IBM LoadLeveler Administration Guide
Release 2.1

Document Number SH26-7220-03

IBM LoadLeveler Administration Guide
Release 2.1

SH26-7220-03

IBM

IBM LoadLeveler Administration Guide Release 2.1

SH26-7220-03

Note

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

Third Edition, August 1995

This edition applies to Version 1 Release 2 Modification 1 of IBM LoadLeveler, (5765-145 for RISC System/6000 and RISC System/6000 Scalable POWERparallel Systems), (5765-227 for Sun), (5765-228 for SGI), (5765-287 for HP). This edition applies to subsequent releases and modifications until otherwise indicated in new editions.

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

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

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

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

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

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

Copyright International Business Machines Corporation 1993, 1994, 1995. 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 material — Property of IBM. Copyright 1986, 1987, 1988, 1989, 1990, 1991 by the Condor Design Team.

Contents

Notices . . . vi
Trademarks and Service Marks . . . vi
About This Book . . . vii
Who Should Use This Book . . . vii
How to Use This Book . . . vii
Overview of Contents . . . vii
Terms and Document Conventions . . . viii
Related Publications . . . viii
Ordering Additional Copies of This Book . . . ix
Viewing Online InfoExplorer Documentation . . . ix
Printing Documentation . . . ix
List of Program Material . . . x
Statement of Program Support . . . x
Summary of New Functions in Release 2 Refresh . . . x
AIX 4.1 Support . . . x
New User Exits . . . x
Enhanced Group, User, and Class Support . . . xi
New and Enhanced Commands . . . xi
Enhanced Starter Functions . . . xii
Enhanced Scheduler Functions . . . xii
Error Logging Enhancements . . . xii
Migration Consideration . . . xii

Chapter 1. What is LoadLeveler? . . . 1
How LoadLeveler Works . . . 2

Chapter 2. Administering and Configuring LoadLeveler . . . 15
Planning Considerations . . . 15
Where to Begin? . . . 16
Quick Set Up . . . 17
Administration File Structure and Syntax . . . 18
Customizing the Administration File . . . 20
Configuring LoadLeveler . . . 34
Configuration File Structure and Syntax . . . 35
Customizing the Global and Local Configuration Files . . . 41
Additional Administration Topics . . . 61
Summary of Process Specific Keywords . . . 64
Summary of Administration File Keywords . . . 69
Summary of Configuration File Keywords and Macros . . . 70

Chapter 3. Submitting Parallel Jobs . . . 77
Setting Up to Allow Users to Submit Parallel Jobs . . . 79
Setting Up to Allow Users to Submit Parallel Jobs to the SP System . . . 80
Setting Up to Allow Users to Submit PVM Jobs . . . 80
Sample Parallel Job Submission Scenarios . . . 82

Chapter 4. Interactive Login Sessions . . . 89
Using Interactive Session Support . . . 89
Troubleshooting . . . 102
ISS Configuration File . . . 105

© Copyright IBM Corp. 1993, 1995

Chapter 5. Gathering Job Accounting Data 107
Collecting Job Resource Data on Serial and Parallel Jobs 107
Collecting Job Resource Data Based on Machines 107
Collecting Job Resource Data Based on Events 108
Collecting Job Resource Information Based on User Accounts 108
Collecting the Accounting Information and Storing it into Files 109
Accounting Reports 109
Sample Job Accounting Scenario 110

Chapter 6. Error Logging Facility 113
Error Logging Overview 113

Chapter 7. Routing Jobs to NQS Machines 121
Setting Up the NQS Environment 121
Designating Machines to Which Jobs Will be Routed 122
Sample Routing Jobs to NQS Machines Scenario 122
NQS Scripts 125

Chapter 8. LoadLeveler Commands 127
llacctmrg - Collects machine history files 128
llcancel - Cancel a Submitted Job 130
llclass - Query Class Information 132
llctl - LoadLeveler Control 134
llfavorjob - Reorder System Queue by Job 137
llfavoruser - Reorder System Queue by User 138
llhold - Hold or Release a Submitted Job 139
llinit - Initializes machines in the LoadLeveler Pool 141
llprio - Changes the Priority of Submitted Job Steps 143
llq - Query Job Status 145
llstatus - Query Machine Status 151
llsubmit - Submit a Job 156
llsummary - Returns job resource information 157

Chapter 9. Administrative Uses for the Graphical User Interface 165
Job Related Administrative Actions 167
Machine Related Administrative Actions 168

Appendix A. Commonly Asked Questions and Answers 173
Why Won't My Job Run? 173
What Happens to Running Jobs When the Machine Goes Down? 175
Why Won't LoadLeveler Start? 176
What Happens if the Central Manager Isn't Operating? 176
How Do I Move the Central Manager Function to Another Machine? 178
Why Does llstatus Indicate that a Machine is Down but llq Indicates a Job is Running on The Machine? 178
Why do I have to setuid = 0? 178
How Do I Determine When My Job Started and Stopped? 179
Why Doesn't LoadLeveler Execute my .profile or .login Script? 179
Can I Have Multiple Administration and Global Configuration Files? 180
Messages 180
Getting Help from IBM 181

Appendix B. LoadLeveler Job Command File Keywords 183
Building a Job Command File 183

LoadLeveler Keywords 183
Job Command File Macros 196

Appendix C. Sample Files 199
Sample Configuration Files 199
Administration Files 206

Appendix D. LoadLeveler Daemons and Processes 213
The Master Daemon 213
The Schedd Daemon 214
The Startd Daemon 215
The Starter Process 216
The Collector Daemon 217
The Kbdd Daemon 217
The Negotiator Daemon 217

Appendix E. LoadLeveler's Application Programming Interfaces 219
Accounting APIs 219
Prolog and Epilog APIs 222
Checkpointing API 228
The Submit API 229
Parallel Job API 231
User Exits 236

Glossary 239

Index 243

Notices

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

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

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

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

IBM Corporation
Mail Station P131
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Request

Trademarks and Service Marks

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

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

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

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

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

About This Book

This manual is designed to help you install and administer the LoadLeveler program product. It describes how to tune LoadLeveler for increased efficiency in your environment and explains the various commands and files provided for use in the configuration process.

This version of the book describes LoadLeveler Version 1.2.1. Changes made since the previous version of the book are indicated by a vertical bar (|) to the left of each change.

Who Should Use This Book

This manual is intended for those who are responsible for LoadLeveler system administration. The role of the system administrator normally includes the following tasks:

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

In order to perform the tasks described in this manual, the LoadLeveler system administrator must be an experienced user of UNIX** commands and be familiar with system management techniques such as SMIT, used in the AIX* environment. Knowledge of networking and NFS** or AFS** protocols will also prove helpful.

How to Use This Book

Overview of Contents

- Chapter 1, “What is LoadLeveler?” on page 1 describes LoadLeveler, its components, and how they work.
- Chapter 2, “Administering and Configuring LoadLeveler” on page 15 describes how to modify the administration and configuration files in order to customize LoadLeveler for site specific requirements.
- Chapter 3, “Submitting Parallel Jobs” on page 77 describes how to schedule batch parallel jobs.
- Chapter 4, “Interactive Login Sessions” on page 89 describes how to achieve interactive login sessions.
- Chapter 5, “Gathering Job Accounting Data” on page 107 describes how to gather accounting data on jobs.
- Chapter 6, “Error Logging Facility” on page 113 describes the error logging functions available when LoadLeveler runs on AIX 4.1.
- Chapter 7, “Routing Jobs to NQS Machines” on page 121 describes LoadLeveler's role in routing NQS jobs to NQS machines.
- Chapter 8, “LoadLeveler Commands” on page 127 explains the LoadLeveler commands and describes the inputs and requirements for each.

