IBM LoadLeveler
User's Guide

Document Number SH26-7226-02

January 24, 1994

Susann Dewey

Information Development
Dept. 34CA 634 Bldg. 202/2
Kingston, NY
Tie Line 8-695-4121
VNET: TADJ @ KGNVMC

IBM

IBM LoadLeveler
User's Guide

SH26-7226-02

IBM

IBM LoadLeveler
User's Guide

SH26-7226-02

Note

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

First Edition, March 1994

This edition applies to Version 1 Release 1.3 of IBM LoadLeveler, (5765-145 for RISC System/6000 and 9076 Scalable POWERparallel Systems), (5765-227 for Sun), (5765-228 for SGI), (5765-287 for HP) and to all 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 below.

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

International Business Machines Corporation
Department 34CA
Neighborhood Road
Kingston, NY
12401

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. 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.. Contains licensed materials — Property of IBM. Copyright 1986, 1987, 1988, 1989, 1990, 1991 by the Condor Design Team

Contents

Notices iv
Trademarks and Service Marks iv
About This Book v
Who Should Use This Book v
How to Use This Book v
Overview of Contents v
Terms and Document Conventions vi
Related Publications vi
Ordering Additional Copies of This Book vi
Summary of New Functions vii
Parallel Job Submission Support vii
Job Accounting vii
Increased Support for Machines Running NQS vii
Submit-only Workstations vii
Additional Hardware Platforms Supported vii
Interactive Session Support viii

Chapter 1. Getting Started with LoadLeveler 1
What is LoadLeveler? 1
What Tasks Can I Accomplish Using LoadLeveler? 3
How Do I Use LoadLeveler's Graphical User Interface? 13

Chapter 2. Building and Submitting Jobs Using the Graphical User Interface 17
Starting Up the LoadLeveler Graphical User Interface 18
Task Scenario Using the Graphical User Interface 18

Chapter 3. Building and Submitting Jobs Using LoadLeveler Commands 29
Task Scenario Using the Commands 29

Chapter 4. LoadLeveler Commands 33

Appendix A. LoadLeveler Job Command File Keywords and Examples . 51
LoadLeveler Keywords 51
Job Command File Syntax 59
Job Command File Examples 60

Appendix B. Graphical User Interface Information 65
Customizing the Graphical User Interface 67

Appendix C. Checkpointing Jobs 79
Checkpointing for the RISC System/6000 80
Checkpointing for the Sun 80
Checkpointing for the SGI 81
Checkpointing for the HP 81

Glossary of Terms and Abbreviations 83

Index 87

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 or other legally protectable rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other IBM 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 the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

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

Trademarks and Service Marks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

LoadLeveler

AIX

IBM

Scalable POWERparallel Systems

9076 SP1

RISC System/6000

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies as follows:

Motif	Open Software Foundation, Incorporated
UNIX	Unix System Laboratories, Incorporated
HP	Hewlett-Packard Company
NFS	Sun Microsystems, Incorporated
Sun	Sun Microsystems, Incorporated
SGI	Silicon Graphics, Incorporated
AFS	Transarc Corporation

About This Book

This book is intended to help you use the LoadLeveler* program product to build and submit jobs and perform other job management related tasks. It primarily contains instructions to use the graphical user interface and the LoadLeveler commands to accomplish these tasks.

Who Should Use This Book

This book is designed for both end users and administrators. These readers should have some experience with graphical user interface concepts such as windows, pull-down menus, and menu bars. They should also have knowledge of AIX* or other UNIX** based operating systems.

How to Use This Book

Overview of Contents

Chapter 1, “Getting Started with LoadLeveler” on page 1 provides introductory information about LoadLeveler, its interface, and the tasks end users can perform.

Chapter 2, “Building and Submitting Jobs Using the Graphical User Interface” on page 17 provides a task scenario that shows how to use the interface.

Chapter 3, “Building and Submitting Jobs Using LoadLeveler Commands” on page 29 provides a task scenario that shows how to use the commands.

Chapter 4, “LoadLeveler Commands” on page 33 provides an alphabetical listing of the commands and their syntax.

Appendix A, “LoadLeveler Job Command File Keywords and Examples” on page 51 provides the keywords you can include in job command files and examples of job command files.

Appendix B, “Graphical User Interface Information” on page 65 provides general information on graphical user interfaces such as windows, menus, and dialog boxes.

Appendix C, “Checkpointing Jobs” on page 79 provides detailed instructions on how to checkpoint jobs for submission to RISC System/6000, 9076 SP1 machines, Sun, or SGI workstations.

A glossary and index are also included.

Terms and Document Conventions

This manual uses visual cues to help you locate and identify information quickly and easily. The following document conventions are used:

Type Style	Used for
bold	Command names, program names, file names, flag names, directory names, path names, keywords, and any text that you must type exactly as it appears.
italic	Parameters and variables. You can substitute the text for items shown in italic. For example, when you have to enter a variable called *filename* you must type the name of the appropriate file in place of *filename*. New terms are introduced in italic. Italic type is also occasionally used for emphasis.
`monospace`	Examples of input text and system prompts or responses that appear on the screen.

Wherever the term UNIX appears in this manual, AIX, the IBM* implementation of UNIX also applies.

Related Publications

IBM LoadLeveler: General Information Manual, GH26-7227

IBM LoadLeveler: Administration and Installation Planning Guide, SH26-7220

IBM LoadLeveler: Quick Reference Summary, SX23-1048

Ordering Additional Copies of This Book

To order additional copies of this book, request form number SH26-7226.

Summary of New Functions

Following is a brief description of the new functions contained in this release:

Parallel Job Submission Support

LoadLeveler now supports parallel job submission for the IBM 9076 SP1. By providing a parallel program written using the Application Programming Interface (API) of the IBM AIX Parallel Environment program product and by modifying your job command file to specify the number of parallel machines you will need, you can submit a parallel job to the IBM 9076 Scalable POWERparallel Systems (SP1).

Job Accounting

LoadLeveler provides accounting information on completed LoadLeveler jobs and stores this information in a file. This job information includes the job parameters, the amount of resources consumed by the job, as well as other job related data. Administrators can collect data on completed jobs submitted from one machine or on all completed jobs on all machines and merge this data into one central file. This feature can help keep track of the different types of jobs being processed, as well as how many resources each type of job requires for processing.

Increased Support for Machines Running NQS

In Release 1.0, LoadLeveler translated scripts containing NQS options into LoadLeveler scripts and submitted the jobs to a machine in the LoadLeveler cluster. Now with the increased support for machines running NQS, LoadLeveler can route any of your jobs (LoadLeveler jobs or non-LoadLeveler jobs) to machines outside of the LoadLeveler cluster that run NQS. This additional support allows you to use all of your machines that run NQS while at the same time taking advantage of LoadLeveler's other jobs scheduling features. In this way, you can include a Cray, for example, in your pool of resources to be managed by LoadLeveler.

Submit-only Workstations

Your installation can now take advantage of the submit-only machine feature that allows workstations that are not part of the LoadLeveler pool to submit jobs to the pool for processing. In addition, you can use these workstations to cancel and query jobs. No LoadLeveler license fee is charged for these workstations.

Additional Hardware Platforms Supported

LoadLeveler now supports the Sun workstation from Sun Microsystems, Incorporated, the SGI workstation from Silicon Graphics, Incorporated, and the HP Apollo 9000 series 700 from Hewlett-Packard Company. In addition, the IBM RISC System/6000 and the IBM 9076 Scalable POWERparallel Systems (SP1) are also supported. The addition of the Sun, SGI, and HP platforms allows you to expand your network cluster into a more heterogeneous environment. LoadLeveler also plans to support additional platforms in the future.

Interactive Session Support

Interactive Session Support (ISS) distributes login and application sessions across the nodes of an IBM Scalable POWERparallel Systems* 9076 SP1 and a cluster comprised of RISC System/6000 workstations by interfacing to a TCP/IP nameserver that converts a machine name to an internet address. With ISS, a specified machine name corresponds to specified nodes in the SP1. ISS polls the specified nodes regularly to determine which node is best suited to support a login session, and provides the internet address of this node to the nameserver. A user or application connects to the SP1 by specifying a generic SP1 or cluster node name which corresponds to the least-loaded node. In this way, ISS balances user logins across all the available nodes of an SP1 or a cluster of RISC System/6000's.

Chapter 1. Getting Started with LoadLeveler

This chapter introduces you to the LoadLeveler* program product, otherwise referred to as LoadLeveler throughout this manual. It provides information on what LoadLeveler is, the tasks you can perform using LoadLeveler, as well as the Graphical User Interface (GUI).

What is LoadLeveler?

LoadLeveler is a batch job scheduling application that allows you to run more jobs in less time by matching your processing needs to your available resources for improved performance and faster turnaround. As a job scheduler, LoadLeveler runs as a set of daemons on each node in your network. It provides a facility for building, submitting, and processing batch jobs quickly and efficiently in a dynamic environment.

Your batch jobs can be either *serial* jobs or jobs that have been *parallelized* to take advantage of the parallel architecture of the IBM 9076 Scalable POWERparallel Systems* (SP1*). The SP1 is a scalable, parallel, high-performance computing system that supports batch serial and parallel processing. Its software allows you to fully utilize the computing power of the SP1 architecture. Serial jobs are processed sequentially whereas, in this context, parallel jobs are jobs that have been coded using the Application Programming Interface (API) of the IBM AIX Parallel Environment* program product. Parallel batch jobs are as easy to schedule as are serial batch jobs. LoadLeveler just requires that you make a simple modification to your job files and then schedules your parallel job to an SP1 machine.

With the increased support for machines outside of the LoadLeveler cluster that run the Network Queueing System (NQS), you can now run your LoadLeveler jobs or NQS scripts either on machines in the LoadLeveler cluster or on machines outside of the cluster. If you are submitting an NQS script to LoadLeveler for processing, LoadLeveler handles it in one of the following ways:

1. LoadLeveler translates the NQS options in the script to equivalent LoadLeveler commands and submits the job to a machine in the LoadLeveler cluster for processing.
2. LoadLeveler routes the NQS script to an NQS machine outside of the LoadLeveler cluster for processing.

LoadLeveler's approach to network job scheduling for UNIX batch jobs differs from the NQS approach. While the NQS-based approach involves managing pipe and batch queues, LoadLeveler lets you view the total computational capabilities as one single source. Therefore, job requirements are dynamically matched with the best available resources based on your preferences, globally defined scheduling policies, and priorities.

LoadLeveler schedules your job on machines in the cluster that have the right operating system and architecture to give you the best turn-around possible. This cluster can consist of RISC System/6000s, an SP1, and the newly supported Sun Microsystems**, Silicon Graphics**, and HP** workstations.

In addition to supporting different hardware platforms, LoadLeveler now schedules a variety of types of jobs that can run on some or all of the hardware platforms supported. The following table summarizes this information:

These jobs:	Can run on these machines:
Batch-serial	Any binary compatible machine in the LoadLeveler cluster.
Batch-parallel	An SP1.

LoadLeveler has both a command line interface and a Motif** based GUI. As an end user, you'll find that LoadLeveler makes it easy to submit and cancel jobs, monitor job status, and set and change job priority.

As a system administrator, you'll find LoadLeveler easy to install and maintain. In addition, system administrators can perform the same tasks as end users plus checking and changing machine status, assigning job class, starting and stopping jobs, and scheduling jobs on all nodes.

LoadLeveler runs in environments where the Network File System (NFS)** or Andrew File System (AFS)** is installed.

What Types of Machines are in the Cluster?

Your LoadLeveler cluster may contain many machines. Each of these machines can play one or more of the following roles:

- Execute LoadLeveler jobs.
- Allow users logged in to this machine to submit LoadLeveler jobs.

  If the user can only use this machine to submit jobs, this machine is known as a submit-only machine.

  The submit-only machine feature allows workstations that are not part of the LoadLeveler pool to submit jobs to the pool for processing. In addition, you can use these workstations to cancel and query jobs. This means that if your machine is a submit-only machine, it contacts a public-scheduling machine that in turn, contacts a machine that will execute the job.
- Participate in the scheduling of LoadLeveler jobs.

  This type of machine runs a process which accepts a job from **llsubmit**, stores the job on the local disk, and sends it to the central manager to be scheduled. This type of machine is sometimes referred to as a public-scheduling machine. Public-scheduling machines act as liaisons between the machine used to submit the job and the machine where the central manager resides.

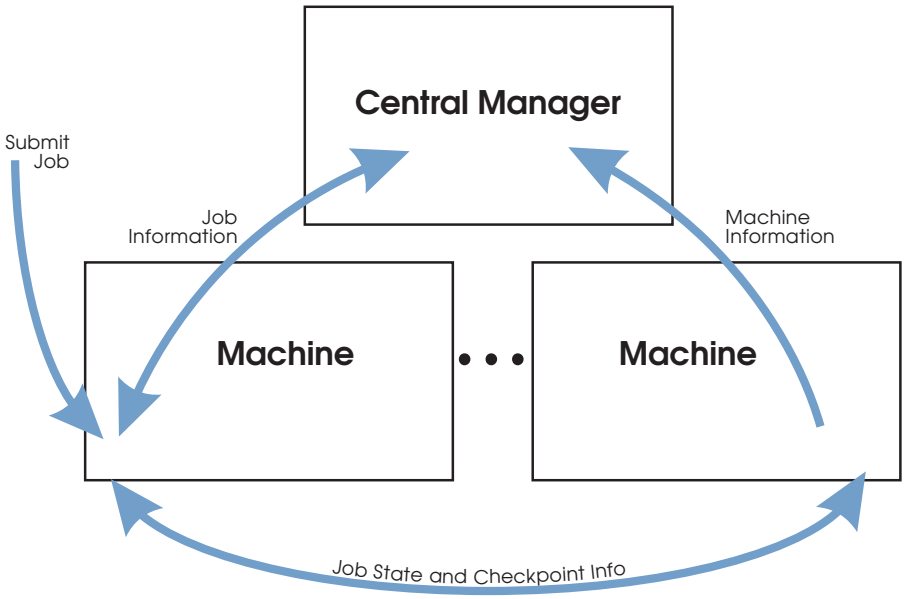
  If this machine accepts jobs that were submitted from other machines, such as submit-only machines, it is referred to as a public-scheduling machine. Public-scheduling machines ensure that the submitted job meets the necessary requirements, such as being syntactically correct, before it passes the job onto the central manager machine where it is then scheduled to a machine that will run the job.

