**/**uptime**, returns the load average information for only your local host.

History Files and schedd

The **schedd** daemon writes to the spool/history file only when a job is completed or removed. Therefore, you can delete the history file and restart **schedd** even when some jobs are scheduled to run on other hosts.

However, you should clean up the **spool/job_queue.dir** and **spool/job_queue.pag** files only when no jobs are being scheduled on the machine.

You should not delete these files if there are any jobs in the job queue that are being scheduled from this machine (for example, jobs with names such as *thismachine.clusterno.jobno*).

Getting Help from IBM

Should you require help from IBM in resolving a LoadLeveler problem, you can get assistance by calling IBM Support. Before you call, be sure you have the following information:

1. Your access code (customer number).
2. The LoadLeveler product number for the:
   - RISC System/6000 and SP is 5765-145
   - Sun is 5765-227
   - SGI is 5765-228
   - HP is 5765-287
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-CALL-AIX (800-225-5249) or 817-962-6902 and select option 8.

The Facsimile number is 817-962-6723.

Office hours are from 8:00 a.m. - 6:00 p.m. Central time Monday through Friday, excluding holidays. A voice mail message may be left if calling other hours. Note that customers can receive service 24 hours a day/every day under special agreement with IBM.

For specific information regarding AIX and AIX product support, ask your IBM representative to provide you with support information from the following document:

Title AIX SUPPORT FAMILY ENHANCEMENTS AND NEW AIX TECHNICAL SUPPORT STRUCTURE

Number 694-018

Date July 26, 1994

Category GI00, GI10, GD00, GD70

Appendix B. Customer Case Studies

This chapter gives you an overview, including configuration information, of some LoadLeveler customers. These profiles are meant to highlight how customers in different industries use LoadLeveler.

Customer 1: Circuit Simulation

This customer performs CPU-intensive work in the area of circuit simulation using Electronic Design Automation (EDA).

System Configuration

The customer has 752 batch servers; 209 are dedicated to run LoadLeveler jobs 24 hours a day (the central manager is excluded). The rest are used by LoadLeveler when they are not in use by their respective owners.

The LoadLeveler administrators control all the 173 dedicated machines. That means that users cannot get onto these systems without submitting a LoadLeveler job. 117 of the dedicated machines are public schedulers. The user machines are submit-only machines, and users do not have access to their root password. If a user needs root access to his or her machine, he or she is allowed alternate root access only; he or she cannot get global root access to all the machines on site. (Site administrators use a common global root password.)

This site runs over 31,000 jobs per week and about 2,800 CPU days of resource utilization. The central manager is a RISC/System 6000 model 370 with 128MB of RAM. The batch machines are generally 80 percent busy. The central manager is about 35 percent to 70 percent busy. The central manager does not run any jobs, it just manages. All of the LoadLeveler machines run one job at a time. (That is, **MAX_STARTERS=1**.)

This customer sees some machines in a down state occassionally. The administrator feels the CPU on these machines are too busy to get a time slice to report its state to the central manager. However, this down state does not cause any problem for this customer.

117 public schedulers are subset of our 173 dedicated machines and are listed in the admin file.

LoadLeveler Configuration

The following figures represent sections of this customer's **LoadL_admin** file for dedicated machines. Notice the default stanza. Also, every machine in the LoadLeveler cluster is listed in this file.

© Copyright IBM Corp. 1993, 1996

```
#--------------------------------------------------------------------------------#
# type = machine default stanza
#--------------------------------------------------------------------------------#

default:  type = machine             # defaults for machine stanzas
          central_manager = false    #  no central manager on machine
          schedd_host = true         #  public schedd on machine
#--------------------------------------------------------------------------------#
# Central Manager
#--------------------------------------------------------------------------------#

mips1:    type = machine          # PRIMARY server - MANAGER   370 128M 3.2.5
          central_manager = true  #  runs negotiator
#--------------------------------------------------------------------------------#
#                               Primary Servers
#--------------------------------------------------------------------------------#

beast100: type = machine
# PRIMARY C=a/b/o/s2/t2           . . 550    128M 3.2.5
beast101: type = machine
# PRIMARY C=a/b/b1/b4/c/o/r/s/t   F . 550    128M 3.2.5
beast102: type = machine
# PRIMARY C=a                     F . 550    128M 3.2.5
beast103: type = machine
# PRIMARY C=a                     . . 550    128M 3.2.5
```

Later in the **Loadl_admin file**, user machines are defined. Notice the default stanza.

```
#--------------------------------------------------------------------------------#

default:  type = machine             # defaults for machine stanzas
          central_manager = false    #  no central manager on machine
          schedd_host = false        #  no public schedd on machine
#--------------------------------------------------------------------------------#

agni:      type = machine
# SECONDARY server - rmkohn           550    64M 3.2.5
akama:     type = machine
# SECONDARY server - poulter          365    64M 3.2.5
alaska:    type = machine
# SECONDARY server - jcahill          340    64M 3.2.5
alcor:     type = machine
# SECONDARY server - drolson          340    64M 3.2.5
```

The following represents a local configuration file for a dedicated, public scheduler machine:

```
#                        PRIMARY LoadL SERVER  -> mips27
#
# this loadl.config.local is tuned for a machine that is part of a compute
# farm.  Interactive users are discouraged.
#
# Run up to one jobs at a time.
#
# Always start a job if there is a class available.
#
# Never suspend a job.
#
# Since jobs never get suspended they never get vacated or killed.
#

SCHEDD_RUNS_HERE    = True
STARTD_RUNS_HERE    = True

Class = { "a" "b" "b1" "b4" "c" "k" "r" "s" "t" }
Feature = { "PRI" }

MAX_STARTERS = 1

POLLING_FREQUENCY       = 30
POLLS_PER_UPDATE        = 15

START           : T
SUSPEND         : F

START_DAEMONS = True
X_RUNS_HERE   = False
```

The following represents a local configuration file for a user's machine.

```
#                       SECONDARY SERVER --> common
#
# This loadl_config.local is tuned to be "nice" to a workstation owner
# who permits loadl jobs on his system but wants good response whenever
# he is doing his own work.
#
# Run only one LoadLeveler job at a time.
#
# Check the keyboard for activity every five seconds.
#
#
# Suspend a job if the load average exceeds 1.4
#
# Continue a job when keyboard again goes idle for 10 minutes and the load
# average is <.5

SCHEDD_RUNS_HERE  = False
STARTD_RUNS_HERE  = True

Class = { "a" "b" "b1" "b4" "c" "o" "r" "s" "t" }
MAX_STARTERS = 1

START           : $(FirstShift_KB9999) && $(StartS1) || ($(Off_Shift) ||
$(Week_End)) && $(Mach_Idle_S)
SUSPEND         : $(CPU_Busy) || $(KeyboardBusy)
CONTINUE        : $(Mach_Idle_C)
VACATE          : ((Class == "a") && $(Vacate_A)) || ($(Vacate_ClassesB)
&& $(Vacate_B)) || $(Vacate_X)
KILL            : $(Kill_Job)

START_DAEMONS = True
X_RUNS_HERE   = True
```

Customer 2: High-Energy Physics

This scientific customer provides experimental facilities for physicists from its 17 member states and for visiting scientists from throughout the world. The computing requirements of these users vary from mail and text processing to heavy batch and parallel processing.

System Configuration

Their processor is an SP2 using RISC System/6000 nodes linked by an internal high-speed network with a centrally managed software environment. The nodes are functionally divided into four groups of 16 each for different types of work: interactive logins, sequential job batch processing, parallel job batch processing and data, and tape and network services.

This customer uses AFS heavily. It provides the single system image for users' home directories and the files common to their experiments. Many software products are served directly out of AFS using symbolic links.

LoadLeveler provides this customer with the following facilities:

- Interactive load balancing of users across nodes on the SP2 and other UNIX services on site.
- Batch services for serial compute jobs.
- Scheduling for parallel applications.