- Chapter 9, “Administrative Uses for the Graphical User Interface” on page 165 describes the tasks that the administrator can perform using the graphical user interface.
- Appendix A, “Commonly Asked Questions and Answers” on page 173 lists common problems and solutions.
- Appendix B, “LoadLeveler Job Command File Keywords” on page 183 describes the various keywords you can define when you submit a LoadLeveler job.
- Appendix C, “Sample Files” on page 199 contains sample configuration and administration files.
- Appendix D, “LoadLeveler Daemons and Processes” on page 213 describes the LoadLeveler program components.
- Appendix E, “LoadLeveler's Application Programming Interfaces” on page 219 describes LoadLeveler's application programming interfaces.

A glossary and index are also included.

Terms and Document Conventions

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

The term **SP** is used to refer to the IBM RS/6000 Scalable POWERparallel System.

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.

Related Publications

IBM LoadLeveler: User's Guide, SH26-7226

IBM LoadLeveler Technical Presentation, ZZ81-0348-00

LoadLeveler Memo To Users, included on the product media.

Ordering Additional Copies of This Book

To order additional copies of this book, request form number SH26-7220.

Viewing Online InfoExplorer Documentation

The LoadLeveler Administration Guide and the LoadLeveler User's Guide are available in InfoExplorer format for AIX customers only. You can obtain the InfoExplorer documentation in two ways:

- You can obtain the InfoExplorer documentation by doing an anonymous FTP (File Transfer Program) to the following machine address:

  `lscftp.kgn.ibm.com (129.40.10.220)`

  The InfoExplorer information can be found in the directory called **pub/pps/loadleveler**.

- You can obtain the InfoExplorer documentation by selecting the InfoExplorer documentation option during the installp process.

Once you receive the information, you can access the InfoExplorer documentation by typing:

```
info -l directory_path
```

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

Printing Documentation

A PostScript file for each of the LoadLeveler books is available for you to print on a PostScript printer. The file names are:

lladmin.ps The LoadLeveler Administration Guide.

lluser.ps The LoadLeveler User's Guide.

You can obtain the PostScript documentation in two ways:

- You can obtain the PostScript documentation by doing an anonymous FTP (File Transfer Program) to the following machine address:

  `lscftp.kgn.ibm.com (129.40.10.220)`

  The PostScript files can be found in the directory called **pub/pps/loadleveler**. In this directory, you will also find a file called **legal.readme** that contains more information regarding these PostScript files.

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

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

```
tr -d '\r' < input_file.ps > output_file.ps
```

List of Program Material

The distribution material contains all of the basic machine-readable material for LoadLeveler. There are no optional machine-readable materials.

The following is the list of basic program publications for LoadLeveler:

- IBM LoadLeveler: Administration Guide (SH26-7220)
- IBM LoadLeveler: User's Guide (SH26-7226)
- IBM LoadLeveler: Licensed Program Specification for the IBM RISC System/6000 Scalable POWERparallel Systems SP and the RISC System/6000 Family, SPARCstation Systems, Hewlett-Packard Systems, and Silicon Graphics Systems (GH23-0040-01)

Statement of Program Support

The distribution material contains the most complete and accurate code and examples.

If you encounter problems when using LoadLeveler, you should consult Appendix A, “Commonly Asked Questions and Answers” on page 173 for procedures to isolate your problem and report any suspected code defects.

Program service for Version 1 Release 2.x will be available until discontinued by IBM upon six months written notice.

Summary of New Functions in Release 2 Refresh

The following is a list of new functions added for this release. Complete descriptions for the new functions can be found in either this book or the *LoadLeveler User's Guide*.

AIX 4.1 Support

LoadLeveler now runs on systems running AIX 4.1.

New User Exits

- **AFS Token Handling**

  This enhancement provides a user exit which allows an installation to, for example, refresh an AFS token when a job is dispatched.

- **Customizable Mail Notification**

  This enhancement provides a user exit which you can use to override the default mail notification method.

- **User Environment Prolog and Epilog**

  This enhancement provides a user exit which allows the administrator to run processes, within the user's environment, before and after a job runs. The user exit can also dynamically set environment variables for the job.

- **Job Script Filter**

This enhancement provides a user exit which acts as filter on job submissions, thereby allowing you to make local modifications to job submission scripts and to other options.

Enhanced Group, User, and Class Support

- **Enhanced Group Configuration**

  The administrator now has the ability to associate a UNIX group ID with a LoadLeveler group. This allows accounting and scheduling to be done for a UNIX group.

- **Full Group Environment**

  LoadLeveler now sets the environment of both the primary and secondary UNIX groups.

- **Enhanced Group and Class Administrator Support**

  A group administrator can now cancel, hold, and resume jobs for a LoadLeveler group. A class administrator can now cancel, hold, and resume jobs for a LoadLeveler class.

- **Maximum Processors in User and Group Stanzas**

  You can now specify the **max_processors** keyword, which determines the maximum number of processors a user may request for a parallel job, in the User and Group stanzas.

- **Master Node Keyword**

  The **master_node_requirement** keyword in a class stanza allows you to have LoadLeveler allocate a master node for parallel jobs in this class.

New and Enhanced Commands

- *~userID* **Support**

  You can now use *~userID* in input, output, error, and executable statements to indicate the home directory of that user.

- **Job Submission Across LoadLeveler Pools**

  By setting an environment variable, you can send LoadLeveler commands to multiple clusters. For example, you can submit jobs to a local cluster *and* to a central cluster.

- **Automatic Directory Creation**

  LoadLeveler now creates directories for output files and error files if those directories do not exist. These directories are automatically created when the job starts.

- **llclass Command**

  The new **llclass** command reports class limits and class usage.

- **llq Command Enhancements**

  The following changes have been made to the **llq** command:

  – You can now query all jobs in a specific class.

  – A user can now query extended CPU usage for all jobs using **llq -x**.

- The long status output now includes information on the restart setting for jobs.

- **llsummary Command Enhancements**

  The following changes have been made to the **llsummary** command:

  - The **llsummary** command has been enhanced to allow additional reporting and filtering options.
  - The **llsummary** command can now display seconds since 1970 in the time range.

- **Default Class Search Support**

  LoadLeveler now searches a list of default classes to find a class whose resources match those of the job you submitted.

- **$(base_executable) Support**

  You can use the base name of an executable in expressions. The base name is the file name of the executable without the directory component. Similar support exists for $(executable).

Enhanced Starter Functions

- **Normalized CPU Units**

  You can now specify the CPU time in *units* rather than CPU seconds. If the LoadLeveler pool consists of machines of varying speeds, the real CPU seconds set for a job are determined by the speed of the processor on which the job runs.

Enhanced Scheduler Functions

- **Fairer CPU Allocation**

  LoadLeveler is now more precise in ensuring a fair allocation of the CPU. The scheduler now can be configured to take into account the number of currently running jobs for a user and the number of currently running jobs for a group.

- *Nice* **Support**

  Loadleveler can now set a job's *nice* value when a job is run.

Error Logging Enhancements

- LoadLeveler running on AIX 4.1 now supports all Error Log functions available with AIX 4.1.

Migration Consideration

If you upgrade LoadLeveler on an SP system, all of the nodes in the SP or the SP system partition must be upgraded.

Chapter 1. What is LoadLeveler?