Keep in mind that one machine can play all three of these roles. Therefore, you may be able to submit a job from your machine and have it executed on the same machine.

What is the Central Manager?

In each LoadLeveler pool there is a machine known as the Central Manager. The Central Manager's principal function is to coordinate LoadLeveler related activities of all the machines in the pool. It maintains the status of all machines and jobs and makes decisions on where jobs should be run.

Normally, there is no need for you to know which role your machine plays or the function of the central manager. Being familiar with this, however, may help you to use LoadLeveler more effectively. The following figure illustrates the relationship between the central manager and the other machines in the cluster. For more detailed information on how LoadLeveler works or how information flows through the LoadLeveler pool, refer to the *IBM LoadLeveler: Administration and Installation Planning Guide*.



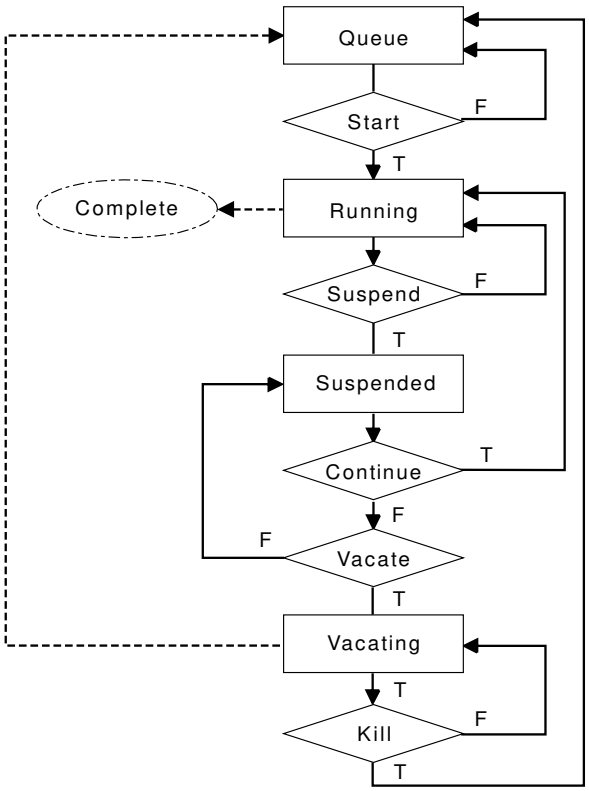
It is important to remember that if the central manager should go down, there is no loss of job information. The jobs that are currently running will continue to run and the jobs that have been submitted but that are not running will be dispatched as soon as the central manager is restarted. In addition, you can continue to submit jobs to LoadLeveler. These jobs will be dispatched when the central manager is restarted.

When the central manager is restarted, it will reconstruct the machine status information from the **startd** daemon and job status information from the **schedd** daemon.

What Tasks Can I Accomplish Using LoadLeveler?

You can use LoadLeveler to perform a variety of job related tasks such as building, submitting, and checking the status of a job in the LoadLeveler pool.

After the central manager selects a job for execution and the job enters the queue, the job can be in one of several phases as shown in the figure on the following page.



In the previous figure, the rectangles represent phases a job can be in and the diamonds represent decisions.

Start The job is running on a machine that meets all the resource requirements requested by the job.

Complete The job has terminated processing.

Suspend The job has been suspended. As an example, the machine hosting this job may have become busy with many interactive foreground activities and is no longer willing to support the batch job. Suspended LoadLeveler jobs will either be continued or vacated.

Continue The job has resumed execution on the same machine and will run to completion unless it is suspended again.

Vacate The job is being vacated from the machine that has been executing the job. Jobs suspended for an amount of time greater than some threshold are usually vacated. If the job is a LoadLeveler checkpoint job, the checkpoint files are sent to the machine that has been monitoring the execution of this job. Under some circumstances the job will go from the Vacate state to the Kill state. Usually, the job is simply returned to the queue for dispatching. When it re-enters the

Start state at some future point in time, a non-checkpoint job will start from the beginning and a checkpoint job will resume from the last checkpoint.

Kill A job taking too much time to vacate may be moved to the Kill state. The job being vacated may have a very large checkpoint file and the time spent in vacating the job has exceeded some threshold. When the job enters the Kill state, the attempt to send checkpoint files to the machine that has been monitoring the execution of the job will be aborted. The job is returned to the queue for dispatching. When it re-enters the Start state at some future point in time, a non-checkpoint job will start from the beginning and a checkpoint job will resume from the last successful checkpoint.

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 pool. They may be modified to meet local requirements.

The following section describes the most common tasks you can perform using LoadLeveler. You can perform most of these tasks using either the GUI or a line command.

Building a Job Command File

Before you can submit a job or perform any other job related tasks, you first need to build the job command file. A job command file specifies the name of the job that you want to submit and can contain other LoadLeveler statements. There are two types of job command files. The first type is for use when executing a binary file. In this case, you need to use the **executable** keyword (discussed later) to specify the executable that you want to process. The second type of job command file is for use when executing shell scripts. In this case, you do not use the **executable** keyword in the job command file. Therefore, the shell script itself is considered to be the executable.

Both of these types of job command files consist of two parts:

- 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. Any keyword or keyword statement is preceded by # @. These lines are treated as comments by the UNIX shell. The name of the job you want to submit is specified with the **executable** keyword. For information on this keyword and other keywords, refer to Appendix A, “LoadLeveler Job Command File Keywords and Examples” on page 51

- Comment statements

  You can use comments to document your job command files, making it easier to understand when you read them later. Since the job command files are shell scripts with the LoadLeveler keyword statements added, you can add comment lines to the file as you would in any shell script. Any line whose first non-blank character is a pound sign (#) and is not a keyword statement is regarded as a comment.

You can build a job command file either by using the GUI or by using a text editor. The following is an example of a simple job command file. For information on the keywords used in job command files, refer to Appendix A, “LoadLeveler Job Command File Keywords and Examples” on page 51.

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

In addition to building job command files to submit to serial-batch processing machines, you can also build job command files to submit to parallel batch machines. Before constructing these job command files, consult your LoadLeveler system administrator to see if your installation is configured for parallel batch job submission.

To construct a parallel job command file, specify, either on the GUI or by using the text editor, the number of machines the parallel job requires to be processed. To do this, use the **parallelmachines** keyword described in Appendix A. If you do use this keyword, do not use the **executable** keyword. Also, add the Parallel Environment variable **MP_PROCS** which specifies the number of parallel processes that the program needs to run. This environment variable must be set equal to the number of parallel machines specified with the keyword **parallelmachines**. For example, assume you wanted to run a parallel program called *my_parallel_program* that requires five parallel nodes to run. Your job command file would look similar to:

```
#
# @ executable =
# @ input = job.in
# @ output = job.out
# @ error = job.err
# @ parallelmachines = 5
# @ checkpoint = no
# @ queue
    my_parallel_program MP_PROCS = 5
```

In this example, checkpointing is set to no because you cannot checkpoint parallel jobs. Therefore, they should not be linked to the LoadLeveler checkpointing libraries.

You can also use multiple MP_PROC statements as long as you do not exceed the number of parallel machines requested with the **parallelmachines** keyword.

Editing a Job Command File

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

Checkpointing a Job

Checkpointing is a method of periodically saving the state of a job so that, if for some reason, the job does not complete, it can be restarted from the saved state. Checkpointing also allows the migration of a long running job to another machine when the first machine is required for other work. Migration can be initiated automatically (by the LoadLeveler scheduler) or upon command from the system administrator. Migration might be initiated automatically because an interactive workstation becomes active again. The system administrator might migrate a job because he wants to perform maintenance on a machine which is executing a very long running job.

To specify that a job be checkpointed when using the GUI to build a job command file, select “yes” for the checkpoint option. When working with a job command file directly, you must include the keyword statement, “checkpoint=yes”. For more information on this keyword, see Appendix A, “LoadLeveler Job Command File Keywords and Examples” on page 51. For detailed information on checkpointing jobs, refer to Appendix C, “Checkpointing Jobs” on page 79.

Setting and Changing the Priority of a Job

Every job has a queue priority associated with it. This means that LoadLeveler dispatches jobs in the queue based upon their queue priority. The higher the queue priority of a job, the sooner it will be dispatched.

You can set or change the priority of any job you submit to LoadLeveler. This includes parallel and serial jobs and NQS jobs. When you build a job you can set a *user priority* for that job by assigning it a number between 0 and 100, inclusive. The higher numbers correspond with higher priority.

When you submit the job, you can change the priority while it is still in the queue or after it has begun to run. One reason you might want to change the priority of a job is if it is restartable. You can change the priority of a job still in the queue by either increasing or decreasing the current value. The number you specify is added to the previous priority you specified and the priority of your job changes in relation to other jobs you previously submitted that are still in the queue.

In addition to user priorities, your system administrator can assign a *class priority* to a group of jobs. For example, he may assign all long-running jobs a lower class priority than fast-running jobs. If you change the user priority of one of your jobs, it will only override another one of your jobs if the second job has an equal or lower class priority than the first.

For example, assume the following:

- Your userid is JANE.
- John and Paul, two co-workers, have userids JOHN and PAUL, respectively.
- All three of you have jobs in the queue and each job has a priority.
- There are two classes assigned (Class A and Class B)

- Class A jobs have a higher priority than Class B jobs.
- John's jobs have a Class A priority.
- Jane's job, JANE1, has a Class A priority. The remainder of Jane's jobs have a Class B priority.
- Paul's jobs have a Class B priority.

Job	Priority	Class
JOHN1	50	A
JOHN2	12	A
JANE1	10	A
JANE2	20	B
JANE3	14	B
JANE4	8	B
PAUL1	3	B

You want to increase the priority of JANE4 by adding a value of 10 to the current value. JANE4 now has a priority of 18 and exceeds JANE3 in the queue. JANE4 does not affect the priority of John or Paul's jobs because a user can only increase or decrease priorities of jobs in relation to their own jobs. Also, even though JANE4 now has a higher user priority than JANE1, it does not override JANE1 in the queue because JANE1 has a higher class type.

Job	Priority	Class
JOHN1	50	A
JOHN2	12	A
JANE1	10	A
JANE2	20	B
JANE4	18	B
JANE3	14	B
PAUL1	3	B

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. When submitting a job to be routed to a machine outside of the LoadLeveler cluster, the job command file must specify the **class** keyword. For example:

```
class = class1
```

When you submit a job, LoadLeveler assigns the job a three part identifier. This identifier is made up of:

- Machine name that participates in the scheduling of jobs.

  This is the name of the machine that schedules the LoadLeveler jobs. This may not be the name of the machine on which the job is running.

- Cluster Id

  A *cluster* is often defined as a group of processors interconnected through a network. LoadLeveler has a unique definition for the term *cluster id*. A *cluster id*, is an identifier that is given to a group of jobs that were initiated from the same job command file. For example, if you created a job command file that submitted the same job five times, possibly with different input and output, each job would have the same cluster id.

- Proc Id

This id is unique for every job you submit. Multiple jobs can have the same cluster id but each job will have a unique proc id.

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 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

You 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.

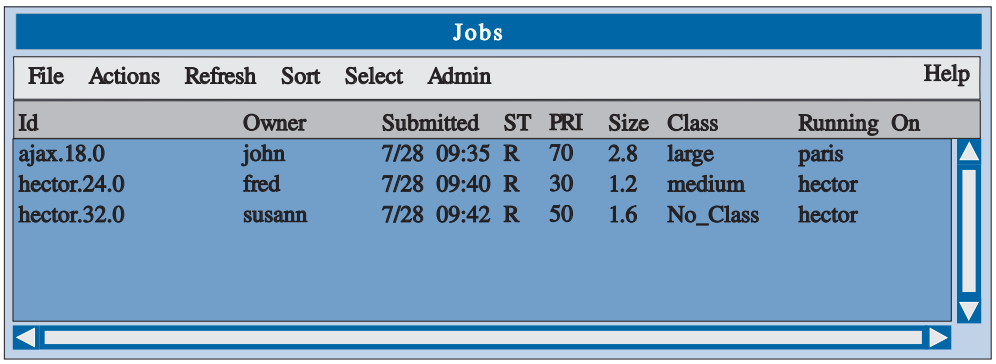
Submitting a Job Command File Using a Submit-Only Machine: You can take advantage of the submit-only machine feature that allows workstations that are not part of the LoadLeveler pool to submit jobs to the pool for processing. You cannot run the GUI on a submit-only machine so you will have to submit a job using the **llsubmit** command.

To start a submit-only machine, follow the procedure in the *LoadLeveler Administration and Installation Planning Guide* to install the submit-only code on your machine.

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

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.



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 using **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: The submit-only machine, in addition to allowing you to submit and cancel jobs, allows you to query the status of jobs. You cannot, however, run the GUI on a submit-only machine so you will have to query a job using the **llq** command.

Placing and Releasing a Hold on a Job

Sometimes you may want to place a hold on a job because you do not want it to run immediately. Placing a hold on a job means that the job remains in the queue until you release the hold on 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. For more information on system holds, see the *IBM LoadLeveler: Administration and Installation Planning Guide*, SH26-7220.

As a user or system administrator, you can also use the **startdate** keyword described in Appendix A 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 November 23, 1993 at 11:30 am, you would enter:

```
startdate = 11/23/93 11:30
```

Cancelling a Job

You can cancel a job that is either in the queue or has left the queue and is running.

Cancelling a Job that was Routed to NQS: You can also cancel a job that was routed to NQS using either the GUI or the **llcancel** command.

Cancelling a Job Using a Submit-Only Machine: The submit-only machine, in addition to allowing you to submit and query jobs, allows you to cancel jobs. You cannot, however, run the GUI on a submit-only machine so you will have to cancel a job using the **llcancel** command.