LoadLeveler Batch Configuration

The batch configuration is designed to maximize short job turnaround while allowing the heavy CPU jobs to get good usage of the resources available.

The basic configuration uses a range of classes – `short`, `medium`, `long` and `verylong` – with a range of maximum job CPU times of from five minutes to six days. An additional class, `night`, provides off-peak and weekend computing time on the interactive areas of the SP2 during periods of low demand. Access to this class is limited to specific users.

Users in different experiments are defined in LoadLeveler groups which provide associated queue priorities. This allows groups with a large computing budget to be given higher priorities. An automated procedure calculates each group's resource utilization over the last month and adjusts their priorities accordingly. This ensures a fair allocation of CPU time among the groups.

LoadLeveler Interactive Configuration

This customer uses the Interactive Session Support facility to provide a name servier which returns the least loaded node according to a site defined metric. This allows a user to be given the least loaded operational node when he or she logs in.

This metric is based on the number of logged in users, with some weight given to those using Xstations. Every few minutes, the system is scanned to evaluate the following:

*Xterminals*3 + *Telnet*2 + *Process*

Where:

- *Xterminals* is the number of users logged in from an Xstation
- *Telnet* is the number logged in via **telnet** or **rlogin**
- *Process* is the number of users who have processes running.

This metric tries to balance users across the system while providing some factor for their likely future utilization. A metric based on the CPU load average is too dependent on the current load to provide good balancing.

The metric can also be set to return a low priority if the file /**etc**/**iss.nologin** exists. This allows the administrator to drain the interactive use of a node if there is scheduled system maintenance. When the maintenance is completed, the file can be removed and the metric will return the correct value for the node. Users will therefore see an improved availability, since they will not be given a node that is about to shutdown.

Processor Configuration

The processors are configured as follows:

- **parallel** nodes support a mixture of short, medium, long, and verylong classes.
- **batch** nodes support the same class mix as parallel. Additional paging space is available on these nodes to provide multiple jobs running per node.
- **interactive** nodes support the night class only. The night class only allows jobs to start after 6 PM and before midnight during the week and anytime on weekends. A maximum CPU time of 8 hours ensures that the jobs are finished

when the prime shift starts. This is configured using LoadLeveler's START expression:

```
Is_Weekend          = (tm_wday==0  || tm_wday==6)
Is_Start_Night_Time = (tm_hour>18)

START: $(Is_Start_Night_Time) || $(Is_Weekend)
```

Customer 3: Computer Chip Design

This customer uses EDA to perform work in the area of computer chip design.

System Configuration

The customer has seven clusters of RISC/System 6000 machines. The largest cluster has 530 machines; the smallest cluster has 87 machines. The total number of machines at this installation is over 1200.

Interactive Configuration

This customer has defined two configuration files for interactive work: one for standard workstations and one for large interactive servers. These files are meant to be tailored to machines of differing processing power.

Standard Workstation Configuration

```
# ------------------------------------------------------------------------ #
# Description: LoadL_config.local for Standard Workstations (<370 Class)
# ------------------------------------------------------------------------ #

# Need 2x Paging Space to Real Memory ( minimum ) For Worst Case Of One
# Suspended and One Foreground Running Job.
#    *) All Jobs (btv,lp) Suspend on LoadAvg or Keyboard/Mouse Movement.
# ------------------------------------------------------------------------ #

# ------------------------------------------------------------------------ #
# Class defines the permissable classes, MAX_STARTERS defines the max
# total jobs to be permitted.
# ------------------------------------------------------------------------ #
  Class        = { "btv" "lp" }
  MAX_STARTERS = 1

# ------------------------------------------------------------------------ #
# The next definitions are used in the expressions below to regulate the
# conditions under which jobs get started, suspended, and evicted.
#
#     All times are specified in units of seconds.
# ------------------------------------------------------------------------ #
  BackgroundLoad   = 0.8
  HighLoad         = 1.6
  StartIdleTime    = 900
  ContinueIdleTime = 900
```

```
# ------------------------------------------------------------------------ #
# LoadAvg is an internal variable whose value is the (Berkeley) load average
# of the machine.
#
#    CPU_Idle - No LoadL job running, or One job just finishing.
#    CPU_Busy - One LoadL job running, second job ( Foreground or Batch )
#               starting up.
#    CPU_Max  - Two LoadL jobs running.
# ------------------------------------------------------------------------ #
  CPU_Idle = (LoadAvg <= $(BackgroundLoad))
  CPU_Busy = (LoadAvg >= $(HighLoad))

# ------------------------------------------------------------------------ #
# This defines a boolean "KeyboardBusy" whose value is TRUE if the keyboard
# or mouse has been used since loadl last checked.  Thus if POLLING_FREQUENCY
# is 5 seconds, KeyboardBusy is TRUE if anybody has used the kbd or mouse in
# the last 5 seconds.
# ------------------------------------------------------------------------ #
  KeyboardBusy = KeyboardIdle < $(POLLING_FREQUENCY)

# ------------------------------------------------------------------------ #
# This statement indicates when a job should be started on this machine
# ------------------------------------------------------------------------ #
  Weekend   = ( (tm_wday >=  6) || (tm_wday <  1) )
  Day       = ( (tm_hour >=  7) && (tm_hour < 18) )
  Night     = ( (tm_hour >= 18) || (tm_hour <  4) )
  Inactive  = ( (KeyboardIdle > $(StartIdleTime)) && $(CPU_Idle) )

  HP        = ( (Class == "btv") )
  LP        = ( ($(Weekend) || $(Night)) )

  START     : ( ($(HP) || $(LP)) && $(Inactive) )

# ------------------------------------------------------------------------ #
# The SUSPEND statement here says that a job should be suspended but not
# killed if:
#                LoadAvg >= 1.6  Or  KeyboardIdle < 5
# ------------------------------------------------------------------------ #
  SUSPEND  : ( $(CPU_Busy) || $(KeyboardBusy) )

# ------------------------------------------------------------------------ #
# This CONTINUE statement indicates that a suspended job should be continued
# if the cpu goes idle and the keyboard/mouse has not been used for the last
# 15 minutes.
# ------------------------------------------------------------------------ #
  CONTINUE : $(CPU_Idle) && KeyboardIdle > $(ContinueIdleTime)

# ------------------------------------------------------------------------ #
# Jobs in the SUSPEND state are never killed, after 60 minutes they are
# relocated to a different machine if possible.
# ------------------------------------------------------------------------ #
  MaxSuspendTime   = 60 * $(MINUTE)
  VACATE    : $(StateTimer) > $(MaxSuspendTime)
  KILL      : F
```

```
# ------------------------------------------------------------------------ #
# If you set START_DAEMONS to False loadl can never start on this machine.
# For example you may want to stop loadl for a couple days for maintenance
# and make sure no procedure automatically restarts it.
# ------------------------------------------------------------------------ #
  START_DAEMONS = True

# ------------------------------------------------------------------------ #
# Set the maximum size each of the logs can reach before wrapping.
# ------------------------------------------------------------------------ #
  MAX_SCHEDD_LOG    = 128000
  MAX_COLLECTOR_LOG = 128000
  MAX_STARTD_LOG    = 128000
  MAX_SHADOW_LOG    = 128000
  MAX_KBDD_LOG      = 128000
```