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

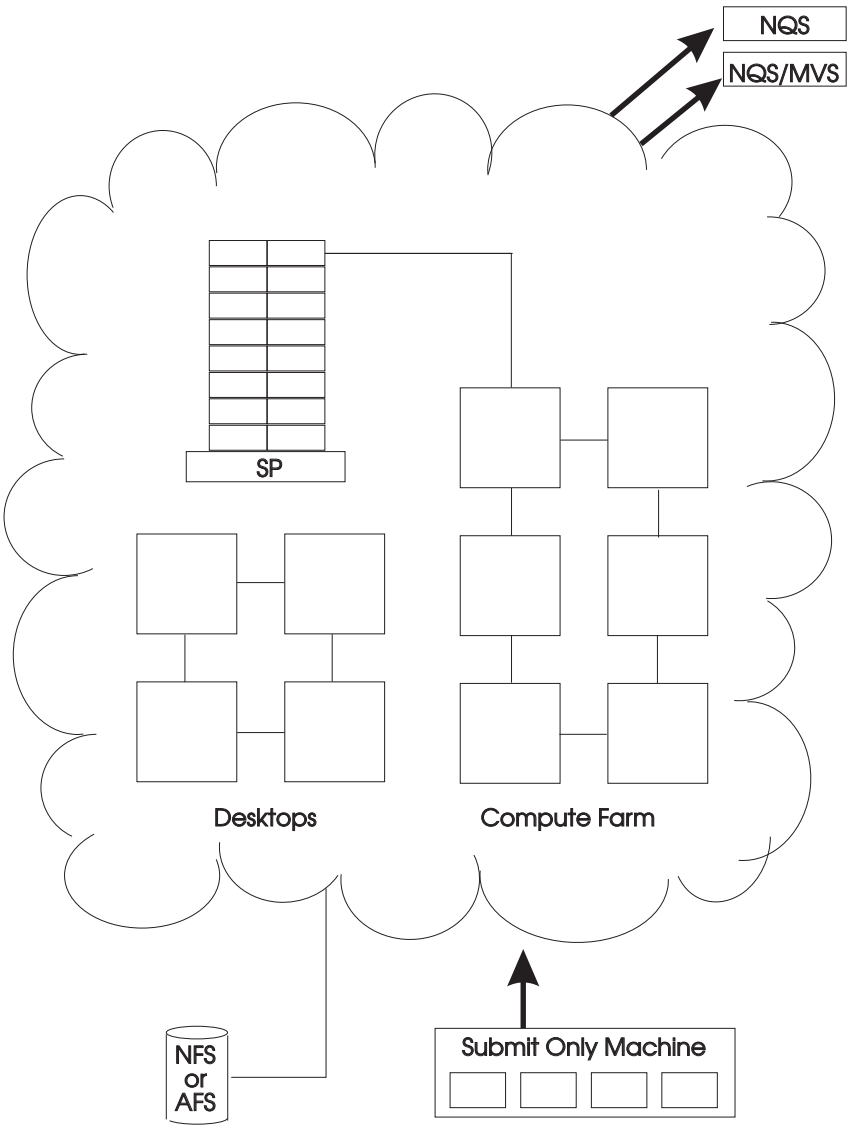


Figure 1. High-level LoadLeveler Configuration

The items within the cloud figure represent the different environments to which LoadLeveler can schedule jobs. The environments are heterogeneous clusters, dedicated nodes, and the RISC System/6000 Scalable POWERparallel System, also called the **SP** system. In addition, LoadLeveler can schedule jobs written for NQS to machines outside of the LoadLeveler cluster for execution. As the diagram also illustrates, there are also *submit-only* machines which allow users of these machines to have access to a limited number of LoadLeveler features. This type of machine is discussed in more detail later.

© Copyright IBM Corp. 1993, 1995

How LoadLeveler Works

This section describes how LoadLeveler works by first providing you with an explanation of some basic concepts and then walking you through the process of scheduling jobs. In addition to the information presented here, you should have a basic understanding of UNIX concepts.

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

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

Jobs

LoadLeveler schedules your jobs on one or more machines for processing. The definition of a *job*, in this context, is a set of *job steps* and for each job step, a different executable may be specified. You can use LoadLeveler to submit jobs which are made up of one or more job steps, and each job step depends upon the completion status of a previous job step. For example, the following diagram illustrates a stream of job steps:

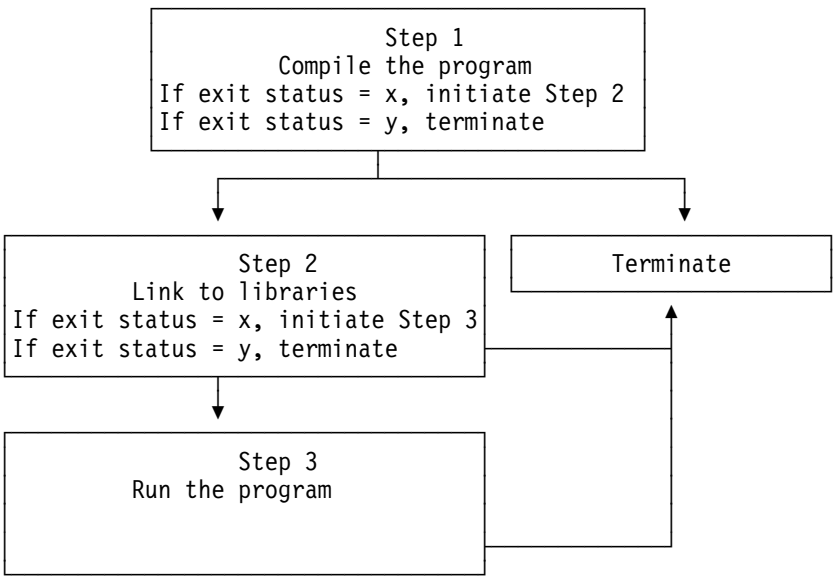


Figure 2. Illustration of Job Steps

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

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

You can submit batch jobs to LoadLeveler for scheduling. Batch jobs run in the background and generally do not require any input from the user. Batch jobs can either be *serial* or *parallel*. Serial jobs are processed sequentially by the machine and parallel jobs (jobs that have been written using a parallel language Application Program Interface (API)) are separated into multiple parts that can be processed simultaneously by several machines.

Machines and Workstations

In this context, machines and workstations are synonymous. In order for LoadLeveler to schedule a job on a machine, the machine must be a valid member of the LoadLeveler *pool*. A pool, sometimes known as a *cluster*, is the combination of all of the different types of machines that use LoadLeveler. There are a variety of types of machines that can be in the pool. They are:

- RISC System/6000
- SP System
- Sun SPARCstation
- SGI IRIS System
- HP Apollo 9000 Series 700 System.

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

To make a machine a member of the LoadLeveler pool, the administrator has to load the LoadLeveler software onto the machine and identify the central manager (described later). Once the machine becomes a valid member of the pool, LoadLeveler can schedule jobs to the machine.

Roles of Machines: Each machine in the LoadLeveler pool performs one or more roles that make job scheduling possible. These roles are described below:

- Scheduling Machine

  When a job is submitted to LoadLeveler, it gets placed in a queue managed by this machine. This machine contacts another machine that serves as a central manager for the entire LoadLeveler pool. (This role is described below). This scheduling machine asks the central manager to find a machine that can run the job and keeps persistent information regarding the job.

- Central Manager Machine

  The role of the central manager is to examine the job's requirements and find one or more machines in the LoadLeveler pool that will run the job. Once it finds the machine(s), it notifies the scheduling machine.

- Executing Machine

  The machine that runs the job is known as the executing machine.

- Submitting Machine

  There are two types of submitting machines. The first type is known as a *submit-only* machine. This type of machine participates in the LoadLeveler pool on a limited basis. Although the name implies that users of these machines can only submit jobs, they can also query and cancel jobs. Users of these machines also have their own Graphical User Interface (GUI) that provides them with the submit-only subset of functions. The submit-only machine

feature allows workstations that are not part of the LoadLeveler pool to submit jobs to the pool.