Working with the Machines and Clients

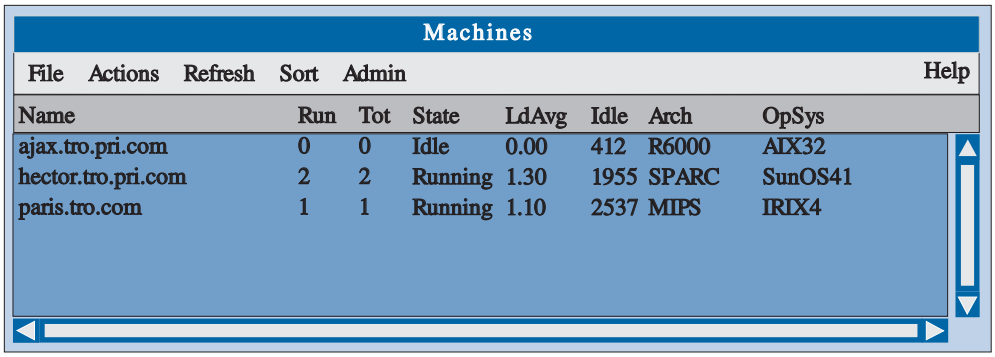
Throughout this book and the LoadLeveler library, the terms *client*, *workstation*, *machine*, and *node* refer to the machines in your cluster. For information on the roles these machines can play, refer to “What Types of Machines are in the Cluster?” on page 2

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

- 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 pool, 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 machines state and amount of installed memory.



- Display 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 jobs in the system by communicating with each node. 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. For example, if you submit a job and it is not running, it may be because the central manager is inoperative. By determining the location of the central manger, you can determine if it is still operating.

  You may also need to display the location of the central manager 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**. You can find out the names of all these machines in the cluster.

  Submit-only machines allow workstations that are not part of the LoadLeveler pool to submit jobs to the pool for processing.

How Do I Use LoadLeveler's Graphical User Interface?

This section briefly describes LoadLeveler's Graphical User Interface (GUI). For detailed information on a GUI and how to use one, refer to Appendix B, “Graphical User Interface Information” on page 65, or to the appropriate Motif documentation.

LoadLeveler's GUI contains many items common to GUIs including windows, menus, pull-down menus, and dialog boxes.

LoadLeveler's Windows

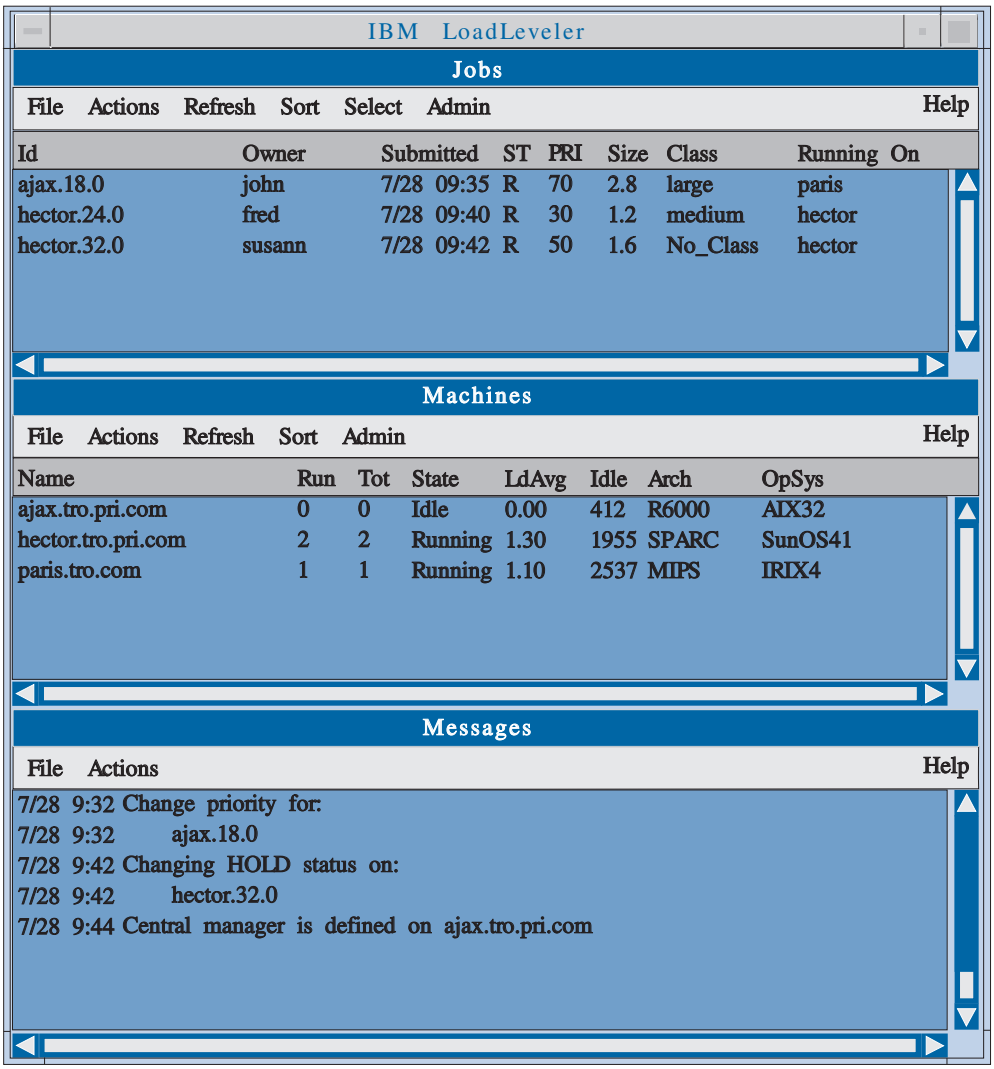
A *window* is the part of a screen in which you can interact with LoadLeveler. LoadLeveler has both primary windows and secondary windows. A primary window is the window in which the main interaction between you and LoadLeveler takes place. The LoadLeveler primary window has three sub-windows:

Jobs window
Use to view job status information, as well as to build and submit job command files.

Machines window
Use to query which machines are available and which ones are running jobs. You can also display detailed information about a machine such as its architecture, operating system, and installed memory.

Messages window
Use to view messages generated by LoadLeveler.

You can view the primary window as LoadLeveler's “main window”, to which you can always return. If you close a primary window, all secondary windows associated with that window disappear. Closing the last primary window exits LoadLeveler. The recommended way, however, of exiting LoadLeveler is to choose the Exit option from the file pull-down menu. The figure on the following page is LoadLeveler's primary window:



Secondary windows prompt you for some input. Secondary windows extend the dialog presented in the parent window, providing more information that you may need in order to complete processing. A Help window is an example of a secondary window. It provides additional information that can help you complete your task.

Another example of a secondary window is a *dialog box*. A dialog box is a movable window that contains controls such as push buttons. You use these controls to provide information to LoadLeveler. The push buttons are arranged across the bottom of the dialog box. There are a number of common dialog box actions. For more information on the push buttons available in LoadLeveler dialog boxes, refer to Appendix B, “Graphical User Interface Information” on page 65.

LoadLeveler's Menu Bars and Pull-down Menus

LoadLeveler's primary window has three sub-windows. Each of these sub-windows has its own menu bar. For a definition of a menu bar, refer to Appendix B, “Graphical User Interface Information” on page 65.

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.

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:

- There are three menu bars on the main window.

  Normally, a window only has one menu bar. LoadLeveler's main window has three menu bars. LoadLeveler was designed this way to make it faster and easier for you to interact with the three sub-windows.

- There are no mnemonics or accelerator keys.

  These are keyboard alternatives to mouse button actions.

- On some dialog boxes, such as the Build dialog box that you use to build a job command file, you need to keep the mouse cursor in the field into which you want to enter text.

- When you display a window and then display a second window, the second window may overlap the first window preventing you from being able to view the first window. In order to locate the first window, you may need to move overlapping windows.

Customizing the Graphical User Interface

You can customize the GUI to suit your needs by overriding default settings. For example, you can set the color, initial size, and location of the main window. For more information, refer to the Xloadl file that exists under **/usr/lib/X11/app-defaults**. See “Customizing the Graphical User Interface” on page 67.

Chapter 2. 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 Chapter 3, "Building and Submitting Jobs Using LoadLeveler Commands" on page 29.

Before using the GUI to build and submit jobs, you should know how to perform the tasks described in this manual. 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 23.
Cancel a job	Refer to "Step 9: Cancel a Job" on page 25.

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

Starting Up the LoadLeveler Graphical User Interface

To start up the graphical user interface, check your path variable to ensure that it is pointing to the LoadLeveler binaries and tools directories. Then, check to see that your display variable is set to your display to ensure that the GUI appears on your display. Finally, type `xloadl` to bring up the GUI. If you want the GUI to run as a background process, type `xloadl &`.

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.

If you do not want to work through the steps presented in this chapter and you know what task you want to perform, refer to the following table:

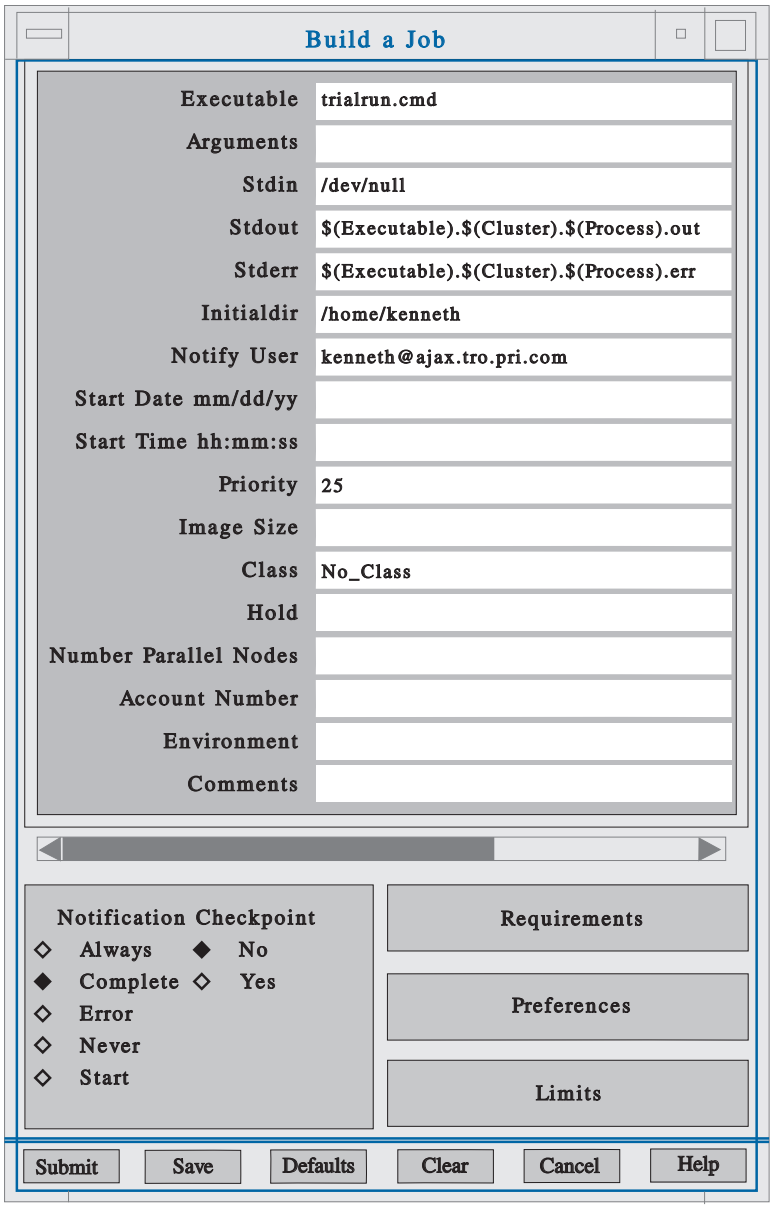
To:	**Go to:**
Build a job	Step 1
Cancel a job	Step 9
Change priorities of jobs in a queue	Step 6
Find the location of the central manager	Step 12
Find the location of the public scheduling machines	Step 13
Display and refresh job status	Step 4
Display and refresh machine status	Step 10
Edit the job	Step 2
Hold a job	Step 7
Release a hold on a job	Step 8
Save LoadLeveler messages in a file	Step 15
Sort the Jobs window	Step 5
Sort the Machines window	Step 11
Specify which jobs appear in Jobs window	Step 14
Submit a job command file	Step 3

Step 1: Build a Job

From the Jobs window:

SELECT File → Build a Job

▲ The Build a Job dialog box appears:



Complete those fields for which you want to override what is currently specified in your skel.cmd defaults file.

Field	Input	Required or Optional
Executable	Name of the program to run. It must be an executable file.	Optional. If omitted, the command file is executed.

Field	Input	Required or Optional
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 $(Executable).$(Cluster).$(Process).out
Stderr	Filename to use as standard error (stderr) by the program.	Optional. The default is $(Executable).$(Cluster).$(Process).err
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.
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.
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.
Number of parallel nodes	Number identifying the number of machines required to process the parallel job.	Optional. The default is 0.
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. For information on the environment specifications, see page 54. This string must be limited to 256 characters.	Optional.
Comments	Comments associated with the job. These comments help to distinguish one job from another job. They are not incorporated into the job command file.	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 Appendix A.

SELECT a Notification option

Your choices are:

Always Notify you when the job starts, completes, and if it incurs errors.

Start Notify you when job starts.
Complete Notify you when job completes. This is the default option as initially defined in the skel.cmd file.
Error Notify you if job cannot run because of an error.
Never Do not notify you.

SELECT a Checkpoint option

Your choices are:

Yes Checkpoint the job.
No Do not checkpoint the job.

Refer to "Checkpointing a Job" on page 7 for information on checkpointing a 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. For any requirements that you specify, LoadLeveler dispatches your job only to one of those machines with resources that match your requirements.

Field	Input	Required or Optional
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	Quantity 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	Quantity 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
Hint: If you type in a resource that is not available, you will NOT receive a message. LoadLeveler holds your job in the queue until the resource becomes available. Therefore, ensure the spelling of your entry is correct.		

PRESS Close to return to the Build a Job dialog box.

PRESS Preferences

▲ The Preferences dialog box appears.

This dialog box 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 before your job runs. 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 effect for your userid on the machine that runs the job are used.

Field	Input	Required or Optional
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. For information on the units, see the online help.	Optional
Core Limit	Maximum size of a core file. For more information, see the online help.	Optional
RSS Limit	Maximum size of the resident set size. It is the largest amount of physical memory a user's process can allocate. For more information, see the online help.	Optional
File Limit	Maximum size of a file that is created. For more information, see the online help.	Optional
Stack Limit	Maximum size of the stack. For more information, see the online help.	Optional

PRESS Close to return to the Build a Job dialog box.