Large Interactive Server Configuration

```
# ------------------------------------------------------------------------ #
# Description: LoadL_config.local for Interactive Large Servers (580-590 Class)

# ------------------------------------------------------------------------ #
# Need 3x Real Memory To Paging Space ( minimum ) For Worst Case Of Two
# Suspended and One Foreground Running Job.
#    *) All Jobs (btv,lp) Suspend on LoadAvg or Keyboard/Mouse Movement.
#    *) Real Memory >= 192meg.
# ------------------------------------------------------------------------ #

# ------------------------------------------------------------------------ #
# Class defines the permissable classes, MAX_STARTERS defines the max
# total jobs to be permitted.
# ------------------------------------------------------------------------ #
  Class        = { "btv" "lp" }
  MAX_STARTERS = 2

# ------------------------------------------------------------------------ #
# The next definitions are used in the expressions below to regulate the
# conditions under which jobs get started, suspended, and evicted.
#
#     All times are specified in units of seconds.
# ------------------------------------------------------------------------ #
  BackgroundLoad   = 0.8
  LowLoad          = 1.0
  HighLoad         = 1.6
  MaxLoad          = 2.0
  StartIdleTime    = 900
  ContinueIdleTime = 900
```

```
# ------------------------------------------------------------------------ #
# LoadAvg is an internal variable whose value is the (Berkeley) load average
# of the machine.
#
#    CPU_Idle - No LoadL job running, or One job just finishing.
#    CPU_Busy - One LoadL job running, second job ( Foreground or Batch )
#               starting up.
#    CPU_Max  - Two LoadL jobs running.
# ------------------------------------------------------------------------ #
  CPU_Idle = (LoadAvg <= $(BackgroundLoad))
  CPU_Run  = (LoadAvg <= $(LowLoad))
  CPU_Busy = (LoadAvg >= $(HighLoad))
  CPU_Max  = (LoadAvg >= $(MaxLoad))

# ------------------------------------------------------------------------ #
# This defines a boolean "KeyboardBusy" whose value is TRUE if the keyboard
# or mouse has been used since loadl last checked.  Thus if POLLING_FREQUENCY
# is 5 seconds, KeyboardBusy is TRUE if anybody has used the kbd or mouse in
# the last 5 seconds.
# ------------------------------------------------------------------------ #
  KeyboardBusy = KeyboardIdle < $(POLLING_FREQUENCY)
# ------------------------------------------------------------------------ #
# This statement indicates when a job should be started on this machine
# ------------------------------------------------------------------------ #
  Weekend   = ( (tm_wday >=  6) || (tm_wday <  1) )
  Day       = ( (tm_hour >=  7) && (tm_hour < 18) )
  Night     = ( (tm_hour >= 18) || (tm_hour <  4) )
  Inactive1 = ( (KeyboardIdle > $(StartIdleTime)) )
  Inactive2 = ( (KeyboardIdle > $(ContinueIdleTime)) )

  HP        = ( (Class == "btv") )
  LP        = ( (Class == "lp") && $(CPU_Idle) )

  START     : ( ($(HP) || $(LP)) && $(Inactive1) )

# ------------------------------------------------------------------------ #
# The SUSPEND statement here says that a job should be suspended but not
# killed if:
#                KeyboardIdle < 5                Or
#                lp  Class  And  LoadAvg >= 1.6   Or
#                btv Class  And  LoadAvg >= 2.0
# ------------------------------------------------------------------------ #
  SUSPEND   : ( ( (Class == "lp")  && $(CPU_Busy) ) || \
                ( (Class == "btv") && $(CPU_Max)  ) || \
                (  $(KeyboardBusy)                )    )

# ------------------------------------------------------------------------ #
# This CONTINUE statement indicates that a suspended job should be continued
# if:
#               lp  Class  And  LoadAvg <= 0.8  And  KeyboardIdle > 15 min  Or
#               btv Class  And  LoadAvg <= 1.0  And  KeyboardIdle > 15 min
# ------------------------------------------------------------------------ #
  CONTINUE  : ( ( (Class == "lp")  && $(CPU_Idle) && $(Inactive2) ) || \
                ( (Class == "btv") && $(CPU_Run)  && $(Inactive2) )    )
```

```
# ------------------------------------------------------------------------ #
# Jobs in the SUSPEND state are never killed, after 60 minutes they are
# relocated to a different box if possible.
# ------------------------------------------------------------------------ #
  MaxSuspendTime   = 60 * $(MINUTE)
  VACATE    : $(StateTimer) > $(MaxSuspendTime)
  KILL      : F

# ------------------------------------------------------------------------ #
# If you set START_DAEMONS to False loadl can never start on this machine.
# For example you may want to stop loadl for a couple days for maintenance
# and make sure no procedure automatically restarts it.
# ------------------------------------------------------------------------ #
  START_DAEMONS = True

# ------------------------------------------------------------------------ #
# Set the maximum size each of the logs can reach before wrapping.
# ------------------------------------------------------------------------ #
  MAX_SCHEDD_LOG    = 128000
  MAX_COLLECTOR_LOG = 128000
  MAX_STARTD_LOG    = 128000
  MAX_SHADOW_LOG    = 128000
  MAX_KBDD_LOG      = 128000
```

Batch Configuration

The following configuration file defines dedicated batch machines. Notice, however, that jobs in the `lp` class will suspend when a machine becomes too busy. So in this sense, the machines are not fully dedicated.

```
# ------------------------------------------------------------------------ #
# Description: LoadL_config.local for Large Batch Servers ( 580 - 590 Class )
# ------------------------------------------------------------------------ #
# Need 3x Real Memory To Paging Space ( minimum ) For Worst Case Of One
# Suspended and Two Foreground Running Job.
#    *) High Priority Jobs (btv) Never Suspend.
#    *) Job Suspension (lp) Based on LoadAvg Only.
#    *) Real Memory >= 192meg.
# ------------------------------------------------------------------------ #

# ------------------------------------------------------------------------ #
# Class defines the permissable classes, MAX_STARTERS defines the max
# total jobs to be permitted.
# ------------------------------------------------------------------------ #
  Class        = { "btv" "lp" }
  MAX_STARTERS = 2

# ------------------------------------------------------------------------ #
# The next definitions are used in the expressions below to regulate the
# conditions under which jobs get started, suspended, and evicted.
#
#     All times are specified in units of seconds.
# ------------------------------------------------------------------------ #
  BackgroundLoad   = 0.5
  HighLoad         = 1.6
  StartIdleTime    = 900
  ContinueIdleTime = 900

# ------------------------------------------------------------------------ #
# LoadAvg is an internal variable whose value is the (Berkeley) load average
# of the machine.
#
#    CPU_Idle - No LoadL job running, or One job just finishing.
#    CPU_Busy - One LoadL job running, second job ( Foreground or Batch )
#               starting up.
#    CPU_Max  - Two LoadL jobs running.
# ------------------------------------------------------------------------ #
  CPU_Idle = (LoadAvg <= $(BackgroundLoad))
  CPU_Busy = (LoadAvg >= $(HighLoad))
```

```
# ------------------------------------------------------------------------ #
# This defines a boolean "KeyboardBusy" whose value is TRUE if the keyboard
# or mouse has been used since loadl last checked.  Thus if POLLING_FREQUENCY
# is 5 seconds, KeyboardBusy is TRUE if anybody has used the kbd or mouse in
# the last 5 seconds.
# ------------------------------------------------------------------------ #
  KeyboardBusy = KeyboardIdle < $(POLLING_FREQUENCY)

# ------------------------------------------------------------------------ #
# This statement indicates when a job should be started on this machine
# ------------------------------------------------------------------------ #
  HP        = ( (Class == "btv") )
  LP        = ( (Class == "lp") && $(CPU_Idle) )

  START     : ( $(HP) || $(LP) )

# ------------------------------------------------------------------------ #
# The SUSPEND statement here says that a "lp" job should be suspended but not
# killed if a high priority job starts up or a foreground job causes the
# Loadavg to be greater than CPU_Busy ( 1.6 ).
# ------------------------------------------------------------------------ #
  SUSPEND   : (Class == "lp") && $(CPU_Busy)

# ------------------------------------------------------------------------ #
# This CONTINUE statement indicates that a suspended job should be continued
# if the cpu goes idle and the keyboard/mouse has not been used for the last
# 15 minutes.
# ------------------------------------------------------------------------ #
  CONTINUE  : $(CPU_Idle) && KeyboardIdle > $(ContinueIdleTime)

# ------------------------------------------------------------------------ #
# Jobs in the SUSPEND state are never killed, after 60 minutes they are
# relocated to a different box if possible.
# ------------------------------------------------------------------------ #
  MaxSuspendTime   = 60 * $(MINUTE)
  VACATE    : $(StateTimer) > $(MaxSuspendTime)
  KILL      : F

# ------------------------------------------------------------------------ #
# If you set START_DAEMONS to False loadl can never start on this machine.
# For example you may want to stop loadl for a couple days for maintenance
# and make sure no procedure automatically restarts it.
# ------------------------------------------------------------------------ #
  START_DAEMONS = True

# ------------------------------------------------------------------------ #
# Set the maximum size each of the logs can reach before wrapping.
# ------------------------------------------------------------------------ #
  MAX_SCHEDD_LOG    = 128000
  MAX_COLLECTOR_LOG = 128000
  MAX_STARTD_LOG    = 128000
  MAX_SHADOW_LOG    = 128000
  MAX_KBDD_LOG      = 128000
```