The second type of submitting machine plays a larger role in the LoadLeveler pool because it can perform more functions than the submit-only machine.

Keep in mind that one machine can assume multiple roles.

Machine Availability: There may be times when some of the machines in the LoadLeveler pool are not available to process jobs. This may be when the owners of the machines have decided to make them unavailable during times they need to use them exclusively. This ability of LoadLeveler to allow users to restrict the use of their machines provides flexibility and control over the resources.

Machine owners can make their personal workstations available to other LoadLeveler users in several ways, some of which are described in the following list:

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

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

How Does LoadLeveler Schedule Jobs to Run on Machines?

Once a user submits a job to LoadLeveler, LoadLeveler examines the job in order to determine what resources it needs to run. Then, LoadLeveler determines which machine in the LoadLeveler pool is best suited to run the job. Once the appropriate machine is found, LoadLeveler dispatches the job to the machine. To provide this function, LoadLeveler uses the concept of *queues*.

Job Queues: A job queue is a list of jobs that are waiting to be processed. When a user submits a job to LoadLeveler, the job enters into a database that resides on one of the machines in the LoadLeveler pool until it is ready to be dispatched to another machine to be run. The following diagram illustrates this:

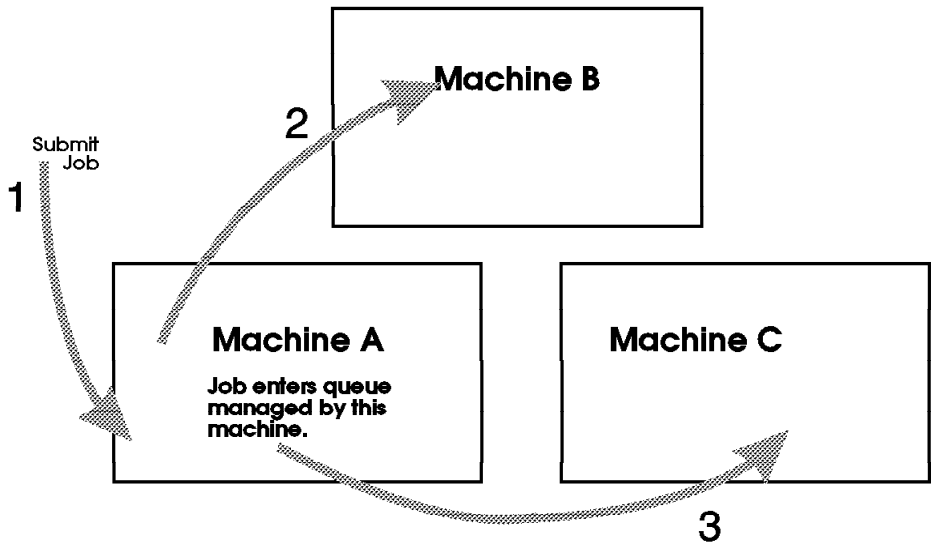


Figure 3. Job Queues

Once LoadLeveler examines the job to determine its required resources, the job is dispatched to a machine to be processed. Arrows 2 and 3 indicate that the job can be dispatched to either one machine, or in the case of parallel jobs, to multiple machines. Once the job reaches the executing machine, it is executed.

Most of the LoadLeveler machines have a UNIX daemon that manages its own list of LoadLeveler jobs. When a user submits a job to LoadLeveler, the job enters into a database that LoadLeveler manages. The jobs do not necessarily get dispatched to machines in the pool based upon a first-come first-serve basis. Instead, LoadLeveler examines the requirements and characteristics of the job and the availability of machines and determines when the best time is for the job to be dispatched.

Job Classes

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

You can specify which types of jobs will run on a machine by specifying the type(s) of job classes the machine will support.

Job Priorities

LoadLeveler also examines a job's priority in order to determine when to schedule it on a machine. A priority of a job is used to determine its position among a list of all jobs waiting to be dispatched. This section discusses the different types of priorities and how LoadLeveler uses these priorities when considering jobs for dispatch.

User Priority: Every job has a user priority associated with it. This priority is specified by the user in the job command file and it is a number between 0 and 100 inclusively. This priority specifies the rank of this job in relation to other jobs belonging to the same user and the same class. The default user priority is 50. Note that this is not the UNIX *nice* priority.

System Priority: Every job has a system priority associated with it. This priority is specified in LoadLeveler's configuration file using the **SysPrio** expression. This priority can take into account the priority assigned to the job class, the group to which the user belongs, and priority assigned to the individual user. The priorities defined in this expression (class, user, and group) are defined later in this book.

How Does a Job's Priority Affect Dispatching Order?

The following table represents the priorities assigned to a job at submission time. Two of the jobs belong to a user named Joe and three belong to a user named Rich.

Table 1. Job list at submission time

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

In order to determine in which order to consider dispatching the jobs, LoadLeveler takes into account both Sysprio and User priority to calculate the Adjusted System Priority. LoadLeveler looks for jobs to be dispatched using the Adjusted System Priorty which is determined as follows:

- All jobs are assigned a System Priority (SYSPRIO) at job submission time.
- Jobs are ordered first by System Priority (SYSPRIO).
- Jobs belonging to the same user and the same class are then assigned the Adjusted System Priority which is the System Priority assigned in order of user priority.

For example, in the previous table, user Joe has two jobs in Class A with SYSPRIOs of 9 and 8 respectively. Since Joe2 has the higher user priority, the adjusted system priority will be 9 because 9 is the highest System Priority within that group. The following table reflects this which shows Adjusted System Priority for all of the jobs.

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

Understanding the SysPrio Expression

As was stated earlier, every job has a system priority associated with it. This priority is specified in LoadLeveler's configuration file using the **SysPrio** expression. This priority can take into account the priority assigned to the job class, the group to which the user belongs, and the priority assigned to the individual user.

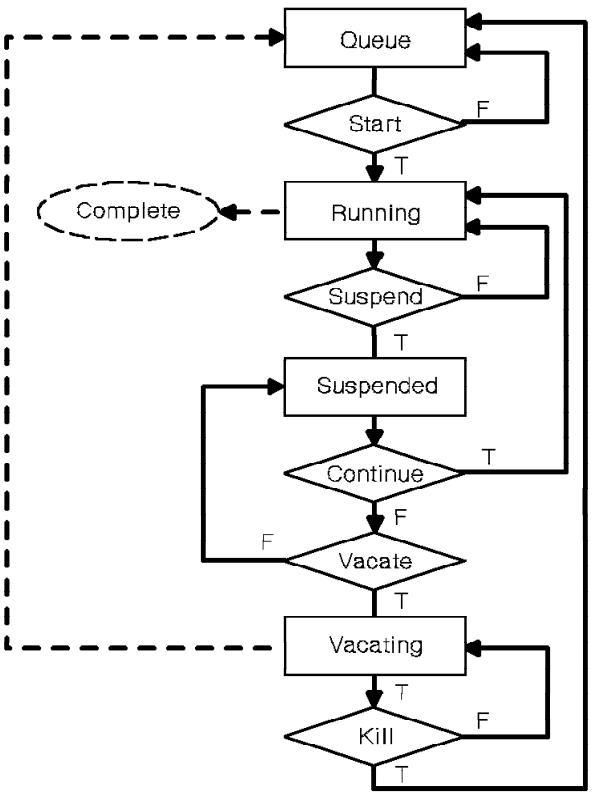
The SysPrio expression can contain the class, group, and user priorities and it is evaluated by LoadLeveler to determine the overall system priority of a job. This system priority number is assigned when the central manager adds the new job to the queue of jobs eligible for dispatch. The following is an example of a system priority expression that takes into account the class priority, the user priority, and the group priority:

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

For more information on the system priority expression, refer to “Step 5: Prioritize the Queue Maintained by the Negotiator” on page 47.