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.
Run the program immediately and discard the information you entered	**PRESS** **Submit** **GO TO** Step 4

To edit the job you created, go to “Step 2: Edit the Job.”

If you already submitted your job, go to “Step 4: Display and Refresh LoadLeveler Job Status” on page 23. Otherwise, go to “Step 3: Submit a Job Command File” on page 23.

Step 2: Edit the Job

To edit the job you just built, from the Jobs window:

SELECT **File → Submit a Job**

▲ The Submit a Job dialog box appears.

SELECT the job file from the file column you want to edit.

PRESS **Edit**

▲ Your job command file appears in a window. You can use any editor to edit the job command file but the editor specified in your .Xdefaults file is the default editor.

If you have an icon manager, an icon may appear. An icom 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.

Go to the next step.

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 from the file column you want to submit.

You can also use the filter field and the directories column to select the job or you can type in the job name in the text entry field.

PRESS **Submit**

▲ The job is submitted for processing.

You can now submit another job or you can press Cancel to exit the window.

Go to the next step.

Step 4: Display and Refresh LoadLeveler 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 **Refresh → Refresh Job Status** or **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.

Go to the next step.

Step 5: Sort the Jobs Window

You can specify a sorting option for the Jobs window. The option you specify determines how the jobs appear in the Jobs window.

From the Jobs window:

To:	Select Sort →
Sort by name	**Sort by Name**
Sort by the time the jobs were submitted	**Sort by Submission Time**
Sort jobs by their user priority	**Sort by Priority**
Sort jobs by the machine to which they were submitted.	**Sort by Machine**
Sort the jobs in the order the central manager attempts to dispatch them. Jobs may actually be dispatched in a different order if the highest priority jobs require resources that are not available. A lower priority job that requires different resources which are available may get dispatched first.	**Sort by Dispatch Order**
Sort by class	**Sort by Class**

Go to the next step.

Step 6: Change Priority of Jobs in Queue

If your job has not yet begun to run and is still in the queue, you can change the priority of the job. Only the owner of a job or the LoadLeveler administrator can change the priority of a job. For more information on priorities, see page 7.

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**

▲ 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

When you submit a job to a machine, the status of the machine is automatically displayed in the Machines window. You can update or refresh this status using the Machines window and selecting **Refresh → Refresh Machine Status** or **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 a sorting option for the Machines window. The option you specify determines how machines appear in the window.

From the Machines window:

To:	Select Sort →
Sort by machine name	**Sort by Name**
Sort by the number of jobs running	**Sort by Run**
Sort jobs by the number of jobs submitted to the selected host	**Sort by Total**
Sort jobs by the machine state.	**Sort by State**
Sort by load average	**Sort by LdAvg**
Sort by keyboard idle time	**Sort by Idle**
Sort by hardware architecture	**Sort by Arch**
Sort by operating system type	**Sort by OpSys**
Not specify a sort.	**No Sort**

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 Public Scheduling Machines

Public scheduling machines are those machines that participate in the scheduling of LoadLeveler jobs. Refer to “What Types of Machines are in the Cluster?” on page 2 for more information.

To get a list of these machines in your cluster, use the Machines window:

SELECT Actions → Find Public Scheduler

▲ A message appears displaying the address 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 specify that all jobs:	Select Select →
In the queue appear	**All***
Belonging to a specific user or group of users appear	**By User***
Submitted on a specific machine or group of machines appear	**By Machine***
Having a particular id appear	**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: * Following this choice, you can further qualify which jobs appear in the Jobs window by specifying only jobs running, jobs idle, or all jobs.	

Select **Select → Show Selection** to refresh the window.

Go to the next step.

Step 15: 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.

Chapter 3. Building and Submitting Jobs Using LoadLeveler Commands

This chapter explains how to build and submit a job to LoadLeveler using the LoadLeveler commands. For detailed descriptions of the commands, refer to Chapter 4, "LoadLeveler Commands" on page 33. In addition, you will learn how to perform other job related tasks. You can accomplish these same tasks by using the GUI. For information on using the GUI, refer to Chapter 2, "Building and Submitting Jobs Using the Graphical User Interface" on page 17.

Task Scenario Using the Commands

The tasks described in this chapter are presented in a typical step by step scenario that you, as a user, might be interested in accomplishing. 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 before building a job.

If you do not want to work through the steps presented in this chapter and you know what task you want to perform, refer to the following table of alphabetized tasks to locate the desired task.

To:	**Go to:**
Build a job	Step 1
Cancel a job	Step 9
Change the priority of a job	Step 5
Display the central manager	Step 10
Display the LoadLeveler clients	Step 11
Display the status of a job	Step 4
Display the status of a machine	Step 8
Edit a job you built	Step 2
Hold a job	Step 6
Release a hold on a job	Step 7
Submit a job	Step 3

For this task scenario, assume the following:

- You are submitting jobs from a machine called *wizard*
- The job command file is *longjob.cmd*
- Your userid is *mary*

Step 1: Build a Job

Since you are not using the graphical user interface, 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. For example:

longjob.cmd

```
# 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
```

Go to the next step.

Step 2: Edit a Job

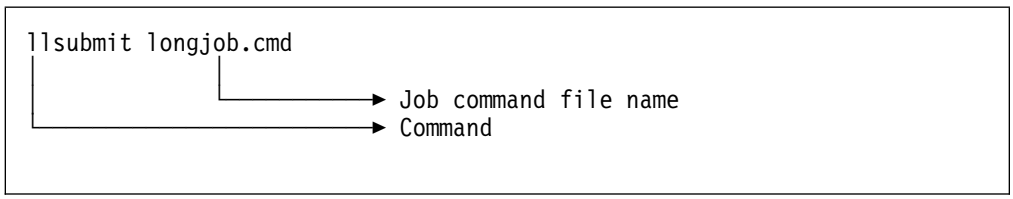
You can optionally edit the job command file you created in step 1.

Go to the next step.

Step 3: Submit a Job

To submit the job command file that you created in step 1, use the **llsubmit** command. See “llsubmit - Submit a Job” on page 50 for the command syntax.

To submit the job command file you created, type in:



LoadLeveler responds by issuing a message similar to:

```
submit: 1 job has been submitted, "wizard.22.0".
```

where *wizard* is the name of the machine to which the job was submitted, 22 is the cluster id and 0 is the proc id.

Although you can obtain this information later on using various methods, record the three part identifier for use later on in this example.

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. See “llq - Query Job Status” on page 42 for the command syntax.

To request the status of the job you previously submitted, type:

```
llq wizard.22.0
    |      |  |
    |      |  └─► Proc id
    |      └────► Cluster id
    └───────────► Machine name to which the job was submitted
```

Step 5: Change the Priority of a Job

You can change the priority of a job that is in the queue or one that is running. 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 effect the job while it is running. It only affects the job if the job re-enters the queue to be dispatched again.

To change the priority of a job, use the **llprio** command. Refer to “llprio - Change Priority of Submitted Job” on page 40 for the command syntax.

To increase the priority of the job you submitted by a value of 10, type:

```
llprio +10 wizard.22.0
```

Step 6: Hold a Job

To place a temporary hold on a job in a queue, use the **llhold** command. See “llhold - Hold or Release a Submitted Job” on page 38 for the command syntax. This command only takes affect if jobs are in the queue or if they get suspended once they begin to run.

To place a hold on *wizard.22.0*, type:

```
llhold wizard.22.0
```

Step 7: Release a Hold on a Job

To release the hold you placed in step 6, use the **llhold** command. See “llhold - Hold or Release a Submitted Job” on page 38 for the command syntax.

To release the hold, type:

```
llhold -r wizard.22.0
       |
       └──────► Release from hold
```

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. See “llstatus - Query Machine Status” on page 46 for the command syntax.

To display the status, type:

```
llstatus -l wizard
         |
         └──────► Long listing
```

For information on the long listing, refer to -- Heading 'LQLIST' unknown --.

Step 9: Cancel a Job

To cancel *wizard.22.0*, use the **llcancel** command. See “llcancel - Cancel a Submitted Job” on page 36 for the command syntax.

To cancel the job, type:

```
llcancel wizard.22.0
```

Step 10: Display the Central Manager

When you used the **llstatus** command in step 8, the system displays the name of the machine on which the central manager is running.

Step 11: Display the Public Scheduling Machines

Public scheduling machines are those machines that participate in the scheduling of LoadLeveler jobs. When you used the **llstatus** command in step 8, the system displayed the names of these machines.

Chapter 4. LoadLeveler Commands

The following table is a brief listing of the user commands and their uses. For information on the administrator commands, see *IBM LoadLeveler: Administration and Installation Planning*.

Command	Purpose
llcancel	Cancels one or more jobs from the queue.
llhold	Places a user or system hold on one or more jobs in the queue. Also, releases a hold.
llprio	Changes the priority of one or more jobs in the queue.
llq	Returns information about one or more jobs in the queue.
llstatus	Returns status information about one or more machines in the pool.
llsubmit	Submits a job to the central manager which dispatches it to an appropriate machine based on requirements in a command file.

How to Read the Syntax Diagrams

Purpose

For each LoadLeveler command, a diagram shows you how to enter the required information correctly on the command line. These diagrams show:

- Which flags can be entered
- Which flags take parameters
- Which flags can and cannot be entered together
- Where you must enter flags or parameters and where you have a choice
- Where you can repeat flag and parameter sequences.

The following discussion explains how to interpret the syntax diagrams. It begins with an example diagram that shows all of the conventions used in the LoadLeveler commands. Each item in the diagram is explained.

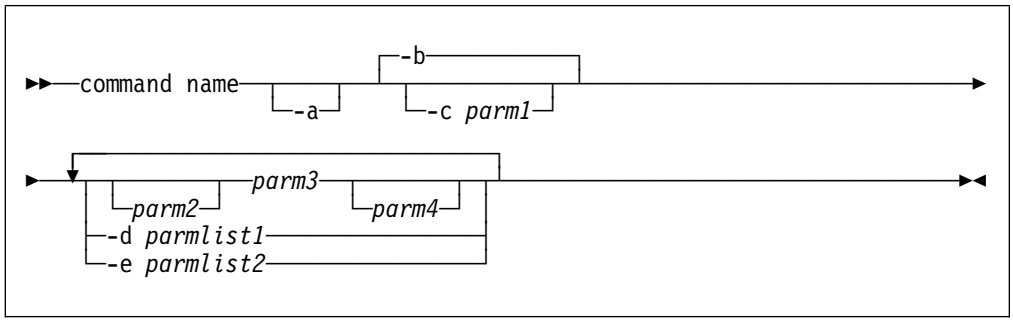
Diagram items that you must enter literally are shown in normal typeface. These items include the command name, all flags, and literal characters.

Variable items appear in italics. These items include parameters that follow flags and parameters that the command reads, such as machine names, user names, and job identifiers.

Required items appear on the central line of the command path.

Default items appear above the central line.

Optional choices are listed below the central line.



where

-a is an optional flag. (Optional flags appear below the line.)

-b is a default flag (defaults appear above the line) that will automatically be in effect unless optional flag **-c** is specified.

-c *parm1* is an optional flag that you can specify instead of flag **-b**. The **-c** flag takes a parameter and requires that information be specified where *parm1* appears.

The single arrow indicates command continuation on the next line.

parm2 is an optional parameter.

parm3 is required if you do not specify the **-d** or **-e** flags.

parm4 is an optional parameter.

-d *parmlist1* is required if you do not specify *parm3* or **-e**. If you specify the **-d** flag, it takes a list of parameters delineated by blanks.

-e *parmlist2* is required if you do not specify *parm3* or **-d**. If you specify the **-e** flag, it takes a different list of parameters, delineated by blanks.

You can specify *parm2*, *parm3*, and *parm4*one or more times, in addition to **-d** and **-e**. This is indicated by the arrow that points back to where you first chose the path of *parm3*, **-d**, or **-e**. You can return to choose another of these options or enter the command as is, since no further options or parameters appear on the path.

Examples

Following are examples of how to enter the command based on the syntax diagram.

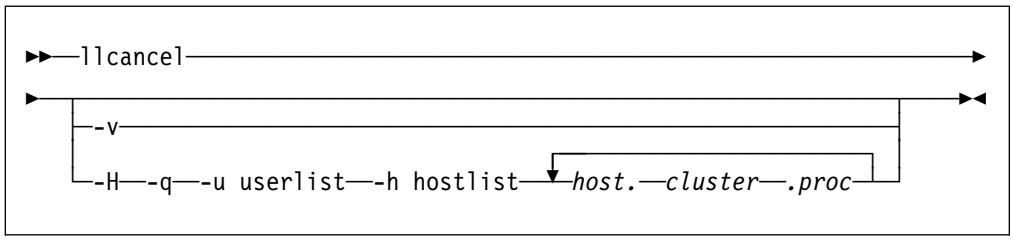
```
command name parm3 -d parmitem
command name -a parm3 parm3 -e parmitem parmitem
command name -a -b parm2 parm3 parm2 parm3 -d parmitem parmitem
command name -a -c parm1 parm3 parm4 parm3 -d parmitem parmitem
command name -c parm1 parm3 parm4 parm3 parm2 parm3 -e parmitem parmitem
```

llcancel - Cancel a Submitted Job

Purpose

Cancels one or more jobs from the LoadLeveler queue.

Syntax



Parameters

where

-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.

host. is the name of the host to which the job was submitted (delimited by dot). The default is the local host.

cluster is the cluster id assigned to the job when it was submitted using the **llsubmit** command.

.*proc* (delimited by dot) is the proc number 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 host machines in the *hostlist* are cancelled. When used alone, only the user's jobs monitored on the **schedd** machine are cancelled.

-h *hostlist* is a blank-delimited list of hosts. All jobs monitored on host machines in this list are cancelled. When issued with the **-u** option, the *userlist* is used to further select jobs for cancellation.

Usage Notes

The **-u** or **-h** flags override the *host.cluster.proc* 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 monitored on machines in *hostlist* 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.

When cancelling a job from a submit-only host, you must specify the host name of the machine that scheduled the job. For example, if you submitted the job from machine A, a submit-only host, and machine B, a scheduling machine, scheduled the job to run, you must specify machine B's host name in the cancel command.