Configuration for a Machine That Schedules (But Doesn't Run) Jobs

The following statements define a machine that schedules jobs but does not run jobs. Notice that the schedd daemon is never forced to *not* run.

```
#
# This loadl local configuration file is set up to make a machine a
# submitter only.
#
# No jobs are allowed to run on this system.
#
MAX_STARTERS            = 0

START                   : F
#
# If you set START_DAEMONS to False loadl can never start on this machine.
# For example you may want to stop loadl for a couple days for maintenance
# and make sure no procedure automatically restarts it.
#
START_DAEMONS           = True
```

Appendix C. Sample Files

This appendix contains sample configuration and administration files. These files, as well as other sample files, are located in the directory called **/usr/lpp/LoadL/nfs/samples**.

Sample Administration File

```
# ******************************************************************************
# LoadL_admin file: Remove comments and edit this file to suit your site.
# This file consists of machine, class, and user stanzas.  Each stanza
# has defaults, as specified in a "defaults:" section.
# Default stanzas are used to set specifications for fields which are
# not filled in.
# Class, user, and group stanzas are optional.  Refer to LoadL_admin man
# page and the LoadLeveler Administration and Installation Guide for
# detailed information about keywords and associated values.  Also see
# LoadL_admin.1 in the  loadl/samples directory for sample stanzas.
################################################################################
# DEFAULTS FOR MACHINE, CLASS, USER, AND GROUP STANZAS:
# Remove initial # (comment), and edit to suit.
#
default: type = machine
#        central_manager = false  # default not central manager
#        schedd_host = false      # default not a public scheduler
#        submit_only = false      # default not a submit-only machine
#        speed = 1                # default machine speed
#        cpu_speed_scale = false  # scale cpu limits by speed
default: type = class             # default class stanza
#        priority = 0             # default ClassSysprio
#        max_processors = -1      # default max processors for class (no limit)

default: type = user              # default user stanza
#        priority = 0             # default UserSysprio
         default_class = No_Class # default class = No_Class (not optional)
         default_group = No_Group # default group = No_Group (not optional)
#        maxjobs = -1             # default maximum jobs user is allowed
                                  # to run simultaneously (no limit)
#        maxqueued = -1           # default maximum jobs user is allowed
                                  # on system queue (no limit).  does not
                                  # limit jobs submitted.

default: type = group             # default group stanza
#        priority = 0             # default GroupSysprio
#        maxjobs = -1             # default maximum jobs group is allowed
                                  # to run simultaneously (no limit)
#        maxqueued = -1           # default maximum jobs group is allowed
                                  # on system queue (no limit).  does not
                                  # limit jobs submitted.
```

© Copyright IBM Corp. 1993, 1996

```
##############################################################################
# MACHINE STANZAS:
# These are the machine stanzas; the first machine is defined as
# the central manager.  mach1:, mach2:, etc. are machine name labels -
# revise these placeholder labels with the names of the machines in the
# pool, and specify any schedd_host and submit_only keywords and values
# (true or false), if required.
##############################################################################
CENTRALMANAGER: type = machine        central_manager = true
#
# mach1:        type = machine
#               schedd_host = true
#
# mach2:        type = machine
#               schedd_host = true
#
# mach3:        type = machine
#               schedd_host = true
#               cpu_speed_scale = true
#               speed = 10
#
# mach4:        type = machine
#
# mach5:        type = machine
#
# mach6:        type = machine
#
# mach7:        type = machine
#               submit_only = true
#
# mach8:        type = machine
#               submit_only = true
##############################################################################
# CLASS STANZAS: (optional)
# These are sample class stanzas; small, medium, large, and nqs are sample
# labels for job classes - revise these labels and specify attributes
# to each class.
##############################################################################
#small:         type = class             # class for small jobs
#               priority = 100           # ClassSysprio
#               include_users=<userlist> # only these users can submit
#                                          jobs of this class
#               exclude_users=<userlist> # only these cannot submit
#               include_groups=<grouplist>
#               exclude_groups=<grouplist>
#               admin = <userlist>
#               nice = <nice value>
#               cpu_limit = 00:15:00     # 15 minute run time limit
#               data_limit = <size>
#               core_limit = <size>
#               file_limit = <size>
#               stack_limit = <size>
#               rss_limit = <size>
#
#medium:        type = class             # class for medium jobs
#               priority = 60            # ClassSysprio
#               cpu_limit= 02:00:00      # 2 hour run time limit
```

```
#large:         type = class             # class for large jobs
#               priority = 20            # ClassSysprio
#               cpu_limit = 24:00:00     # 24 hour run time limit
#               nice = -10               # Set nice value
#
#verylong:      type = class
#               class_comment = "verylong queue"
#               nice = 19
#               priority = 0
#               cpu_limit = 3000:00:00
#               job_cpu_limit = 3100:00:00
#
#parallel:      type = class
#               class_comment = "restricted access"
#               max_processors = 8
#               include_users = tbel ghtc3 hrrcr bjac3 japost roethl
#               priority = 0
#               cpu_limit = unlimited
#               job_cpu_limit = unlimited
#
#nqs:           type = class            # class for nqs jobs
#               NQS_class = true        # will be routed to NQS
#               NQS_submit = nqs_pipe_q_name # name of pipe queue
#               NQS_query = q_name1@host1 q_name2@host2 ... # names of q ueues
##############################################################################
# GROUP STANZAS: (optional)
# These are sample group stanzas; group1, group2 are sample labels
# for groups - revise these labels and specify attributes to each group.
##############################################################################
#group1:        type = group
#               priority = 80
#               maxjobs = 40
#               maxqueued = 80
#               admin = user1 user2
#               max_processors = 8
#
#group2:        type = group
#               priority = 50
#               maxjobs = 20
#               maxqueued = 40
##############################################################################
# USER STANZAS: (optional, default user stanza not optional)
# These are sample user stanzas; user1, user2, user3 are sample labels
# for users - revise these labels and specify attributes to each user.
##############################################################################
# user1:        type = user
#               priority = 80
#               default_class = small
#               default_group = group1
#               maxjobs = 20
#               maxqueued = 40
#
# user2:        type = user
#               priority = 50
#               default_class = medium long verylong
#               default_group = Unix_Group
#               maxjobs = 10
#               maxqueued = 20
#
# user3:        type = user
#               priority = 10
#               maxjobs = 5
#               maxqueued = 10
##############################################################################
```