How Does a Job's Status Change as it is Processed?

After LoadLeveler selects a job for execution and the job enters the queue, the job can be in one of several phases as shown in the following figure:



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

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

Suspended	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 available 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.
Vacating	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. Usually, the job is simply returned to the queue for dispatching.
Kill	A job taking too much time to vacate may be moved to the Kill state.

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.

LoadLeveler Daemons and Processes

This section lists and describes the processes and daemons that LoadLeveler uses to process jobs. For detailed information on these daemons and processes, refer to Appendix D, “LoadLeveler Daemons and Processes” on page 213.

LoadL_master

Referred to as the **master** daemon, this manages all LoadLeveler daemons on its machine.

LoadL_schedd

Referred to as the **schedd** daemon, this reviews batch submissions on its machine and manages a set of jobs submitted to the machine.

LoadL_startd

Referred to as the **startd** daemon, this accepts jobs for dispatch on its machine.

LoadL_starter

Spawned by **startd** daemon, the **starter** process manages a running job on the executing machine.

LoadL_kbdd

Referred to as the **keyboard** daemon, this monitors X and mouse activity on platforms that do not already have this functionality.

LoadL_collector

Referred to as the **collector** daemon, this is the collector of machine status from all machines in the LoadLeveler pool.

LoadL_negotiator

Referred to as the **negotiator** daemon, this is the daemon that collects the job status from all the machines in the LoadLeveler pool.

> **Note**
>
> Some of these processes reside on every machine in the LoadLeveler pool. The **collector** and **negotiator** reside only on the machine designated as the central manager. See "Step 8: Specify Alternate Central Managers" on page 51 for information on the central manager.

Central Manager

The Central Manager consists of the **negotiator** and the **collector** daemons. Together, these two daemons coordinate LoadLeveler activities among the machines in the pool. It maintains the status of all machines and jobs and makes decisions on where jobs should be run.

The LoadLeveler Job Cycle

The following diagram illustrates the information flow through the LoadLeveler pool:

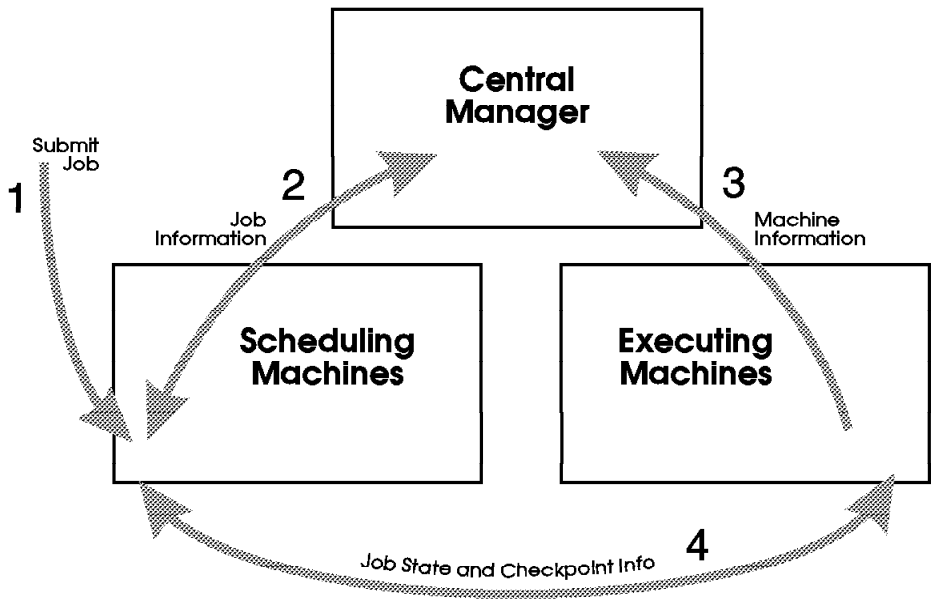


Figure 4. High-level illustration of job flow

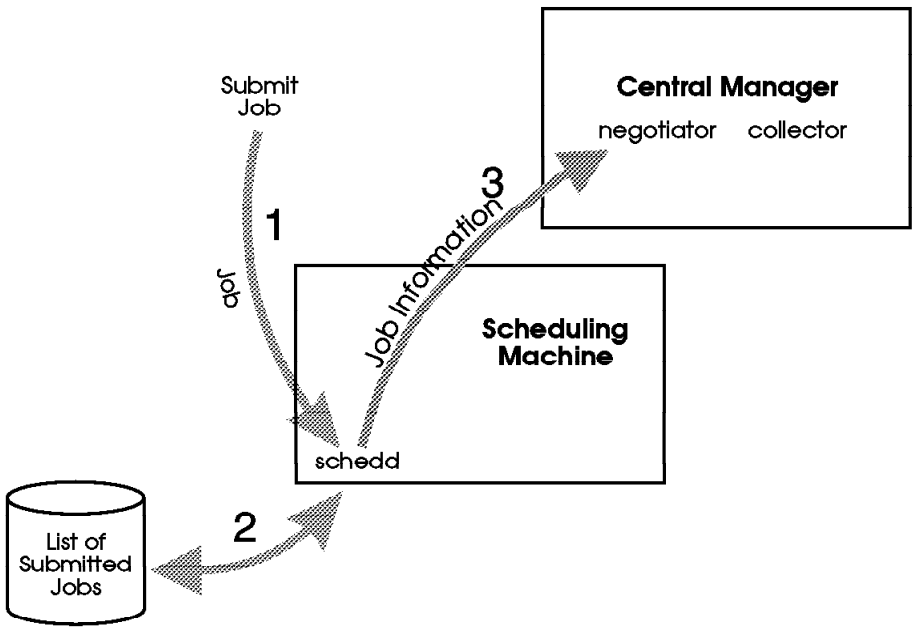
In general there is a managing machine known as the **central manager**. There are also machines that acts as scheduling machines and machines that serve as the executing machines. The arrows in the previous diagram illustrate the following:

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

- Arrow 4 indicates that the scheduling machine contacts the executing machine and provides it with information regarding the job.

This high-level diagram is broken down into the following more detailed diagrams illustrating the flow of a job using LoadLeveler.

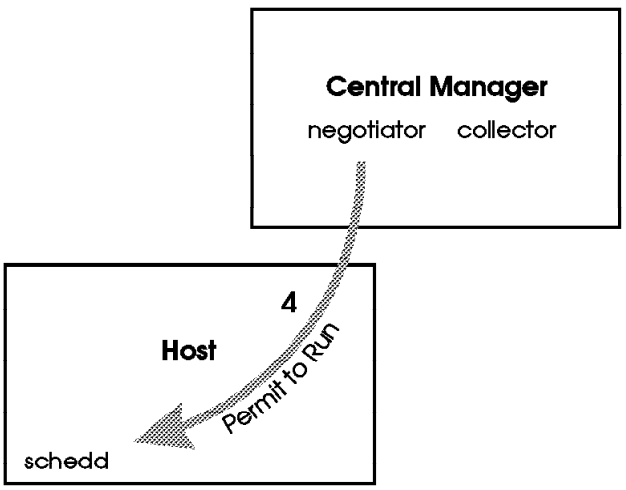
1 Submit a LoadLeveler job



This diagram illustrates that the scheduling machine has the **schedd** daemon on it. It can also have the **startd** daemon on it. The central manager machine has two daemons that reside on it. These daemons are the **negotiator** and **collector** daemons. The arrows in the diagram illustrate the following:

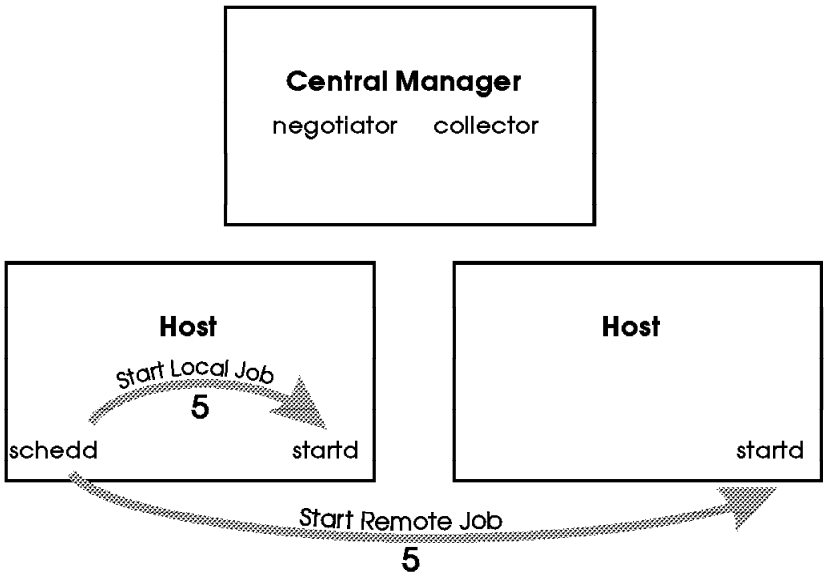
- Arrow 1 indicates that a job has been submitted to the scheduling machine.
- Arrow 2 indicates that the **schedd** daemon, on the scheduling machine, stores all of the relevant job information on local disk.
- Arrow 3 indicates that the **schedd** daemon sends job description information to the **negotiator** daemon in the central manager machine.

2 Permit to run



In this diagram, arrow 4 indicates that the **negotiator** daemon authorizes the **schedd** daemon to begin taking steps to run the job. This authorization is called a *permit to run*. Once this is done, the job is labeled as pending.

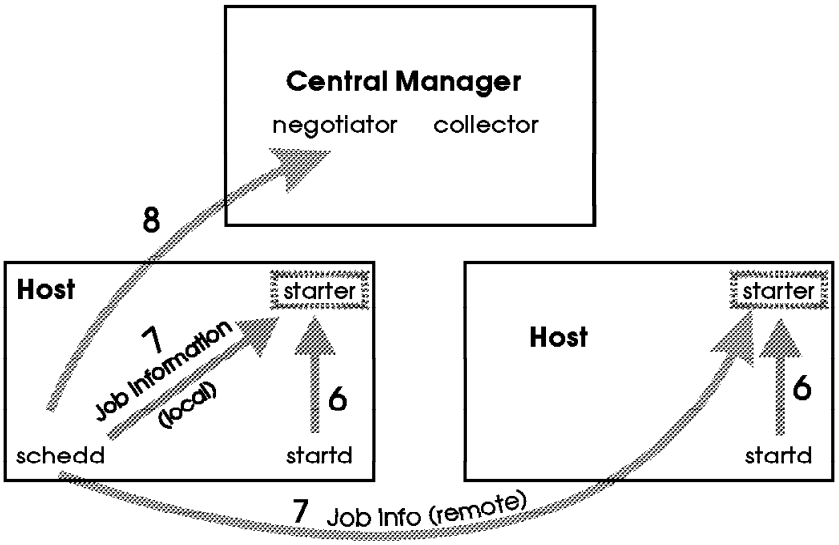
3 Prepare to run



In the last diagram, arrow 4 indicated that the **negotiator** daemon informed the **schedd** daemon that it could contact an executing machine to run the job. This executing machine can either be a local machine or a remote machine. A local machine is the same machine as the **schedd** daemon is on. A remote machine is another machine in the pool.

In this diagram, arrow 5 illustrates that the **schedd** daemon contacts the **startd** daemon on the executing machine and requests that it start the job.

4 Initiate job

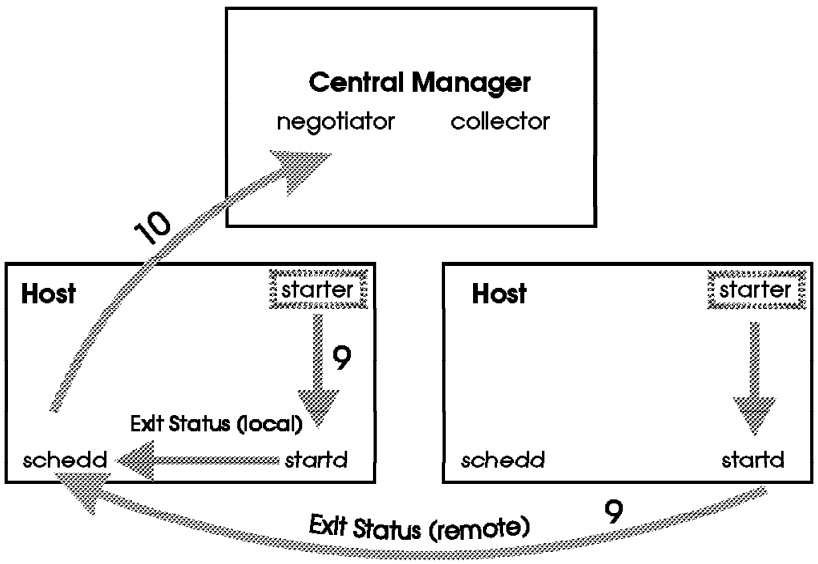


The arrows in this diagram illustrate the following:

- The two arrows numbered 6 indicate that the **startd** daemon, on the executing machine, spawns a **starter** process and awaits more work.
- The two arrows numbered 7 indicate that the **schedd** daemon sends the **starter** process the job information and the executable. The executable is part of the job that gets processed.
- Arrow 8 indicates that the **schedd** daemon notifies the **negotiator** daemon that the job has been started and the **negotiator** daemon marks the job as running.