Examples

This example cancels the job bronze.18.3.

```
llcancel bronze.18.3
```

This example cancels all members of cluster id 8 submitted to host machine gold.

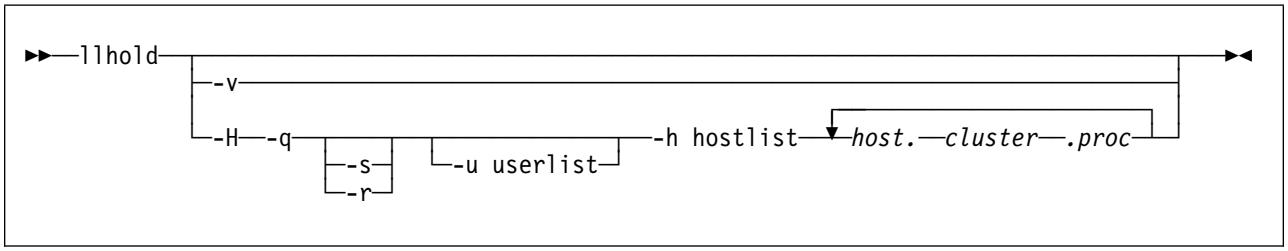
```
llcancel gold.8
```

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



where

-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 under which the command was built.

-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 host 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 hosts. All jobs monitored on host 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 hosts in *hostlist*.

When issued by an administrator, all jobs monitored on the hosts 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 host machines in the *hostlist* are acted upon.

host is the name of the host to which the job was submitted (delimited by dot). The default is the local host.

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.

cluster is the cluster id assigned to the job when it was submitted using the **llsubmit** command.

.*proc* (delimited by dot) is the proc number assigned to the job by LoadLeveler when it was submitted using the **llsubmit** command. The default is to include all members of the cluster.

Usage Notes

Only jobs in the system queue can be operated on with this command.

To insure 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 jobs 23.0 and 19.1 on the local host in user hold.

```
llhold 23.0 19.1
```

This example releases jobs 23.0, 19.1 and 20.3 on the local host from a hold state.

```
llhold -r 23.0 19.1 20.3
```

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

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

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

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

This example releases from a hold state all jobs on host machines bronze, iron, and steel that smith submitted.

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

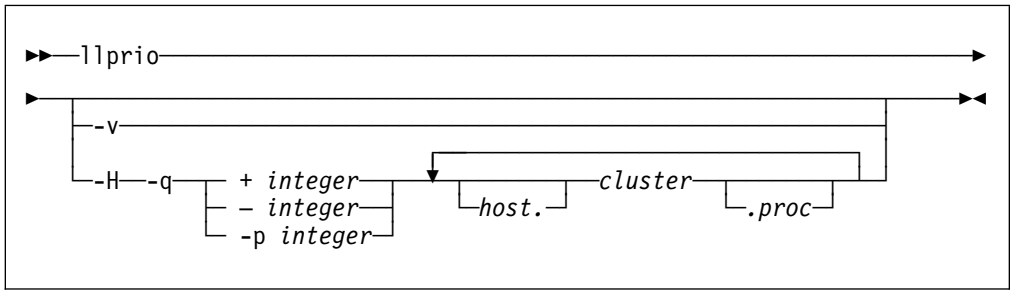
llprio - Change Priority of Submitted Job

Purpose

Changes the priority of one or more jobs in the LoadLeveler queue. It does not affect running jobs. You can adjust the priority by supplying a **+** (plus) or **-** (minus) immediately followed by an *integer*value. The priority of a job ranges from 0 to 100 inclusively, with higher numbers corresponding to greater priority. The default priority is 50. Only the owner of a job or the system administrator can change the priority of that job.

Note: llprio does not preempt a job that is running, even if its priority is lower than other jobs.

Syntax



Parameters

where

-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.

+/– *integer* operates on the current priority of the job, making it higher (closer to execution) or lower (further from execution) by adding or subtracting the value of *integer*.

-p *integer* 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.

host is the name of the host to which the job was submitted (delimited by dot). The default is the local host.

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.

cluster is the cluster id assigned to the job when it was submitted using the **llsubmit** command.

.proc (delimited by dot) is the proc number assigned to the job when it was submitted using the **llsubmit** command.

Usage Notes

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 priority applied by a user is relative only to *that user's other jobs*. If you have three jobs enqueued, you can reorder those three jobs with **llprio** but the result does not affect jobs submitted by other users, regardless of their priority and position in the queue.

Although you can use **llprio** to move one job ahead of others you have submitted, you can only overtake jobs of an equal or lower class priority, as set by the system administrator.

Examples

This example raises the priority of job 4.1 submitted to machine bronze by a value of 25.

```
llprio +25 bronze.4.1
```

This example sets the priority of job 18.4 submitted to machine silver to 100, the highest possible value.

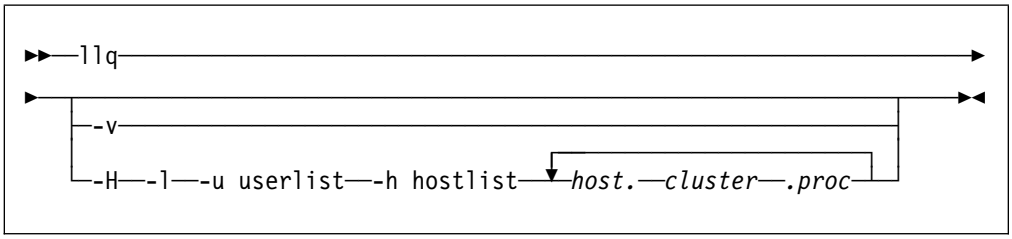
```
llprio -p 100 silver.18.4
```

llq - Query Job Status

Purpose

Returns information about jobs in the LoadLeveler job queue. It does not provide information on NQS jobs. See "Output" on page 43 for a description of the information generated by this command.

Syntax



Parameters

where

-H provides a short usage message.

-l specifies that a long listing be generated for each job for which status is requested. Fields included in the long listing are shown on page 43. Only the owner of a job and the administrator can use this option.

If **-l** is *not* specified, then the standard listing, described on page 43 is generated.

-v outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

*host***.** is the name of the host to which the job was submitted (delimited by dot). The default is the local host.

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.

cluster is the cluster id assigned to the job when it was submitted using the **llsubmit** command.

.*proc* (delimited by dot) is the proc number 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 host 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 hosts. All jobs monitored on host machines in this list are queried. When issued with the **-u** option, the *userlist* is used to further select jobs for querying.

Usage Notes

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

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

Examples

This example generates a long listing for job 8.2 submitted to host machine *gold*.

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

This example generates a standard listing for all members of job cluster 12 submitted to the local host.

```
llq 12
```

Output

The Standard Listing

The standard listing, generated when **-l** is *not* specified with **llq**, includes the following fields:

Id — job identifier presented as *host.cluster.proc* number.

Owner — userid of the job submitter.

Submitted — date and time of job submission.

ST — current job status, where:

- *R* indicates job is running.
- *I* indicates job idle. Job has been submitted, dispatched, vacated, or never run.
- *P* indicates job pending. Job has been dispatched to a server but is not yet running.
- *C* indicates job completed. This only occurs when multiple jobs have been submitted from a job command file.
- *X* indicates job removed. This only occurs when multiple jobs have been submitted from a job command file or when the **llcancel** command is issued while the **schedd** host is down.
- *S* indicates job has been put in system hold. See "llhold - Hold or Release a Submitted Job" on page 38.
- *H* indicates job has been put in user hold. See "llhold - Hold or Release a Submitted Job" on page 38.

PRI — user priority of the job, where the values are defined in the **llprio** command. See "llprio - Change Priority of Submitted Job" on page 40.

Size — size of the executable submitted, in megabytes.

Class — job class.

Running On — if running, the machine the job is running on. This is blank when the job is not running.

The Long Listing

The long listing, returned by **llq** when the **-l** option is specified, includes these fields:

Owner userid of the job submitter.

Queue Date date and time job was received by LoadLeveler.

Status status of the job, where:

- *Running* indicates job is running on the designated server.
- *Idle* indicates job idle. Job has been submitted and dispatched but was vacated or never run.
- *Pending* indicates job pending. Job has been dispatched to a server but is not yet running.
- *Completed* indicates job completed. This only occurs when multiple jobs have been submitted from a job command file.
- *Removed* indicates job removed. This only occurs when multiple jobs have been submitted from a job command file.
- *System Hold* indicates job has been put in system hold. See "llhold - Hold or Release a Submitted Job" on page 38.
- *User Hold* indicates job has been put in user hold. See "llhold - Hold or Release a Submitted Job" on page 38.

Completion Date date and time job was completed.

User Priority priority of the job, as specified by the user.

user_sysprio user system priority of the job, where the value is defined in the configuration and administration files.

class_sysprio class priority of the job, where the value is defined in the configuration and administration files.

System Priority overall system priority of the job, where the value is defined in the configuration and administration files.

q_sysprio submission date priority of the job.

Notifications notification status for the job, where:

- *always* notification is sent through mail for all four jobs 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)
Hold Job Until	job is not dispatched before 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.
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.
Requirements	job requirements as specified when the job was submitted.
Preferences	job preferences as specified when the job was submitted.
Running Host	name of the machine where job is running.
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.
Class	job class as specified when job was submitted.

The following appear only when they have been specified in the job command file.

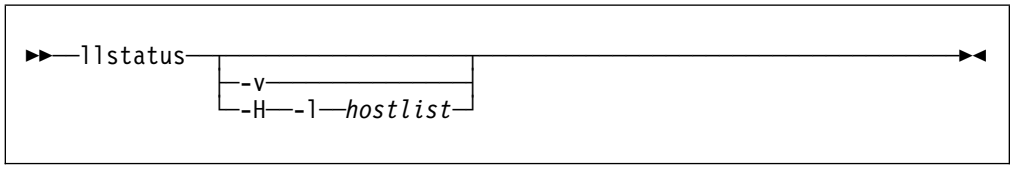
CPU limit	CPU soft and/or hard limit as specified when job was submitted.
data limit	data hard and soft limits as specified when job was submitted.
core limit	core hard and soft limits as specified when job was submitted.
file limit	file hard and soft limits as specified when job was submitted.
stack limit	stack hard and soft limits as specified when job was submitted.
RSS limit	RSS hard and soft limits as specified when job was submitted.

llstatus - Query Machine Status

Purpose

Returns status information about machines in the LoadLeveler pool. It does not provide status on any NQS machine. See “Output” on page 47 for a description of the information generated by this command.

Syntax



Parameters

where

-H provides a short usage message.

-l specifies that a long listing be generated for each machine for which status is requested. The fields included in the long listing are described in “Output” on page 47. If **-l** is *not* specified, the standard list is generated.

-v outputs the name of the command, release number, service level, service level date, and operating system used to build the command.

hostlist is a blank-delimited list of host machines for which status is requested.

Usage Notes

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 host machines silver and gold machines.

```
llstatus -l silver gold
```

Output

The Standard Listing

The standard listing, generated by **llstatus** when **-l** is *not* specified, includes the following fields:

Name hostname of the machine.

Run number of jobs submitted to this machine that are running on this machine or else where in the cluster.

Total total number of jobs submitted to this machine that are currently in queue.

State state of the machine, where:

- *Busy* Maximum number of jobs are running on this machine.
- *Down* LoadLeveler is not running on this machine. A machine obtains this state if the daemons on that machine have not been able to report their status to the central manager. This may occur because the machine is actually down or there is a network failure.
- *Drain* running jobs continue but no additional jobs are started on this machine.
- *Flush* all jobs on this machine are flushed. No new jobs will be accepted.
- *Idle* machine is running no jobs.
- *None* LoadLeveler is running on this machine but no jobs can be run.
- *Running* machine is running one or more jobs and is capable of running more.
- *Suspend* machine has suspended all jobs due to a system suspend.

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, returned by **llstatus** when the **-l** option is specified, includes these fields:

Name hostname of the machine.

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 vacated jobs are killed on this machine.

SYSPRIO actual expression that determines overall system priority of the job, where the value is defined in the configuration file.

Users number of different users with jobs submitted to this machine.

Running number of jobs submitted to this machine that are running.

Idle number of jobs submitted to this machine that are not running.

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 machine, where:

- *Busy* Maximum number of jobs are running on this machine.
- *Down* LoadLeveler is not running on this machine. A machine obtains this state if the daemons on the machine have not been able to report their status to the central manager. This may occur because the machine is actually down or there is a network failure.
- *Drain* running jobs continue but no additional jobs are started on this machine.
- *Flush* all jobs on this machine are flushed. No new jobs will be accepted.
- *Idle* machine is running no jobs.
- *None* LoadLeveler is running on this machine but no jobs can be run.
- *Running* machine is running one or more jobs.

Suspend machine has suspended all jobs due to a system suspend.

Entered Current State
date and time when machine state was set.

Virtual Memory available swap space, in kilobytes, on this machine.

Memory physical memory, in megabytes, on this machine.

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.

Cpus number of CPUs on this machine.

LoadAvg Berkeley one-minute load average on machine.

Subnet TCP/IP subnet that this machine resides on.

Class set of available classes (those not currently in use) on this machine as defined in the configuration file.

Initiators set of all classes on this machine, both those in use and those not in use.

Feature set of features available on this machine.

Subnet TCP/IP subnet that this machine resides on.

Max_Jobs_Running
maximum number of jobs that any single user can run simultaneously 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.

time stamp date and time the central manager last received a status update from this machine.

llsubmit - Submit a Job

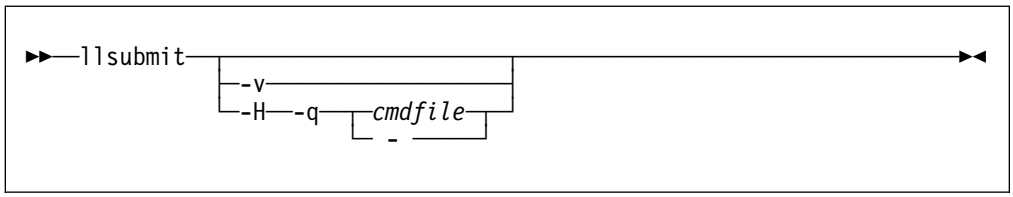
Purpose

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.