Sample Configuration File

```
#
# Machine Description
#
ARCH = R6000
#
#  Specify LoadLeveler Administrators here:
#
LOADL_ADMIN = loadl
#
# Default to starting LoadLeveler daemons when requested
#
START_DAEMONS = TRUE
#
# Machine authentication
#
# If TRUE, only connections from machines in the ADMIN_LIST are accepted.
# If FALSE, connections from any machine are accepted.  Default if not
# specified is FALSE.
#
MACHINE_AUTHENTICATE = FALSE
#
# Specify which daemons run on each node
#
SCHEDD_RUNS_HERE = True
STARTD_RUNS_HERE = True
#
# Specify information for backup central manager
#
# CENTRAL_MANAGER_HEARTBEAT_INTERVAL = 300
# CENTRAL_MANAGER_TIMEOUT = 6
#
# Specify pathnames
#
RELEASEDIR      = $(tilde)
LOCAL_CONFIG = $(tilde)/LoadL_config.local
ADMIN_FILE      = $(tilde)/LoadL_admin
LOG         = $(tilde)/log
SPOOL         = $(tilde)/spool
EXECUTE  = $(tilde)/execute
HISTORY  = $(SPOOL)/history
BIN  = $(RELEASEDIR)/bin
LIB  = $(RELEASEDIR)/lib
#
# Specify port numbers
#
MASTER_STREAM_PORT     = 9616
NEGOTIATOR_STREAM_PORT = 9614
SCHEDD_STREAM_PORT     = 9605
STARTD_STREAM_PORT     = 9611
COLLECTOR_DGRAM_PORT   = 9613
STARTD_DGRAM_PORT      = 9615
MASTER_DGRAM_PORT      = 9617
```

```
#
# Turn on/off the internal LoadLeveler scheduling algorithm
# Default is on
#
SCHEDULER_API = NO
#SCHEDULER_API = YES
#
# Specify accounting controls
#
ACCT    = A_OFF
ACCT_VALIDATION  = $(BIN)/llacctval
GLOBAL_HISTORY  = $(SPOOL)
#
# Specify prolog and epilog path names
#
# JOB_PROLOG =
# JOB_EPILOG =
# JOB_USER_PROLOG =
# JOB_USER_EPILOG =
#
# Refresh AFS token program.
#
# AFS_GETNEWTOKEN =
#
# Customized mail delivery program.
#
# MAIL =
#
# Customized submit (job command file) filter program.
#
# SUBMIT_FILTER =
#
# Specify checkpointing intervals
#
MIN_CKPT_INTERVAL    = 900
MAX_CKPT_INTERVAL    = 7200
# LoadL_KeyboardD Macros
#
KBDD                 = $(BIN)/LoadL_kbdd
KBDD_LOG             = $(LOG)/KbdLog
MAX_KBDD_LOG         = 64000
KBDD_DEBUG           =
#
# Specify whether to start the keyboard daemon
#
#if HAS_X
X_RUNS_HERE   = True
#else
X_RUNS_HERE   = False
#endif
```

```
#
# Specify whether to use X server XGetIdleTime() protocol extension
#
#if HAS_X_EXTENSIONS
USE_X_IDLE_EXTENSION = True
#else
USE_X_IDLE_EXTENSION = False
#endif
#
#  LoadL_StartD Macros
#
STARTD   = $(BIN)/LoadL_startd
STARTD_LOG  = $(LOG)/StartLog
MAX_STARTD_LOG  = 64000
STARTD_DEBUG  =
POLLING_FREQUENCY = 5
POLLS_PER_UPDATE = 24
JOB_LIMIT_POLICY = 120
JOB_ACCT_Q_POLICY = 300
#
#ifdef KbdDeviceName
KBD_DEVICE              = KbdDeviceName
#endif
#ifdef MouseDeviceName
MOUSE_DEVICE            = MouseDeviceName
#endif
#
#  LoadL_SchedD Macros
#
SCHEDD   = $(BIN)/LoadL_schedd
SCHEDD_LOG  = $(LOG)/SchedLog
MAX_SCHEDD_LOG  = 64000
SCHEDD_DEBUG  =
SCHEDD_INTERVAL  = 120
#
CLIENT_TIMEOUT  = 30
#
# Negotiator Macros
#
NEGOTIATOR   = $(BIN)/LoadL_negotiator
NEGOTIATOR_DEBUG  =
NEGOTIATOR_LOG   = $(LOG)/NegotiatorLog
MAX_NEGOTIATOR_LOG  = 64000
NEGOTIATOR_INTERVAL  = 60
MACHINE_UPDATE_INTERVAL         = 300
NEGOTIATOR_PARALLEL_DEFER = 300
NEGOTIATOR_PARALLEL_HOLD = 300
NEGOTIATOR_REDRIVE_PENDING = 90
NEGOTIATOR_RESCAN_QUEUE  = 90
NEGOTIATOR_REMOVE_COMPLETED = 0
```

```
#
# Sets the interval between recalculation of the SYSPRIO values
# for all the jobs in the queue
#
NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL = 0
#
# Starter Macros
#
STARTER   = $(BIN)/LoadL_starter
STARTER_DEBUG  =
STARTER_LOG             = $(LOG)/StarterLog
MAX_STARTER_LOG         = 64000
#
# LoadL_Master Macros
#
MASTER   = $(BIN)/LoadL_master
MASTER_LOG  = $(LOG)/MasterLog
MASTER_DEBUG  =
MAX_MASTER_LOG  = 64000
RESTARTS_PER_HOUR = 12
PUBLISH_OBITUARIES = TRUE
OBITUARY_LOG_LENGTH = 25
#
# Specify whether log files are truncated when opened
#
TRUNC_MASTER_LOG_ON_OPEN     = False
TRUNC_STARTD_LOG_ON_OPEN     = False
TRUNC_SCHEDD_LOG_ON_OPEN     = False
TRUNC_KBDD_LOG_ON_OPEN       = False
TRUNC_STARTER_LOG_ON_OPEN    = False
TRUNC_NEGOTIATOR_LOG_ON_OPEN = False
#
#       NQS Directory
#
# For users of NQS resources:
# Specify the directory containing qsub, qstat, qdel
#
# NQS_DIR   = /usr/bin
#
# Specify ISS metric keywords
#
# CUSTOM_METRIC  =
# CUSTOM_METRIC_COMMAND =
#
# Machine control expressions and macros
#
OpSys : "$(OPSYS)"
Arch   : "$(ARCH)"
Machine : "$(HOST).$(DOMAIN)"
#
# Expressions used to control starting and stopping of foreign jobs
#
MINUTE  = 60
HOUR  = (60 * $(MINUTE))
StateTimer = (CurrentTime - EnteredCurrentState)
BackgroundLoad  = 0.7
HighLoad  = 1.5
StartIdleTime  = 15 * $(MINUTE)
ContinueIdleTime = 5 * $(MINUTE)
MaxSuspendTime  = 10 * $(MINUTE)
MaxVacateTime  = 10 * $(MINUTE)
```

```
#
KeyboardBusy = KeyboardIdle < $(POLLING_FREQUENCY)
CPU_Idle = LoadAvg <= $(BackgroundLoad)
CPU_Busy = LoadAvg >= $(HighLoad)
#
#  Refer to LoadL_config man page for an explanation of these control
#  expressions
#
# START  : $(CPU_Idle) && KeyboardIdle > $(StartIdleTime)
# SUSPEND : $(CPU_Busy) || $(KeyboardBusy)
# CONTINUE : $(CPU_Idle) && KeyboardIdle > $(ContinueIdleTime)
# VACATE : $(StateTimer) > $(MaxSuspendTime)
# KILL  : $(StateTimer) > $(MaxVacateTime)
#
START  : T
SUSPEND  : F
CONTINUE : T
VACATE  : F
KILL  : F
#
# Expressions used to prioritize job queue
# The following expression is an example.
#
#SYSPRIO: (ClassSysprio * 100) + (UserSysprio * 10) + (GroupSysprio * 1)- (QDate)
#
# The following (default) expression for SYSPRIO creates a FIFO job queue.
#
SYSPRIO: 0 - (QDate)
#
# Expressions used to prioritize machines
#
# The following example orders machines by the load average
# normalized for machine speed:
#
#MACHPRIO: 0 - (1000 * (LoadAvg / (Cpus * Speed)))
#
# The following (default) expression for MACHPRIO orders
# machines by load average.
#
MACHPRIO: 0 - (LoadAvg)
#
# The MAX_JOB_REJECT value determines how many times a job can be
# rejected before it is canceled or put on hold.  The default is -1,
# which indicates no limit to the number of times a job can be rejected.
#
MAX_JOB_REJECT = -1
#
# When ACTION_ON_MAX_REJECT is HOLD, jobs will be put on user hold
# when the number of rejects reaches the MAX_JOB_REJECT value.  When
# ACTION_ON_MAX_REJECT is CANCEL, jobs will be canceled when the
# number of rejects reaches the MAX_JOB_REJECT value. The default
# value is HOLD.
#
ACTION_ON_MAX_REJECT = HOLD
```