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

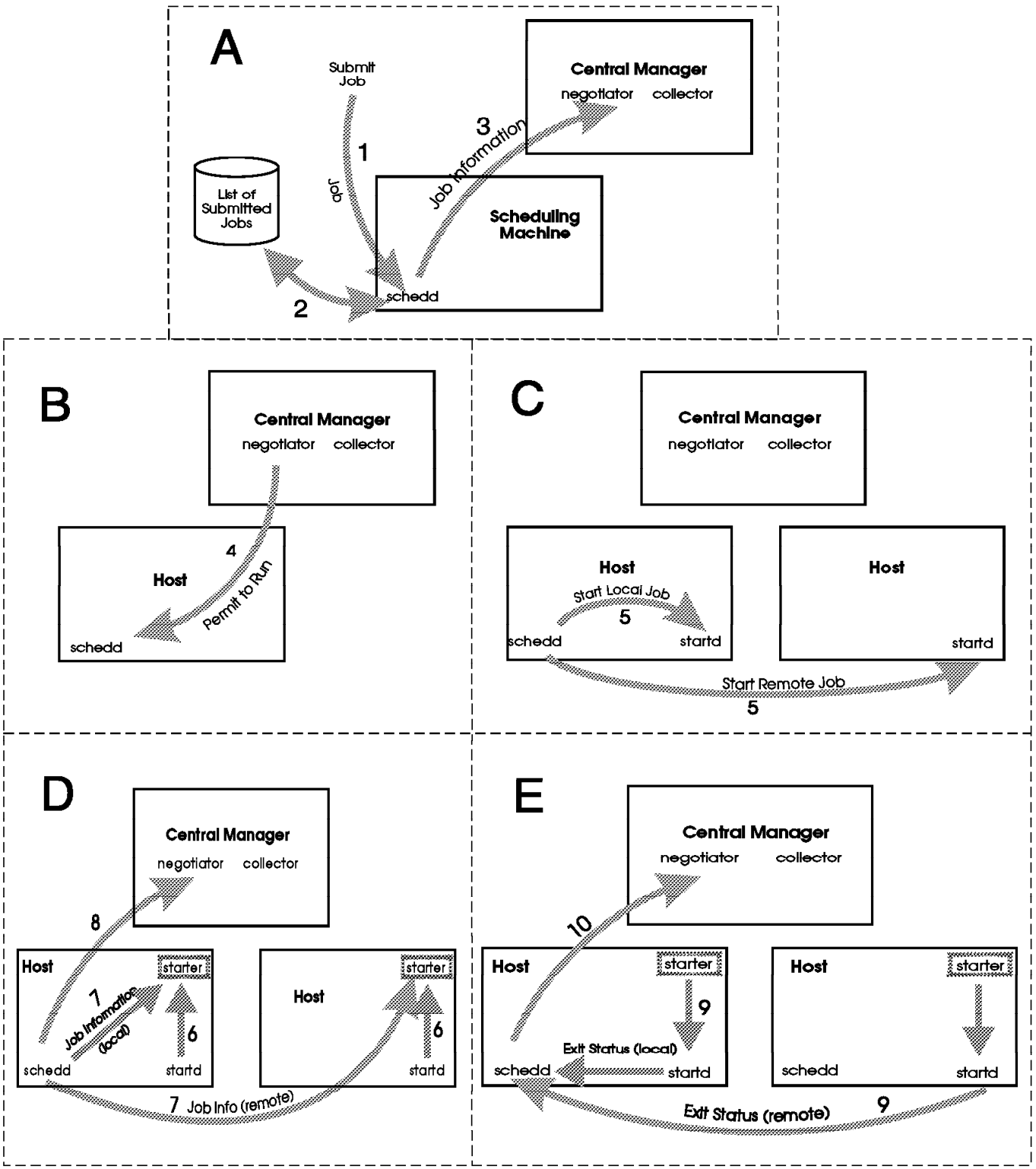
5 Complete job



The arrows in this diagram represent the following:

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

6 Summary of Job Flow Cycle



This diagram is an aggregate of all of the previous pictures. It illustrates the complete job flow.

Chapter 2. Administering and Configuring LoadLeveler

After installing LoadLeveler, you need to customize it by modifying both the *administration* file and the *configuration* file. The administration file optionally lists and defines the machines in the LoadLeveler cluster (also called a pool) and the characteristics of classes, jobs, and groups.

The configuration file contains many parameters that you can set or modify that will control how LoadLeveler operates. LoadLeveler provides you with one global configuration file that you can use to manage your LoadLeveler cluster.

In order to easily manage LoadLeveler, you should have only one administration file and one global configuration file, centrally located on a machine in the LoadLeveler cluster. Every other machine in the cluster must be able to read the administration and configuration file that are located on the central machine. LoadLeveler does not prevent you from having multiple copies of administration files but you need to be sure to update all the copies whenever you make a change to one. Having only one administration file prevents any confusion.

Similarly, you should have only one global configuration file. You can, however, have multiple local configuration files that specify information specific to individual machines. For more information on the global and local configuration files, refer to "Configuring LoadLeveler" on page 34.

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

Planning Considerations

Hardware availability

Loading a workstation that already has too little memory, disk space, or paging space, with all of the LoadLeveler daemons can affect LoadLeveler's performance and affect CPU consumption.

Node availability

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

Common name space

To run jobs on any machine in the LoadLeveler cluster, you must have the same uid (the system ID number for a user) and gid (the system ID number for a group) on every machine in the cluster. If you do not have a user ID on one machine, your jobs will not run on that machine.

Performance

Faster networks, such as FDDI, allow you to manage LoadLeveler files from a centralized directory without major losses in performance. Slower networks, however, decrease the performance of all software products running in distributed mode.

You should keep the log directory in a local file system in order to maximize performance.

© Copyright IBM Corp. 1993, 1995

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

Resource Handling
Some nodes in the LoadLeveler pool might have special software installed that users might need to run their jobs successfully. You should configure LoadLeveler to distinguish those nodes from other nodes using, for example, job classes.

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

LoadLeveler Interactive Session Support (ISS)
Since LoadLeveler ISS feature is handled using name resolution, you will need to decide which node will act as the ISS Domain Name Server. For more information, refer to Chapter 4, “Interactive Login Sessions” on page 89.

The LoadLeveler User ID and Group ID

LoadLeveler has a special user ID, called the *LoadLeveler user ID*, which performs specific functions. By default, this user ID is called **loadl**. Some of the characteristics of **loadl** are:

- LoadLeveler daemons run under this ID.
- Many LoadLeveler files are owned by this ID.
- The configuration files, by default, reside on this ID.

LoadLeveler also has a group ID, which is also called **loadl**.

For information on changing the default user ID and group ID, and for more information on the configuration files, see “Configuring LoadLeveler” on page 34.

Where to Begin?

Setting up LoadLeveler involves defining machines, users, and the way in which they interact to process jobs quickly and efficiently. However, those with more experience in system administration and job scheduling will perform these administration tasks with greater ease and more confidence than those who may be new to such tasks. Therefore, there are two entry points for various levels of experience.

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

Intermediate or Beginner: If you are experienced in UNIX system administration but are unfamiliar with job scheduling systems or your experience is limited, first review the section “Quick Set Up” on page 17. This will provide you with a quick overview. If you'd like, you can set up your environment using the steps presented

in that section or if you would like a slower approach, go to “Administration File Structure and Syntax” on page 18 and read from this section until the end of the chapter.

Expert: If you are very familiar with UNIX system administration and job scheduling, and have some idea how you want to distribute your workload, go to “Quick Set Up.” Each step in this short procedure refers you to a detailed discussion of the task at hand. The sample configuration and administration files included in the samples subdirectory (and shown in Appendix C, “Sample Files” on page 199) will also provide valuable assistance.

Quick Set Up

If you are very familiar with UNIX system administration, and job scheduling, and don't want to read through this whole chapter but prefer to start up with only the essentials, perform the tasks listed here to get LoadLeveler up and running on your network quickly in a default configuration. For a quick set up, it is recommended that you use **loadl** as the LoadLeveler user ID. Afterward, you can fine tune your configuration for greater efficiency when you become more familiar with the details of LoadLeveler.

The sample configuration and administration files included in the samples subdirectory (and shown in Appendix C, “Sample Files” on page 199) will also provide valuable assistance.

Step 1. Ensure that the installation procedure has completed successfully and that the configuration file, **LoadL_config**, exists in LoadLeveler's home directory or in the directory specified in /**etc/LoadL.cfg** (if this file exists). See “Configuring LoadLeveler” on page 34 for more information.

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

LOADL_ADMIN = *list of user name(s) (required)*
where *user name(s)* is a blank-delimited list of those individuals who will have administrative authority.

Refer to “Step 1: Define LoadLeveler Administrators” on page 41 for more instructions.

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

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

With these settings, the CONTINUE, VACATE, and KILL statements have no effect. See “Step 2: Manage a Job's Status Using Control Functions” on page 42 for more information on these statements.

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

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

These are baseline settings that provide a test environment which you can later adjust for more efficient operation. These keywords are explained in “Summary of Process Specific Keywords” on page 64.

Step 5. Define a machine to act as the LoadLeveler central manager by coding one machine stanza as follows in the **LoadL_admin** file. (Replace *machinename* with the actual name of the machine.)

```
machinename: type = machine
             central_manager = true
```

Note: Do not specify more than one machine as the central manager. Also, if during installation, you ran **llinit** with the **-cm** flag, the central manager is already defined in the LoadL_admin file because the **llinit** command takes parameters you entered and updates the administration and configuration files. See “Step 1: Specify Machine Stanzas” on page 20 for more information.