Users with **uid** or **gid** equal to 0 are not allowed to issue the **llsubmit** command.

Syntax



Parameters

Where

-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.

Examples

In this example, a command file named *qtrlyrun.cmd* is submitted.

```
llsubmit qtrlyrun
```

Appendix A. LoadLeveler Job Command File Keywords and Examples

This appendix provides an alphabetical list of the keywords you can use in a LoadLeveler script. It also provides examples you can follow along with to help you create your own scripts.

LoadLeveler Keywords

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.

The syntax is:

```
account_no = string
```

where `string` consists 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 program. Specify **yes** to checkpoint your program at regular intervals. In this case, your program must be linked with the LoadLeveler libraries. Otherwise, LoadLeveler will reject the program at submit time. Specify `no` if you do not want LoadLeveler to checkpoint the program. The default is `no`.

The syntax is:

```
checkpoint = yes | no
```

class

Specifies the name of a job class defined locally in your cluster. If not specified, the default job class is assigned.

The syntax is:

```
class =  name
```

For example, if you are allowed to submit jobs belonging to classes called *largejobs* and *weekendjobs*, your class keyword would be similar to either of the following:

```
class = largejobs
class = largejobs weekendjobs
```

comment

Describes characteristics or distinguishing features of the job.

core_limit

Specifies the maximum size of a core file.

When a job exceeds the softlimit it receives a signal. When a job reaches the hardlimit it is terminated. The softlimit signal allows the job time to take action, such as saving information and stopping gracefully, before reaching the hardlimit.

If no hardlimit is specified, then the limit set by the administrator in the class stanza of the administration file is used. If no limit is set in the administration file, then the limit set on the machine that runs the job is used. If the resulting hardlimit is less than the specified softlimit, then the softlimit is set to the hardlimit. A specified hardlimit must be greater than or equal to the specified softlimit and less than or equal to the limit set by the administrator in the class stanza of the administration file. If a specified hardlimit is greater than the limit on the executing machine, then the hardlimit is set to the machine's limit.

Express the size for hardlimit and softlimit as:

```
  integer{.fraction}{units}
```

where integer and fraction represent strings of up to eight digits. The units may be:

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

If you do not specify units, bytes are assumed. Alternatively, the value may be either of the following strings:

rlim_infinity	indicates the largest positive number
unlimited	same as rlim_infinity
copy	indicates that the limit currently active when the job is submitted should be used.

The syntax is:

```
core_limit = hardlimit,softlimit
```

For example,

```
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
```

cpu_limit

Specifies the maximum amount of CPU time that a submitted job can use. See "core_limit" on page 52 for an explanation of hardlimit and softlimit. Express the size as:

```
{{hours:}minutes:}seconds{.fraction}
```

If you specify fraction, it is rounded to seconds. Alternatively, you can express the value as one of the following strings:

rlim_infinity indicates the largest positive number

unlimited same as rlim_infinity

copy indicates that the limit currently active when the job is submitted should be used.

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
```

data_limit

Specifies the maximum size of the data segment to be used by the submitted job. See "core_limit" on page 52 for an explanation of these values.

The syntax is:

```
data_limit = hardlimit,softlimit
```

Note the examples in the following table:

Example	Explaination
data_limit = ,125621	125,621 bytes soft limit
data_limit = 5621kb	5,755,904 bytes hard limit
data_limit = 2mb	2,097,152 bytes hard limit
data_limit = 2.5mw,2mb	10,485,760 bytes hard limit, 2mb soft limit

environment

Specifies your initial environment variables when your job starts. Separate environment specifications with semicolons. An environment specification may be one of:

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 ; ...
```

error

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

The syntax is:

```
error = filename
```

executable

For serial jobs, identifies the name of the program to run. Include one executable keyword in a job command file. 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
```

file_limit

Specifies the maximum size of the file. See "core_limit" on page 52 for an explanation of these values.

The syntax is:

```
file_limit = hardlimit,softlimit
```

hold

Specifies whether you want to place a hold on your program when you submit it. There are three types of holds:

user specifies user hold

system specifies system hold

usersys specifies user and system hold

The syntax is:

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

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

```
hold = user
```

image_size

Maximum virtual image size, in kilobytes, to which your program will grow during execution. LoadLeveler tries to execute your job on a machine that has enough resources to support executing and checkpointing your job. If you do not specify the image size of your job command file, LoadLeveler assumes its size. If you underestimate the image size of your job, your job 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 meg, the keyword statement would be:

```
image_size = 11meg
```

initialdir

The pathname of the directory to use as the initial working directory during execution of the job. 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.

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

The syntax is:

```
input = filename
```

For example:

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

notification

Specifies when the user specified in notify_user is sent mail. The options are:

always	Notify you when the job begins, ends, or if it incurs error conditions.
error	Notify you only if the job fails.
start	Notify you only when the job begins.

never Never notify you.

complete Notify you only when 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 completes, your notification keyword would be:

```
notification = complete
```

notify_user

Specifies the user to whom mail is sent based on the **notification** keyword. If not specified, the job owner receives the mail.

The syntax is:

```
notify_user = userid
```

For example, if you are the job owner but you want a co-worker whose name and userid are Bob to receive mail regarding the job, your notify keyword would be:

```
notify_user = bob
```

output

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

The syntax is:

```
output = filename
```

For example:

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

parallelmachines

Specifies the number of nodes you want allocated to run the parallel job. The default number is zero, implying that the job is serial. If not specified, LoadLeveler assumes the job is serial.

The syntax is:

```
parallelmachines = n
```

where n is a number greater than or equal to zero.

Only use this keyword when submitting parallel jobs created using the IBM AIX Parallel Environment application programming language to the SP1. For more information on creating the parallel programs for the SP1, refer to the *IBM AIX Parallel Programming Reference* or the *IBM AIX Parallel Environment Operation and Use Guide*. Also, when using this keyword, do not specify the **executable** keyword.

For example, if your parallel job requires 5 parallel machines to run, your parallelmachines keyword would be:

```
parallelmachines = 5
```

preferences

List of characteristics that you prefer be available on the remote machine executing jobs 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. See “requirements” for a description of the characteristics that can be specified.

The syntax is:

```
preferences = Boolean expression
```

For example,

```
preferences = (Memory <=16) && (Arch == "R6000")
```

or

```
preferences = Memory >= 64
```

queue

Places one copy of the job in the queue. If desired, you can specify input, output, error and argument statements between queue statements.

The syntax is:

```
queue
```

requirements

List of requirements which a remote machine must meet to execute any jobs in the cluster. The requirements supported are:

- Memory

  Specifies the amount of physical memory required in megabytes.

- Feature

  Specifies the name of a feature defined locally on your LoadLeveler cluster.

- Machine

  Specifies the name(s) of machines on which to run the jobs.

- Disk

  Specifies the amount of disk space in kilobytes you believe is required in the 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. The job must be compiled on a machine that matches these requirements.

- 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 job must be compiled on a machine that matches these requirements.

Specify multiple requirements in a single requirements statement.

The syntax is:

```
requirements = Boolean expression
```

For example,

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

restart

Restarts jobs that terminate before completing. If `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. See “core_limit” on page 52 for an explanation of these values.

The syntax is:

```
rss_limit = hardlimit,softlimit
```

shell

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

The syntax is:

```
shell = name
```

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

```
shell = /bin/ksh
```

stack_limit

Specifies the maximum size of the stack that is created. See “core_limit” on page 52 for an explanation of these values.

The syntax is:

```
stack_limit = hardlimit,softlimit
```

startdate

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

The syntax is:

```
startdate = date  time
date is MM/DD/YY
time is HH:mm(:ss)
```

For example, if you want the job to run on May 5th, 1994 and you don't care what time of day it runs, the keyword would be:

```
startdate = 05/23/94
```

user_priority

Sets the initial priority of your job. Priority only affects your jobs. It orders jobs you submitted with respect to other jobs submitted by you, not with respect to jobs submitted by other users.

The syntax is:

```
user_priority = number
```

where number is a number between 0 and 100, inclusive. A higher number indicates the job will be selected before a job with a lower number. The default priority is 50.

For example, assume you already submitted two jobs. The first job has a priority of 95 and the second job has a priority of 50. You want to submit a third job and you want it to run after the first job but before the second job. Your keyword statement might be:

```
user_priority = 75
```

This example assumes that the second job has not started to run yet.

Job Command File Syntax

The following general rules apply to job command files.

- Comments begin with a #.
- Statements begin with a # @. Note that there can be any number of blanks between the # and the @.
- 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.
- Keywords are case insensitive. This means you can enter them in lower case, upper case, or mixed case.

Job Command File Examples

The following are examples of job command files.

Example 1: Building and Submitting a Simple Job File

Assume you have a long running job that you usually start before you go home in the evening. You run this program from your home directory. The program reads an input file from your home directory and writes the results to a file in your home directory. You can run this job using LoadLeveler by creating the following script called *longjob.cmd*.

```
# The name of this script  ⟶ Comment line
# is longjob.cmd
#
# @ executable = longjob   ⟶ Defines executable code
# @ input = longjob.in     ⟶ Defines the input file
# @ output = longjob.out   ⟶ Defines the output file
# @ error = longjob.err    ⟶ Defines file to which to print errors
# @ queue                  ⟶ Places job in queue
```

After you create the script file, you can submit the file to LoadLeveler using the llsubmit command:

```
llsubmit longjob.cmd
```

When your job ends, you will receive mail containing exit status of your job and a summary of usage statistics.

Example 2: Submitting Multiple Jobs Files From a Single Script

Assume you want to run the executable *longjob* two times with two different sets of input. Rather than issuing the llsubmit command twice, you can repeat the *queue* command twice in the script called *longjob.cmd*. For example:

```
#   The name of this script ⟶  Comment line
#   is longjob.cmd
# @ executable = longjob    ⟶  Defines executable code
# @ input = longjob.in1     ⟶  Defines input file
# @ output = longjob.out1   ⟶  Defines output file
# @ error = longjob.err1    ⟶  Defines file to which to print errors
# @ queue                   ⟶  Places job in queue
# @ input = longjob.in2     ⟶  Defines input file
# @ output = longjob.out2   ⟶  Defines output file
# @ error = longjob.err2    ⟶  Defines file to which to print errors
# @ queue                   ⟶  Places job in queue
```

After you create the file, you can submit the file to LoadLeveler using the llsubmit command:

```
llsubmit longjob.cmd
```

LoadLeveler responds by issuing the following:

```
submit: 2 jobs have been submitted, "machine.24.0" through "machine.24.4"
```

When your job ends, you will receive mail containing exit status of your job and a summary of usage statistics.

Example 3: 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. 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.

This example illustrates the following:

- You can refer to the LoadLeveler name of your job symbolically, using $(Host), $(Cluster), and $(Process) in the LoadLeveler script file.
- $(Process) increases after each **queue** command. Therefore, you only need to specify input, output, error once to have LoadLeveler name these files correctly.
- $(Host) refers to the name of the machine.

Assume that you created five input files and each input file has different initial conditions for the program. This example takes advantage of the fact that the Process part of the job id starts at 0 and increases by one for each job in a Cluster.

```
%ls longjob.in.*
longjob.in.0
longjob.in.1
longjob.in.2
longjob.in.3
longjob.in.4
```

Submitting the following LoadLeveler script results in your program running five times, each time with a different input file. LoadLeveler generates the output file from the LoadLeveler job ids. This ensures that the results from the different runs are not jumbled. The job command file looks like:

```
%cat longjob.cmd
# @ executable = longjob
# @ input = longjob.in.$(Process)
# @ output = longjob.out$(Cluster).$(Process)
# @ error = longjob.err.$(Cluster).$(Process)
# @ queue
# @ queue
# @ queue
# @ queue
# @ queue
```

Now, type the command:

```
llsubmit longjob.cmd
```

LoadLeveler responds by issuing the following:

```
submit: 5 jobs have been submitted, "machine.23.0" through "machine.23.4"
```

The following table illustrates where the five jobs 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 4: Using the Job Command File as the Executable

This example uses a script file that does extensive preparation for a job and cleans up after. There are two ways to accomplish this. You can type the name of your shell script as the executable or you can follow the method illustrated here.

This example illustrates the following:

- The name of the script is *long.sh*. You can name your LoadLeveler script anything.
- The script does not contain the **executable** keyword. When you do not use this keyword, LoadLeveler assumes the script is the executable. You could specify *executable=long.sh* as the executable.
- Even though the shell uses *longjob.in* as the input to the program *longjob*, and writes output to *longjob.results*, we specified an output and error line for LoadLeveler. This is because the output and error files will contain standard out (stdout) and standard error (stderr) information for the shell, *long.sh*, not for the program *longjob*. You don't have to specify output and error files but without them, finding problems may be difficult.
- If you don't specify input, output, or error files, LoadLeveler connects the appropriate file descriptor to /dev/null.

Since statements in LoadLeveler scripts are preceded by shell comment characters, you can imbed LoadLeveler statements directly in your shell scripts. This eliminates the need to maintain separate files. LoadLeveler ignores all lines that do not start with # @.

You can execute the following script titled *long.sh* at your terminal or submit it directly to LoadLeveler with no changes:

```
$ chmod +x long.sh
$ cat long.sh
#!/bin/sh
### specify "queue" so LoadLeveler will accept it" ###
# @ output = long.$(Cluster).$(Process)
# @ error = long.err.$(Cluster).$(Process)
# @ queue
#
### Begin our job here ###
#
if [ -f longjob.results]
then
 rm -f longjob.results
fi
cat longjob.in | longjob > longjob.results
```

Type in the command:

```
llsubmit long.sh
```

LoadLeveler responds by issuing the following:

```
submit: 1 jobs has been submitted, "machine.24.0".
```

Example 5: Building and Submitting a Parallel Job File

In addition to building serial batch jobs, you can also build job command files to submit parallel batch jobs. Before constructing these job command files, consult your LoadLeveler system administrator to see if your installation is configured for parallel batch job submission.

To construct a parallel job command file, specify, either on the GUI or by using the text editor, the number of machines the parallel job needs in order to run. To do this, use the **parallelmachines** keyword described in Appendix A. If you do use this keyword, do not use the **executable** keyword. Also, parallel jobs cannot be checkpointed, so they should not be linked to the LoadLeveler checkpointing libraries, and the **checkpoint** keyword should either be set to no or not used at all (default value is no). Following is an example of a parallel job called *my_parallel_job* requiring five parallel nodes to run:

```
#
# @ input = job.in
# @ output = job.out
# @ error = job.err
# @ parallelmachines = 5
# @ checkpoint = no
# @ queue
    my_parallel_job
```

Multiple parallel jobs can be submitted using one job description file. For example, if you want to submit three parallel jobs, *my_parallel_job1*, *my_parallel_job2*, and *my_parallel_job3*, requiring 3, 10, and 4 parallel machines respectively, a sample jobs description file would be:

```
#
# @ input = job.in
# @ output = job.out
# @ error = job.err
# @ parallelmachines = 10
# @ checkpoint = no
# @ queue
    my_parallel_job1
    my_parallel_job2
    my_parallel_job3
```

Notice that the **parallelmachines** keyword is set to the maximum number of parallel machines needed among the three jobs. The three jobs will execute sequentially. This means that *my_parallel_job1* will complete and then *my_parallel_job2* begins and so on.

If *my_parallel_job* is a Parallel Environment (PE) job, it is specified as:

```
MP_PROCS =< number of parallel machines needed> PE_parallel_program
```

Also, ensure that there is no *host.list* file in the directory that the parallel job is to be run because LoadLeveler does not support targeting specific SP1 parallel nodes for PE jobs.

Refer to the *IBM AIX Parallel Environment Operation and Use Guide* for more information on compiling, linking, and executing PE jobs.

Appendix B. Graphical User Interface Information

This appendix provides some introductory information on graphical user interfaces. This chapter does not provide complete nor detailed instructions on using either the LoadLeveler user interface or any other graphical user interface.

This appendix also discusses how to customize your graphical user interface by modifying the **Xloadl** file.

A graphical user interface usually contains windows, menus, pull-down menus, and dialog boxes. For detailed information on parts of a graphical user interface, refer to the appropriate Motif documentation.

What is a Window?

A *window* is a division of a screen in which you can run programs, display information, and receive feedback such as processing status. In general, a window is that part of the screen through which you and an application communicate. This section discusses the types of windows and the layout of a window.

What are the Types of Windows?

There are two types of windows: primary windows and secondary windows.

Primary Windows

A primary window is the window in which the main interaction between you and an application takes place. You can move and resize primary windows. Within a primary window are choices of actions specific to an application. You can view the primary window as the application's “main window”, to which you can always return. If you close a primary window, all secondary windows associated with that window disappear. Closing the last primary window ends the application session.

Secondary Windows

A secondary window is a window of short duration, such as a dialog box. Usually secondary windows appear only long enough to show some information or to obtain some input from you.

A secondary window is always related to a parent window, which may be either a primary window or another secondary window. Secondary windows extend the dialog presented in the parent window, providing more information you may need to complete processing. A Help window is an example of a secondary window. It provides additional information that can help you complete your task.

Secondary windows disappear when you complete the dialog. You can move secondary windows and you may or may not be able to resize them. If you minimize a primary window, its secondary windows are temporarily removed from the display. If you close a primary window, its secondary windows go away.

What is the Layout of a Window?

A window usually has a border that goes around the window, a title bar at the top of the window, a menu bar underneath the title bar, and *minimize* and *maximize* buttons. You can use the border to resize the window by clicking on the border with the mouse and dragging the border to a new location. The title bar across the top of the window displays the title of the window. The menu bar appears below the title bar and displays the actions you can perform using the window. The *minimize* and *maximize* buttons let you iconify the window or fill the entire display area with the window, respectively. The remaining portion of the window is the client area. On a primary window the menu bar contains the application's actions, across the top of the client area. On a dialog box, push-buttons across the bottom of the client area present the application's actions. For more specific information on the layout of a window or a window's components, refer to appropriate Motif documentation.

What is a Dialog Box?

A dialog box is a movable window, fixed in size, that contains controls such as push-buttons. You use these controls to provide information to the application.

The push-buttons are arranged across the bottom of the dialog box. There are a number of common dialog box actions. Following is a list of these actions, in the approximate order they appear (from left to right). Not all actions appear in all dialog boxes.

Yes	Indicates an affirmative response to a question, and closes the dialog box.
No	Indicates a negative response to a question, and closes the dialog box.
OK	Changes the current environment variables to the values you have chosen in the dialog box. When you press **OK**, the dialog box closes.
Stop	Ends the task in progress at the next possible breaking point.
Save	Stores any variables you have modified in the dialog box in the current user preference profile.
Cancel	Closes the dialog box without accepting any of the variable selections you have made on the dialog box.
Close	Removes the dialog box.
Help	Provides help information.
Submit	Submits the job to LoadLeveler for execution.
Filter	Applies the file selection filter and updates the Files selection area.

How Do I Enter Data in the Interface?

Many of the LoadLeveler windows and dialog boxes employ the following ways to enter data. These ways are:

Text entry fields
: A field you can select and type directly into.

Radio buttons
A circle with text beside it. Radio buttons are combined to show a user a fixed set of choices from which they can only select one.

Check boxes
A square box with associated text that represents a choice. When a user selects a choice, an X appears in the check box to indicate that the choice is in effect.

What is a Menu Bar?

The menu bar is a horizontal list of menu topics that appear below the title area in the primary window. When you select a menu topic, a pull-down menu appears. When you select an item from a pull-down menu, other actions can occur that may result in further pull-down menus, dialog boxes, or other controls to appear. Selecting a menu topic from the menu bar does not cause an action to occur; selecting an item from a pull-down menu causes an action to occur. Therefore, you can browse a menu bar and examine the pull-down menus associated with each menu topic without causing unintended actions to occur.

What is a Pull-down Menu?

A pull-down menu is a menu that emerges in a downward direction from a point or line at or near the top of the screen. 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.

Items in pull-down menus have one of three possible things following them. The following table illustrates:

Item following	Example	Effect
Nothing	Refresh All	Causes an action to occur.
Ellipses (...)	Build Job ...	Causes a dialog box or secondary window to appear.
Right pointing arrow (►)	By User ►	Causes another pull-down menu (cascading menu) to appear.

What is a Scroll Bar?

A scroll bar is a graphical device used to change a user's view point of a list or data file. A scroll bar consists of a slider, scroll area, and scroll arrows. A user changes the view by sliding the slider up or down in the scroll area or by pressing one of the scroll arrows. This causes the view to scroll up or down in the window adjacent to the scroll bar.

Customizing the Graphical User Interface

You can customize the graphical user interface by modifying the Xloadl file in **/usr/lib/X11/app-defaults** directory. This file contains the default values for the graphical user interface. Following is a brief description of the syntax of an Xloadl file. Most of the resource names in the file are common to all AIX or UNIX Xloadl files.

Syntax of an Xloadl File

- Comments begin with !
- Resource variables 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 graphical user interface 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:

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

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.
label.string	The text associated with the object.

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.		

Modifying Application Defined Resources

Resource Variable	Values	Description
editCommand	Available editors	Command to use when you select “edit” from the Submit window.
prompt	[true|false] The default is True.	Specifies whether you are prompted for confirmation of actions.
job_sort	Current sort is: name time usage priority machine	Specifies how the jobs are sorted in the Jobs window.
job_refresh	Any integer. Default is 0.	Specifies the length of time between automatic updates of the Jobs Window.
job_select_machine	Any valid hostname.	Host name to use for job filtering. When job filter is set as “allmachine”, only jobs from this machine are shown.
job_select_name	Any valid userid. The default is your userid.	User name to use for job filtering When job filter is set as “alluser”, only jobs from this user are shown. If job_select_user is not specified, the default is the userid from which xloadl was started.
machine sort	Currest sort is: name running total priority state ldavg idle arch opsys.	Specifies how the machines are sorted in the Machines window.
machine_refresh	Integer. The default is 60 seconds.	Specifies the length of time between automatic updates of the Machines window.
logging	[true|false] The default is false.	Specifies whether to automatically log messages to a file.
logfile	Characters and/or Integers. The default file is `xloadl.log`	Specifies the file name to which messages are logged.
snapfile	Characters and/or Integers. The default file is `xloadl.snapshot`	Name of the file to use when requesting a snapshot of the log.

Resource Variable	Values	Description
skelfile	Characters and/or Integers	Name of the file to use as a template for the Job Builder window.
job filter	Current filters are: running by user, idle by user, all by user, running by host, idle by host, all by host, all running, all idle, and all jobs.	If you select any of the “by_user” filters, you may also specify which user in the “job_select_name” resource. If you select any of the “by_host” filters, you may also specify which user in the “job_select_machine” resource.

Modifying the Titles of the Main Windows

To change the titles of any of the windows, locate the correct resource variable and modify it using any combination of characters and/or integers.

Resource Variable	Description
job_status*title*labelString	Title of the Jobs Window.
machine_status*title*labelString	Title of the Machines Window.
message_area*title*labelString	Title of the Messages Window.

Modifying the Labels in the Build a Job Window

To change the resource variables (labels) in the Build a Job window, locate the correct resource variable and modify it using any combination of characters and/or integers.

Resource Variable	Current Value
builder*work_area*Executable.labelString	Executable
builder*work_area*Stdin.labelString	Stdin
builder*work_area*Stdout.labelString	Stdout
builder*work_area*Stderr.labelString	Stderr
builder*work_area*NotifyUser.labelString	Notify user
builder*work_area*StartDate.labelString	Start Date mm/dd/yy
builder*work_area*StartTime.labelString	Start Time hh:mm:ss.
builder*work_area*Priority.labelString	Priority
builder*work_area*ImageSize.labelString	Image Size
builder*work_area*Class.labelString	Class
builder*work_area*Notification.labelString	Notification
builder*work_area*Syscalls.labelString	Syscalls
builder*work_area*Checkpoint.labelString	Checkpoint
builder*work_area*Syscalls_Local.labelString	Local
builder*work_area*Syscalls_Remote.labelString	Remote
builder*work_area*Checkpoint_Yes.labelString	Yes
builder*work_area*Checkpoint_No.labelString	No

Resource Variable	Current Value
builder*work_area*Notify_Always.labelString	Always
builder*work_area*Notify_Complete.labelString	Complete
builder*work_area*Notify_Error.labelString	Error
builder*work_area*Notify_Never.labelString	Never
builder*work_area*Notify_Start.labelString	Start
builder*action_area*Submit.labelString	Submit
builder*action_area*Save.labelString	Save
builder*action_area*Defaults.labelString	Defaults
builder*action_area*Clear.labelString	Clear
builder*action_area*Close.labelString	Close
builder*action_area*Help.labelString	Help

Modifying the Labels in the Requirements Window

To change the resource variables (labels) in the Requirements window, locate the correct resource variable and modify it using any combination of characters and/or integers.

Resource Variable	Current Value
requirements*work_area*Architecture.labelString	Architecture
requirements*work_area*OpSys.labelString	Operating Sys
requirements*work_area*Disk.labelString	Disk
requirements*work_area*Memory.labelString	Memory
requirements*work_area*Machines.labelString	Machines
requirements*work_area*Features.labelString	Features
builder*requirements*action_area*Help.labelString	Help
builder*requirements*action_area*Close.labelString	Close

Modifying the Labels in the Preferences Window

To change the resource variables (labels) in the Preferences window, locate the correct resource variable and modify it using any combination of characters and/or integers.

Resource Variable	Current Value
preferences*work_area*Architecture.labelString	Architecture
preferences*work_area*OpSys.labelString	Operating Sys
preferences*work_area*Disk.labelString	Disk
preferences*work_area*Memory.labelString	Memory
preferences*work_area*Machines.labelString	Machines
preferences*work_area*Features.labelString	Features
builder*preferences*action_area*Help.labelString	Help
builder*preferences*action_area*Close.labelString	Close

Modifying the Labels in the Limits Window

To change the resource variables (labels) in the Limits window, locate the correct resource variable (label) and modify it using any combination of characters and/or integers.

Resource Variable	Current Value
limits*work_area*cpu_limit.labelString	Cpu Limit
limits*work_area*rss_limit.labelString	Rss Limit
limits*work_area*core_limit.labelString	Core Limit
limits*work_area*file_limit.labelString	File Limit
limits*work_area*stack_limit.labelString	Stack Limit
limits*work_area*data_limit.labelString	Data Limit
builder*limits*action_area*Help.labelString	Help
builder*limits*action_area*Close.labelString	Close

Modifying the Labels in the Account Report Data Window

To change the resource variables (labels) in the Account Report Data window, locate the correct resource variable (label) and modify it using any combination of characters and/or integers.

Resource Variable	Current Value
reporter*work_area*Format.labelString	Format
reporter*work_area*Start.labelString	Start date
reporter*work_area*End.labelString	End date
reporter*work_area*Short.labelString	Short
reporter*work_area*Long.labelString	Long
reporter*work_area*Input.labelString	Input data file
reporter*work_area*Output.labelString	Output to file

Modifying the Jobs Window Pull-Down Menu Bar Labels

The following table lists the resource variables (labels) on the pull-down menus of the Jobs window. You can change the resourece variables on these pull-down menus by modifying them using any combination of characters and/or integers but it is not recommended that you make any modifications unless you are very familiar with resource files.

Resource Variable	Current Value
The following resource variables are for the File pull-down menu	
job_status*File_Close_All.labelString	Close All
job_status*File_Minimize_All.labelString	Minimize All
job_status*File_Machines.labelString	Machines...
job_status*File_Messages.labelString	Messages...
job_status*menubar*File.labelString	File
job_status*File_Build_Job.labelString	Build Job...
job_status*File_Submit_Job.labelString	Submit Job...
job_status*File_sep.labelString	sep
job_status*File_Exit.labelString	Exit
The following resource variables are for the Actions pull-down menu	
job_status*menubar*Actions.labelString	Actions
job_status*Actions_Cancel.labelString	Cancel
job_status*Actions_Priority.labelString	Priority
job_status*Actions_Hold.labelString	Hold
job_status*Actions_Release.labelString	Release From Hold
job_status*Actions_Details.labelString	Details
job_status*Actions_sep.labelString	sep
job_status*Actions_Select_All.labelString	Select All
job_status*Actions_Deselect_All.labelString	Deselect All

Resource Variable	Current Value
The following resource variables are for the Refresh pull-down menu	
job_status*menubar*Refresh.labelString	Refresh
job_status*Refresh_Jobs.labelString	Refresh Jobs
job_status*Refresh_Auto.labelString	Set Auto Refresh...
job_status*Refresh_sep.labelString	sep
job_status*Refresh_All.labelString	Refresh All
The following resource variables are for the Sort pull-down menu	
job_status*menubar*Sort.labelString	Sort
job_status*Sort_by_Name.labelString	Sort by Name
job_status*Sort_by_Class.labelString	Sort by Class
job_status*Sort_by_Submission_Time.labelString	Sort by Submission Time
job_status*Sort_by_Priority.labelString	Sort by Priority
job_status*Sort_by_Machine.labelString	Sort by Machine
job_status*Sort_by_Dispatch_Order.labelString	Sort by Dispatch Order
The following resource variables are for the Select pull-down menu	
job_status*menubar*Select.labelString	Select
job_status*Select_All.labelString	All
job_status*Select_All_Running.labelString	Jobs Running
job_status*Select_All_Idle.labelString	Jobs Idle
job_status*Select_All_Jobs.labelString	All Jobs
job_status*Select_User.labelString	By User
job_status*Select_User_Running.labelString	Jobs Running...
job_status*Select_User_Idle.labelString	Jobs Idle...
job_status*Select_User_Jobs.labelString	All Jobs...
job_status*Select_Machines.labelString	By Machine
job_status*Select_Machines_Running.labelString	Running Jobs Submitted To...
job_status*Select_Machines_Idle.labelString	Idle Jobs Submitted To...
job_status*Select_Machines_All.labelString	All Jobs Submitted To...
job_status*Select_Job_Id.labelString	By Job Id...
job_status*Select_Show.labelString	Show Selection
The following resource variables are for the Admin pull-down menu	
job_status*menubar*Admin.labelString	Admin
job_status*Admin_Favor_User.labelString	Favor User...
job_status*Admin_End_Favor_User.labelString	UnFavor User...
job_status*Admin_Favor_Job.labelString	Favor Job
job_status*Admin_End_Favor_Job.labelString	UnFavor Job
job_status*Admin_Set_SysHold.labelString	Syshold
job_status*Admin_Release.labelString	Release From Hold
The following resource variables are for the Help pull-down menu	

Resource Variable	Current Value
job_status*menubar*Help.labelString	Help
job_status*Help_File.labelString	File
job_status*Help_Actions.labelString	Actions
job_status*Help_Overview.labelString	Overview
job_status*Help_Windows.labelString	Windows
job_status*Help_Refresh.labelString	Refresh
job_status*Help_Sort.labelString	Sort
job_status*Help_Select.labelString	Select
job_status*Help_Admin.labelString	Admin

Modifying the Machines Window Pull-Down Menu Bar Labels

The following table lists the resource variables (labels) on the pull-down menus of the Machines window. You can change the resource variables on these pull-down menus by modifying them using any combination of characters and/or integers but it is not recommended that you make any modifications unless you are very familiar with resource files.

Resource Variable	Current Value
The following resource variables are for the File pull-down menu	
machine_status.menubar.File.labelString	File
machine_status*File_Build_Job.labelString	Build Job...
machine_status*File_Submit_Job.labelString	Submit Job...
machine_status*File_sep.labelString	sep
machine_status*File_Exit.labelString	Exit
The following resource variables are for the Actions pull-down menu	
machine_status*menubar*Actions.labelString	Actions
machine_status*Actions_Details.labelString	Details
machine_status*Actions_Find_Central_Manager.labelString	Find Central Manager
machine_status*Actions_Find_LoadL_Clients.labelString	Find Public Submit Machines
machine_status*Actions_sep.labelString	sep
machine_status*Actions_SelectAll.labelString	Select All
machine_status*Actions_DeselectAll.labelString	Deselect All
The following resource variables are for the Refresh pull-down menu	
machine_status*menubar*Refresh.labelString	Refresh
machine_status*Refresh_Machines.labelString	Refresh Machines
machine_status*Refresh_Auto.labelString	Auto Refresh...
machine_status*Refresh_sep.labelString	sep
machine_status*Refresh_All.labelString	Refresh All
The following resource variables are for the Sort pull-down menu	
machine_status*menubar*Sort.labelString	Sort

Resource Variable	Current Value
machine_status*Sort_by_Name.labelString	Sort by Name
machine_status*Sort_by_Run.labelString	Sort by Run
machine_status*Sort_by_Total.labelString	Sort by Total
machine_status*Sort_by_State.labelString	Sort by State
machine_status*Sort_by_LdAvg.labelString	Sort by LdAvg
machine_status*Sort_by_Idle.labelString	Sort by Idle
machine_status*Sort_by_Arch.labelString	Sort by Arch
machine_status*Sort_by_OpSys.labelString	Sort by OpSys
machine_status*Sort_NoSort.labelString	No Sort
The following resource variables are for the Admin pull-down menu	
machine_status*menubar*Admin.labelString	Admin
machine_status*Admin_Start.labelString	Start LoadL
machine_status*Admin_Stop.labelString	Stop LoadL
machine_status*Admin_sep.labelString	sep
machine_status*Admin_reConfig.labelString	reConfig
machine_status*Admin_Recycle.labelString	Recycle
machine_status*Admin_sep.labelString	sep
machine_status*Admin_Drain.labelString	Drain
machine_status*Admin_Flush.labelString	Flush
machine_status*Admin_Suspend.labelString	Suspend
machine_status*Admin_Resume.labelString	Resume
machine_status*Admin_Reschedule.labelString	Reschedule
The following resource variables are for the Help pull-down menu	
machine_status*menubar*Help.labelString	Help
machine_status*Help_File.labelString	File
machine_status*Help_Overview.labelString	Overview
machine_status*Help_Actions.labelString	Actions
machine_status*Help_Refresh.labelString	Refresh
machine_status*Help_Sort.labelString	Sort
machine_status*Help_Admin.labelString	Admin

Modifying the Messages Window Pull-Down Menu Bar Labels

The following table lists the resource variables on the pull-down menus of the Messages window. You can change the resource variables on these pull-down menus by modifying them using any combination of characters and/or integers but it is not recommended that you make any modifications unless you are very familiar with resource files.

Resource Variable	Current Value
The following resource variables are for the File pull-down menu	
message_area*menubar*File.labelString	File
message_area*File_Jobs.labelString	Jobs...
message_area*File_Machines.labelString	Machines...
message_area*File_Minimize.labelString	Minimize
message_area*File_Close.labelString	Close
message_area*File_Build_Job.labelString	Build Job...
message_area*File_Submit_Job.labelString	Submit Job...
message_area*File_sep.labelString	sep
message_area*File_Exit.labelString	Exit
The following resource variables are for the Actions pull-down menu	
message_area*menubar*Actions.labelString	Actions
message_area*Actions_Clear.labelString	Clear Window
message_area*Actions_File.labelString	Start Logging to File...
message_area*Actions_End.labelString	End Logging
message_area*Actions_Snapshot.labelString	Snapshot Log...
The following resource variables are for the Help pull-down menu	
message_area*menubar*Help.labelString	Help
message_area*Help_File.labelString	File
message_area*Help_Actions.labelString	Actions

Appendix C. Checkpointing Jobs

"Checkpointing a Job" on page 7 provides a brief discussion of checkpointing. This appendix provides a more detailed discussion of checkpointing and provides instructions for checkpointing serial jobs on the RISC System/6000, the SP1, the Sun, and the SGI workstations.

Note

You cannot checkpoint parallel jobs so these jobs should not be linked to the LoadLeveler libraries.

Jobs requesting checkpointing that are not linked with the LoadLeveler libraries will be rejected when you submit them.

Some system calls have been changed to allow LoadLeveler to save certain state related information. For example, "open()" records all files used by the process so that at checkpoint time the current file pointer can be saved and then restored at restart time.

For some processes, it is impossible to obtain or recreate the state of the process. For this reason, you should only checkpoint jobs whose state is 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.

Any checkpointing job should not use the following system calls in order to prevent unpredictable results from occurring:

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

Another limitation of checkpointing jobs is file I/O. Since individual write calls are not traced, the file recovery scheme requires that all file updates be *idempotent*. In other words, 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.

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

The system administrator controls the frequency of checkpointing by specifying the **MIN_CKPT_INTERVAL** and **MAX_CKPT_INTERVAL** in seconds in the configuration file. The first checkpoint occurs at the **MIN_CKPT_INTERVAL**. For subsequent checkpoints the interval between successive checkpoints is doubled until the **MAX_CKPT_INTERVAL** is reached. For example, given **MIN_CKPT_INTERVAL=1000** and **MAX_CKPT_INTERVAL=7500**, checkpoints will be taken at the following intervals: 1000, 2000, 4000, 7500, 7500, ... The default **MIN_CKPT_INTERVAL** value is 900 (15 minutes). The default **MAX_CKPT_INTERVAL** value is 7200 (120 minutes). The system administrator can set these keyword values globally, so that all machines in the cluster have the same value, or he can specify a different value for each machine.

As discussed earlier, all of your checkpointing jobs must be linked with the LoadLeveler libraries and LoadLeveler checks to make sure the job is linked correctly. When submitting a job to a machine other than the RISC System/6000, LoadLeveler does not recognize the executable. This means that you can submit checkpointing jobs only to machines of the same type. For example, you can submit checkpointing jobs from a Sun to a Sun, from a RISC System/6000 to RISC System/6000, or from an SGI to another SGI machine.

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 relinking. 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 Sun

When submitting a checkpoint job to a Sun 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