How the Configuration File and the Administration File Work Together

```
# This configuration file defines classes.
# Cjour (day class): cpu_limit=30,
#       data_limit=unlimited.
# Dnuit (night class): cpu_limit=16,
#       data_limit=64mb.
# Fnuit (night class): cpu_limit=16,
#       data_limit=unlimited.
# Fnuit suspended during the day.
# Dnuit is suspended if LoadAverage > 3
# and the job is continued if the LoadAverage <= 1.5
START_DAEMONS = TRUE
X_RUNS_HERE = FALSE
# Definition of classes
Class = { "Cjour" "Dnuit" "Fnuit" }
# Define the maximum number of LoadLeveler
# jobs that will run on this machine.
MAX_STARTERS = 3
HighLoadStD = 1.5  # HighLoad Start for Fnuit
HighLoadSuD = 3
MIN_CKPT_INTERVAL = 900     # Specify how often
MAX_CKPT_INTERVAL = 7200    # checkpointing occurs
# Define the macros Jour and Nuit
Nuit = ((tm_hour >= 17) || (tm_hour <=9) || \
       (tm_wday == 6) || (tm_wday == 0))
Jour = ((tm_wday > 0) && (tm_wday < 6) \
       && (tm_hour >=9) && \
       (tm_hour <=17))
# A Cjour job will start during
# the day.  A Dnuit job will start during
# the night.  A Fnuit job will start
# during the night if the load average
# is less than or equal to 1.5.
#
START: (((Class == "Cjour") && $(Jour) || \
       ((Class == "Dnuit") && $(Nuit) || \
       ((Class == "Fnuit") && $(Nuit) && \
       (LoadAvg <=$(HighLoadStD))))
# The Dnuit job will get suspended
# during the day if the LoadAvg is
# greater than HighLoadSuD.  The Fnuit job
# will be suspended during the day.
SUSPEND: (((Class == "Dnuit") && $(Jour) && \
       ((LoadAvg >= $(HighLoadSuD))) || \
       ((Class == "Fnuit") && $(Jour)))
# The Dnuit job may continue during the day if
# the LoadAvg is less than HighLoadStD.
CONTINUE: (((Class == "Dnuit") && $(Jour) && \
       ((LoadAvg <= $(HighLoadStD))) || \
       ((Class == "Fnuit") && $(Nuit)))
VACATE: F
KILL: F
# The priorities of the classes
SYSPRIO: 0-(Qdate) + (ClassSysprio) * 10)
```

```
Administration File
default: type=machine
         central_manager=false
         schedd_host=true
default: type=class
         priority=50
         cpu_limit=unlimited
         data_limit=unlimited
 default: type=user
      priority=50
         default_class=A_jour
         max_jobs=2
# machine stanzas
fturbo1: type=machine
         central_manager=true
fturbo2: type=machine
# class stanzas
Cjour: type=class
       priority=50
       cpu_limit=00:30:00
#
Dnuit: type=class
 priority=50
       cpu_limit=16:00:00
 data_limit=64mb
 Fnuit: type=class
    priority=50
      cpu_limit=16:00:00
# user stanzas
user1: type=user
       priority=50
       default_class=No_class
       max_jobs=20
user2: type=user
       priority=50
       default_class=No_class
    max_jobs=10
```

Glossary

This section contains some of the terms that are commonly used in the LoadLeveler books and in this book in particular.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the American National Standard *Vocabulary for Information Processing* (Copyright 1970 by American National Standards Institute, Incorporated), which was prepared by Subcommittee X3K5 on Terminology and Glossary of the American National Standards Committee X3. ANSI definitions are preceded by an asterisk (*).

Other definitions in this glossary are taken from *IBM Vocabulary for Data Processing, Telecommunications, and Office Systems* (GC20-1699), *IBM DATABASE 2 Application Programming Guide for TSO Users* (SC26-4081), and *Internetworking With TCP/IP, Principles, Protocols, and Architecture*, by Douglas Comer, Copyright 1988 by Prentice Hall, Incorporated

© Copyright IBM Corp. 1993, 1996

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.

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.

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 protocal 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.

default. An alternative value, attribute, or option that is assumed when none has been specified.

H

host. A computer connected to a network, and providing an access method to that network. A host provides end-user services.

M

menu. A display of a list of available functions for selection by the user.

Motif. The UNIX industry's standard user interface, originally developed by the Open Systems Foundation. Motif is based on the X-Window system and is a Presentation Manager look-alike. Motif is available for all IBM AIX workstations.

N

network. An interconnected group of nodes, lines, and terminals. A network provides the ability to transmit data to and receive data from other systems and users.

NFS. Network File System.

node. In a network, the point where one or more functional units interconnect transmission lines. A computer location defined in a network.

NQS. Network Queueing System.

P

parameter. *(1) A variable that is given a constant value for a specified application and that may denote the application. *(2) An item in a menu for which the operator specifies a value or for which the system provides a value when the menu is interpreted. *(3) A name in a procedure that is used to refer to an argument that is passed to the procedure. *(4) A particular piece of information that a system or application program needs to process a request.

process. *(1) A unique, finite course of events defined by its purpose or by its effect, achieved under defined conditions. *(2) Any operation or combination of operations on data. *(3) A function being performed or waiting to be performed. *(4) A program in operation. For example, a daemon is a system process that is always running on the system.

S

server. (1) A function that provides services for users. A machine may run client and server processes at the same time. (2) A machine that provides resources to the network. It provides a network service, such as disk storage and file transfer, or a program that uses such a service.

shell. The shell is the primary user interface for the UNIX operating system. It serves as command language interpreter, programming language, and allows foreground and background processing. There are three different implementations of the shell concept: Bourne, C and Korn.

stream. An internet standard transport level protocol that provides the reliable, full duplex, stream service on which many application protocols depend. TCP allows a process on one machine to send a stream of data to a process on another. It is connection-oriented in the sense that before transmitting data, participants must establish a connection. Software implementing TCP usually resides in the operating system and uses the IP protocol to transmit information across the Internet. It is possible to terminate (shut down) one direction of flow across a TCP connection, leaving a one-way (simplex) connection. The Internet protocol suite is often referred to as TCP/IP because TCP is one of the two most fundamental protocols.

System Administrator. The user who is responsible for setting up, modifying, and maintaining LoadLeveler.

U

user. Anyone who is using LoadLeveler.

W

working directory. All files without a fully qualified path name are relative to this directory.

workstation. *(1) A configuration of input/output equipment at which an operator works. *(2) A terminal or microcomputer, usually one that is connected to a mainframe or to a network, at which a user can perform applications.

Index

Special Characters

/etc/LoadL.cfg file 27, 96
/etc/services file 111

A

account 83
account_no 40
accounting
  API 257
  collecting data 163
  in job command file 40
  llacctmrg command 186
  llsummary command 221
  reports 165
ACCT 106
ACCT_VALIDATION 107
ACTION_ON_MAX_REJECT 119
Adapter
  requirement in job command file 51, 59
admin keyword 86, 94
ADMIN_FILE 109
administering LoadLeveler
  administration file 73
  LoadL_admin file 77
  overview 73
  stanzas 78
administration file
  keywords 127
  sample 313
  structure and syntax 77
administrators 97
AFS authentication 119
AFS authentication user exit 280
AFS token handling 280
AFS_GETNEWTOKEN 119
AGENT_LOG 142, 143
AGENT_TRACE 142, 143
AIX Error Log 169
  sample formatted output 169
AIX Error Notification Facility 170
alias 79
alternate central manager 107
application programming interfaces
  accounting 257
  checkpointing 265
  monitor 267
  parallel jobs 268
  scheduling 273
Arch
  requirement in job command file 51
Arch *(continued)*
  reserved macro 123
ARCH configuration file keyword 100
arguments 40
available classes 123

B

BIN 109
building jobs
  using a job command file 23
  using the GUI 232

C

cancelling jobs
  using llcancel 33
  using the GUI 241
central manager 5, 34, 79, 107, 243, 290
central_manager keyword 80
CENTRAL_MANAGER_HEARTBEAT_INTERVAL 107
CENTRAL_MANAGER_TIMEOUT 107
changing job priority
  example 33
  using llprio 205
  using the GUI 241
checkpointing
  API 265
  ckpt function 266
  system_initiated 29, 40
  system_initiated checkpointing 113
  user_initiated 29, 40
  user_initiated checkpointing 113
choice button 235
ckpt 266
class
  keyword
    for ISS 143
    for machines 100
    in job command file 41
  multiple job classes 296
  querying class information 190
  stanzas
    examples 92
    format 85
class stanza keywords
  admin 86
  class_comment 86
  core_limit 91
  cpu_limit 91
  data_limit 91
  exclude_groups 86

© Copyright IBM Corp. 1993, 1996

class stanza keywords *(continued)*
exclude_users 86
file_limit 91
include_groups 86
include_users 87
master_node_requirement 87
max_processors 87
maxjobs 87
nice value keyword 87
NQS_class 87
NQS_query 87
NQS_submit 87
priority 88
rss_limit 92
stack_limit 92
wall_clock_limit 92
class_comment 86
ClassSysprio 103
CLIENT_TIMEOUT 112
cluster
definition 3
querying multiple clusters 27
submitting jobs to multiple clusters 27
COLLECTOR_DGRAM_PORT 112
commands 185
comment 41
common name space 73
completed job state 18
configuration file
keywords 129
sample 316
structure and syntax 97
configuring LoadLeveler
global configuration file 96
LoadLeveler user ID 96
local configuration file 96
constants 123
CONTINUE expression 98
control functions 98
copy 90
core_limit 41, 86, 91
cpu_limit 41, 86, 91
cpu_speed_scale 79, 80
Cpus
reserved macro 123
using with MACHPRIO 105
CurrentTime 123
CUSTOM 146
CUSTOM_METRIC 101
CUSTOM_METRIC_COMMAND 101
customizing 97
CustomMetric 105

D

daemons
kbdd 18
master 13
negotiator 17
schedd 14
startd 15
data_limit 42, 86, 91
debugging
controlling output 110
error logging facility 169
default_class 83, 84
default_group 83, 84
deferred job state 18
dependency 42, 296
diagnosing problems 285
Disk
requirement in job command file 50
reserved macro 124
using with MACHPRIO 105
displaying job status
using the command llq 32
using the GUI 239
displaying machine status
public submit machines 243
using llstatus 33
using the GUI 242
domain (domain name) 124

E

editing jobs 27, 238
EnteredCurrentState 124
environment 43
epilog programs 112, 259
error keyword 44
error log events
classification 169
error logging
definition 169
DETECTING MODULE 169
getting notified 170
overview 169
switch error log reports 170
exclude_groups 86
exclude_users 86, 94
executable 25, 37, 44
EXECUTE 109
executing machine 5
execution window for jobs 294
exit status 48, 219
expressions
CONTINUE 98
KILL 98
START 98

expressions *(continued)*
SUSPEND 98
VACATE 98
extended accounting report 165
external scheduler 273

F

favor jobs 251
llfavorjob command 197
favor users 250
llfavoruser command 199
feature
configuration file keyword 101
requirement in job command file 50
file_limit 45, 86, 91
filtering a job script 281

G

GetHistory 166, 258
GLOBAL_HISTORY 107, 165
graphical user interface
customizing 246
overview 229
starting 229
tasks 232
Xloadl 229, 246
Xloadl_so 229, 246
group 45
default 84
LoadLeveler 94
UNIX 84, 94
group stanza keywords
admin 94
exclude_users 94
include_users 94
max_processors 95
maxidle 94
maxjobs 94
maxqueued 95
priority 95
group stanzas
examples 95
format 93
GroupQueuedJobs 103
GroupRunningJobs 103
GroupSysprio 103
GroupTotalJobs 103
GUI (see graphical user interface) 250

H

HEARTBEAT_INTERVAL 144
HEARTBEATS_PER_UPDATE 144
help
calling IBM 297
in the GUI 231
high performance switch error reports 170
hints for running LoadLeveler 293
HISTORY 109
history file 297
hold 45
holding jobs
using llhold 28, 33
using the GUI 241
host 124
hostname 124

I

idle job state 18
image_size 45
include_groups 86
include_users 86, 87, 94
initialdir 46
input 46
interactive sessions 137
ISS
AGENT_LOG 142, 143
AGENT_TRACE 142, 143
CLASS 143
configuring
CUSTOM 146
ending an interactive session 155
establishing an interactive session 155
HEARTBEAT_INTERVAL 144
HEARTBEATS_PER_UPDATE 144
installing 138
ISS custom configuration 140
ISS round-robin configuration 140
Iss_config file
ISS_PORT 142, 145
LIMIT 145
METRIC 145
MONITOR_TRACE 142, 147
NAME 148
POLICY 148
ROUNDROBIN 147
SERVERS 149
starting ISS 155
stopping ISS 156
troubleshooting 156
ISS_PORT 142, 145

J

job
accounting 163
building a job command file 23, 232
cancelling 28, 241

job *(continued)*
class name 41
diagnosing problems with 285, 286, 288
editing 27, 238
exit status 48, 219
filter 281
holding 28, 241
parallel 58, 286
priority 29, 241
releasing a hold 241
running 293
samples 32
serial 23
states 18
status 27, 207, 209, 239
submit-only 288
submitting 23, 26, 239
job command file
example 24, 35, 36, 37
syntax 24
JOB_ACCT_Q_POLICY 163
job_cpu_limit 46, 86
JOB_EPILOG 113, 259
JOB_LIMIT_POLICY 163
job_name 47
JOB_PROLOG 113, 259
job_type 47, 58
JOB_USER_EPILOG 113, 259
JOB_USER_PROLOG 113, 259

K

kbdd daemon 18
KeyboardIdle 124
keywords
administration file 78—126, 127
class stanza 86
configuration file 97—121, 122, 129, 134
reserved macros 122, 134
user-defined 121, 134
group stanza 94
job command file 40—54
machine stanza 79
reserved 122—126
user stanza 83
user-defined 121—122
KILL expression 99

L

LIB 109
libckpt.a 40, 115
libllapi.a 258, 266, 268, 274
libllapi.h 258, 266, 268, 274
libload.a 40, 115
LIMIT 145
limits 91
llacctmrg 186
llcancel 188
llclass 190
llctl 193
llfavorjob 197
llfavoruser 199
llhold 200
llinit 203
llprio 205
llq 207
llstatus 214
llsubmit 219
llsummary 221
load average 297
LoadAvg
reserved macro 124
using with MACHPRIO 105
loadl user ID 96
LoadL_admin file 77, 299, 313
LOADL_ADMIN keyword 97
LoadL_config file 96, 316
LoadL_config.local file 96, 300, 304
LOADL_PROCESSOR_LIST 66
LoadLeveler user ID 96
LOCAL_CONFIG 109
LOG 109
log files 109
KBDD_LOG 110
MASTER_LOG 110
MAX_KBDD_LOG 110
MAX_NEGOTIATOR_LOG 110
MAX_STARTER_LOG 110
NEGOTIATOR_LOG 110
SCHEDD_LOG 110
STARTD_LOG 110
STARTER_LOG 110