```
ld -o a.out -dc -e start -Bstatic /usr/lpp/LoadL/sunosnfs/lib/loadl_rt0.o
a.o b.o c.o /usr/lpp/LoadL/sunosnfs/lib/libckpt.a
/usr/lpp/LoadL/sunosnfs/lib/libloadl.a /usr/lib/libc.a
```

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 crt0.o (loadl_rt0.o) and LoadLevler's libckpt.a and libloadl.a. You can find these in the directory **/usr/lpp/LoadL/nfssgi/lib**. Since UNIX 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:

/usr/bin/ld -o test01 -G 8 -g0 -nocount /usr/lpp/LoadL/sginfs/ lib/loadl_rt0.o -count test01.o -nocount /usr/lpp/LoadL/lib/libckpt.a /usr/lpp/LoadL/lib/libloadl.a -lrpcsvc -lsun -lc -lm /usr/lib/crtn.o

FORTRAN Example

Compiling:

f77 -c test01.f

Linking:

/usr/bin/ld -o test01 -Xnobsschange -G 8 -g0 -nocount /usr/lpp/LoadL/lib/loadl_rt0.o -count test01.o -nocount /usr/lpp/LoadL/lib/libckpt.a /usr/lpp/LoadL/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/hpfs/lib/libloadl.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/hpfs/lib/libloadl.a -lcl -lisamstub -lc

Glossary of Terms and Abbreviations

This glossary includes terms and definitions from:

- The *Dictionary of Computing*, SC20-1699.
- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies can be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018.
- The *ANSI/EIA Standard - 440A: Fiber Optic Terminology*, copyright 1989 by the Electronics Industries Association (EIA). Copies can be purchased from the Electronic Industries Association, 2001 Pennsylvania Avenue N.W., Washington, D.C. 20006.
- The *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1).

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) and *IBM DATABASE 2 Application Programming Guide for TSO Users* (SC26-4081).

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.

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.

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 Software Foundation (OSF). 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.

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

A
account_no 51
Andrew File System (AFS) 2
arguments 51

B
building jobs 5, 19

C
cancelling jobs 25, 32
  using the command llcancel 32
  using the GUI 25
central manager 3
changing job priority 24, 31, 40
  using the command llprio 31
  using the GUI 24, 40
checkpoint 51
checkpoint a job 7, 79
class 52
command syntax 34
comment 52
comment statements 5
core_limit 52
cpu_limit 53

D
data_limit 53
dialog boxes 14, 66
displaying 25, 26
  central manager 26
  machine status 25
  public scheduling machines 26
displaying job status 23, 31
  job status 23
  using the command llq 31
  using the GUI 23
displaying machine status 32
  using the command llstatus 32

E
editing jobs 7, 22
environment 54
error 54
executable 54

F
file_limit 54

G
graphical user interface 13, 15, 65
  customizing 15

H
hold 54
holding a job 31
  using the command llhold 31
holding jobs 11, 24
  using the GUI 24

I
IBM Scalable POWERparallel Systems 1
image_size 55
initialdir 55
input 55

J
job status 23, 42
jobs 19, 22, 23, 24, 25
  building 19
  cancelling 25
  changing job priority 24
  displaying job status 23
  editing 22
  holding 24
  releasing a hold 25
  submitting 23

K
keywords 51
  account_no 51
  arguments 51
  checkpoint 51
  class 52
  comment 52
  core_limit 52
  cpu_limit 53
  data_limit 53
  environment 54
  error 54
  executable 54
  file_limit 54
  hold 54

keywords *(continued)*
image_size 55
initialdir 55
input 55
notification 55
notify_user 56
output 56
parallelmachines 56
preferences 57
queue 57
requirements 57
restart 58
rss_limit 58
shell 58
stack_limit 58
startdate 58
user_priority 59

L

llcancel 36
llhold 38
llprio 40
llq 42
llstatus 46
llsubmit 50
LoadLeveler 1
definition 1
NQS compatibility 1

M

machine status 46
menu bar 15, 67
messages 27

N

Network File System (NFS) 2
notification 55
notify_user 56
NQS 1

O

output 56

P

parallelmachines 56
preferences 57
primary windows 65
priority 40
public scheduling machines 12
pull-down menus 15, 67

Q

querying a job 10, 42
queue 57

R

requirements 57
restart 58
rss_limit 58

S

scroll bars 67
secondary windows 14, 65
SGI 79
shell 58
stack_limit 58
startdate 58
status 46
submit-only machine 10, 11
submitting jobs 8, 23, 30, 50
using the command llsubmit 30
using the GUI 23
Sun 79

U

user interface 18
user_priority 59

W

windows 13, 14, 65, 66
layout of 66
types of
primary 65
secondary 14, 65

X

Xloadl 67

IBM®

Printed in U.S.A.

SH26-7226-02