M

Machine
requirement in job command file 50
reserved macro 124
machine stanza keywords
alias 79
central_manager 80
cpu_speed_scale 80
master_node_exclusive 81
max_jobs_scheduled 81
name_server 81
pvm_root 81
rm_host 81
schedd_host 82
speed 82
submit_only 82

machine stanzas
examples 83
format 78
machine status 214
MACHINE_AUTHENTICATE 101
MACHINE_UPDATE_INTERVAL 119
macros 122
Arch 123
AvailableClasses 123
Cpus 123
CurrentTime 123
Disk 124
domain (domain name) 124
EnteredCurrentState 124
host or hostname 124
in a job command file 53
KeyboardIdle 124
LoadAvg 124
Machine 124
Memory 124
OpSys 124
QDate 124
state 124
tilde 124
UserPrio 125
VirtualMemory 125
MAIL keyword 281
mail program 281
master daemon 13
master node 57
MASTER_DGRAM_PORT 112
master_node_exclusive 81
master_node_requirement 87
MASTER_STREAM_PORT 112
MasterMachPriority 105
MAX_CKPT_INTERVAL 115
MAX_JOB_REJECT 119
max_jobs_scheduled 79, 81
max_processors 47, 58, 83, 85, 86, 94, 95
maxidle 83, 84, 94, 295
maxjobs 83, 84, 86, 87, 94, 295
maxqueued 83, 84, 94, 95, 295
Memory
requirement in job command file 50
reserved macro 124
using with MACHPRIO 105
menu bar 229
messages 245
METRIC 145
migration considerations xii
MIN_CKPT_INTERVAL 115
min_processors 47, 58
monitor API 267
MONITOR_TRACE 142, 147
MOUSE_DEVICE 119

N

NAME 148
name_server 79, 81
named.boot 151
named.data 152
negotiator daemon
definition 6
description 17
job states 18
keywords 119
NEGOTIATOR_INTERVAL 119
NEGOTIATOR_LOADAVG_INCREMENT 119
NEGOTIATOR_PARALLEL_DEFER 56
NEGOTIATOR_PARALLEL_HOLD 56
NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL 120
NEGOTIATOR_REJECT_DEFER 120
NEGOTIATOR_REMOVE_COMPLETED 120
NEGOTIATOR_RESCAN_QUEUE 120
NEGOTIATOR_STREAM_PORT 112
nice value 86, 87
notification 48
notify_user 48
NotQueued job state 18
NQS
options 179
routing jobs to NQS machines 177
NQS jobs
cancelling 181
obtaining status 180
submitting 179
NQS scripts 181
NQS_class 86, 178
NQS_DIR 109, 178
NQS_query 86, 178
NQS_submit 86, 178

O

OBITUARY_LOG_LENGTH 120
operators 123
OpSys
requirement in job command file 51
reserved macro 124
output 48, 296

P

parallel jobs
API 268
checklist 287
job command file examples 60
master node 57
migration considerations 68
overview 55
scheduling considerations 56
status 67

parallel_path 49, 59
pending job state 18, 289
performance 73
POE
  job command file 60
POLICY 148
POLLING_FREQUENCY 120
POLLS_PER_UPDATE 120
Pool
  requirement in job command file 51, 60
port numbers 111
preferences 49
priority 29, 83, 88, 94, 95, 205
proc 14
productivity aids 293
prolog programs 112, 259
public scheduling machine 5
public scheduling machines 31, 34
PUBLISH_OBITUARIES 121
pull-down menus 230
PVM
  job command file 62
  planning considerations 56
  restrictions 58
pvm_root 79, 81
PVMe
  job command file 61
  restrictions 58

Q

QDate 103, 124
query a job 240
  llq command 207
  using the GUI 240
querying class information
  llclass command 190
querying multiple clusters 27
questions and answers 285
queue 49

R

reject pending job state 19
release from hold 251
RELEASEDIR 109
remove pending job state 19
requirements 50, 59
reserved macros 122, 134
restart 51
RESTARTS_PER_HOUR 121
rlim_infinity 90
rm_host 79, 81, 289
ROUNDROBIN 147
rss_limit 52, 86, 92
running jobs at a specific time of day 294

S

sample formatted output
  AIX Error Log 169
schedd daemon 14, 289
schedd_host 79, 82
SCHEDD_INTERVAL 121
SCHEDD_RUNS_HERE 102
SCHEDD_STREAM_PORT 112
SCHEDULER_API 121
scheduling
  API 273
  external 273
scheduling considerationsfor parallel jobs 56
scheduling machine 5
serial job command files 24
SERVERS 149
service numbers 111
shell 52, 235
  pool 236
short report, accounting 165
signals 269
speed 79, 82, 105, 167
SPOOL 109
  log 109
stack_limit 52, 86, 92
stanzas
  class 85
  default 78
  label 77
  machine 78
  type 77
  user 78
START expression 98
start LoadLeveler 251
START_DAEMONS 102
startd daemon 15, 289
STARTD_DGRAM_PORT 112
STARTD_RUNS_HERE 102
STARTD_STREAM_PORT 112
startdate 52
starter process 17
state 124
states of a job 18
status 214, 219
step_name 53
stop LoadLeveler 251
submit-only machine
  cancelling jobs from 28
  definition 3
  keywords 82
  master daemon interaction 13
  querying jobs from 27
  querying multiple clusters 27

submit-only machine *(continued)*
schedd daemon interaction 14
submitting jobs from 26
troubleshooting 288
types 5
SUBMIT_FILTER 281
submit_only keyword 79, 82
submitting jobs
across multiple clusters 27
using a job command file 26
using llsubmit 32, 219
using the GUI 239
support services 297
SUSPEND expression 98
switch error log 170
syshold 251
SYSPRIO 29, 103
system priority 29
system_initiated checkpointing 29, 40, 113

T

tbx adapter error reports 170
TCP/IP nameserver 159
TCP/IP service and port numbers 111
tilde 124
tm_hour 125
tm_isdst 125
tm_mday 125
tm_min 125
tm_mon 125
tm_sec 125
tm_wday 125
tm_yday 125
tm_year 125
troubleshooting 285
TRUNC_KBDD_LOG_ON_OPEN 110
TRUNC_MASTER_LOG_ON_OPEN 110
TRUNC_NEGOTIATOR_LOG_ON_OPEN 110
TRUNC_SCHEDD_LOG_ON_OPEN 110
TRUNC_STARTD_LOG_ON_OPEN 110
TRUNC_STARTER_LOG_ON_OPEN 110

U

unfavor jobs 251
unfavor users 251
Unix_group 94
user name 73
user priority 29
user stanza keywords
account 83
default_class 84
default_group 84
max_processors 85
maxidle 84
user stanza keywords *(continued)*
maxjobs 84
maxqueued 84
priority 83
user stanzas
examples 85
format 78
user-defined keywords 121, 122
user_initiated checkpointing 29, 40, 113
user_priority 53
UserPrio 104, 125
UserQueuedJobs 104
UserRunningJobs 104
UserSysprio 104
UserTotalJobs 104

V

VACATE expression 98
vacate job state 19
VirtualMemory
reserved macro 125
using with MACHPRIO 105

W

wall_clock_limit 53, 86, 92
world wide web information ix

X

X_RUNS_HERE 102
Xloadl 229, 246
Xloadl_so 229, 246