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

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

```
llctl -h hostname start
```

See “llctl - LoadLeveler Control” on page 134 for more information.

You can also issue the following command to start LoadLeveler on all machines beginning with the central manager. When using this command though, be sure that all the machines are listed in the administration file.

```
llctl -g start
```

This command only affects machines that are defined in the administration file.

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

Administration File Structure and Syntax

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

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

Class stanza
Describes the characteristics of the job classes.

User stanza
Lists the LoadLeveler users.

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

Generally, stanzas take the following format:

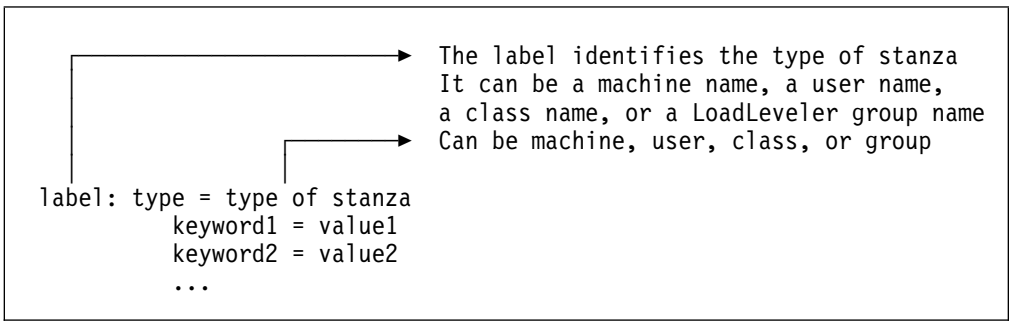


Figure 5. Administration File Stanzas

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

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

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

```
machine_a: type            = machine
           central_manager = true

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

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

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

The characteristics of a stanza are:

- Every stanza has a label associated with it.
- Every stanza has a *type* field that specifies it as a *user*, *class*, *machine*, or *group* stanza.
- New line characters are ignored. This means that the different parts of a stanza may cross line boundaries or be included on the same line.
- White space is ignored, other than to delimit keyword identifiers. This eliminates confusion between tabs and spaces at the beginning of lines.

- A cross-hatch sign (#) identifies a comment and may appear anywhere on the line. All characters following this sign on that line are ignored.
- Multiple stanzas of the same label are allowed, but are not merged. It is not recommended that you have multiple stanzas of the same label because results may be unpredictable.

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

Customizing the Administration File

You can add as many stanzas as you would like to the administration file. To assist you in modifying this file, this section is presented in a step-by-step manner that walks you through the process. You do not have to perform the steps in the order that they appear here.

Step 1	Specify machine stanzas
Step 2	Specify user stanzas
Step 3	Specify class stanzas
Step 4	Specify group stanzas

Step 1: Specify Machine Stanzas

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

If you do not specify a machine stanza for a machine in the pool, the machine and the central manager will still communicate and jobs will still be scheduled to the machine but the machine will be assigned the default values described in the default machine stanza.

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

- If you set the **MACHINE_AUTHENTICATE** keyword to **true** in the configuration file, then you must create a machine stanza for each node that LoadLeveler includes in the pool.
- If the machine's hostname (the name of the machine returned by the UNIX hostname command) does not match an interface name. In this case, you must specify the interface name as the machine stanza name and specify the machine's hostname using the **alias** keyword.
- If the machine's hostname does match an interface name but not the correct interface name.
- If you want to use the resource manager to schedule parallel jobs on the SP system, you need to add the **rm_host** keyword to the machine stanza.

Machine stanzas take the following format:

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

Figure 6. Machine Stanzas

You must specify one and only one machine stanza identifying the central manager. For example:

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

alias = *machine_name*

where *machine_name* is a blank-delimited list of one or more machine names. Depending upon your network configurations, you may need to add **alias** keywords for machines that have multiple interfaces.

Note: In general, if your cluster is configured with machine hostnames which match the hostnames corresponding to the IP address configured for the LAN adapters which LoadLeveler is expected to use, you will not have to specify the **alias** keyword. For example, if all of the machines in your cluster are configured like this sample machine, you should not have to specify the **alias** keyword.

```
Machine porsche.kgn.ibm.com
```

- The hostname command returns **porsche.kgn.ibm.com**
- The Ethernet adapter address 129.40.8.20 resolves to hostname **porsche.kgn.ibm.com**.

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

1. Machine **yugo.kgn.ibm.com**
   - The hostname command returns **yugo.kgn.ibm.com**.
   - The Ethernet adapter address 129.40.8.21 resolves to hostname **chevy.kgn.ibm.com**.
   - No adapter address resolves to **yugo**.

   You need to code the machine stanza as:

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

2. Machine **rover.kgn.ibm.com**
   - The hostname command returns **rover.kgn.ibm.com**.

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

You need to code the machine stanza as:

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

central_manager = true| **false** | alt
where **true** designates this machine as the LoadLeveler central manager host, where both the **negotiator** and **collector** daemons run. Only one machine stanza should have this keyword defined as **true**.

false specifies that no central manager runs on this machine.

alt specifies that this machine can serve as an alternate central manager in the event that the primary central manager is not functioning. For more information on recovering if the primary central manager is not operating, refer to “What Happens if the Central Manager Isn't Operating?” on page 176. Submit-only machines cannot have their machine stanzas set to this value.

Note:

If you are going to select machines to serve as alternate central managers, you should look at the following keywords in the configuration file or the default values will be used:

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

For information on setting these keywords, see “Step 8: Specify Alternate Central Managers” on page 51.

pvm_root = *pathname*
Where *pathname* specifies the location of the directory in which PVM is installed. The default pathname is **/u/loadl/pvm3**.

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

submit_only = true| **false**
where **true** designates this as a submit-only machine. If set to **true**, set the **central_manager** and **schedd_host** to **false**.

speed = *number*
where *number* is a floating point number that is used for machine scheduling purposes in the **MACHPRIO** expression. For more information on machine scheduling and the MACHPRIO expression, see “Step 6: Prioritize the Order of Executing Machines Maintained by the Negotiator” on page 48. In addition, the **speed** keyword is also used to define the weight associated with the machine.

This weight is used when gathering accounting data on a machine basis. The default weight is 1.0.

The following example illustrates how the **speed** keyword can be used for assigning weights to machines. To see how the **speed** keyword can be used to schedule machines, refer to "Step 6: Prioritize the Order of Executing Machines Maintained by the Negotiator" on page 48.

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

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

For example, suppose a LoadLeveler cluster consists of two machines, A and B, where machine B is three times as fast as machine A. That is, machine A has **speed=1.0**, and machine B has **speed=3.0**. Suppose a job has **job_cpu_limit = 12**. The job will get 12 seconds if it runs on machine B, and four seconds if it runs on machine A.

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

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

max_jobs_scheduled = *number*
where *number* is the maximum number of jobs from this scheduling (**schedd**) machine that can run in the LoadLeveler pool at one time.

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

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

- Polls the resource manager for SP system specific information such as the adapter, state, pool, and subpool and updates the central manager. If the resource manager has dedicated a machine for interactive or batch use, LoadLeveler takes the machine off line and marks it as reserved. The user will now be able to specify **Adapter** and **Pool** through the **requirements** keyword in the job command file.

- Allocates machines selected by the **negotiator** through the resource manager for both serial and parallel jobs. All machines are allocated as shared so that serial, parallel and interactive jobs may run simultaneously. Machines may become dedicated through interactive jobs. The job will return to the queue if the allocation fails.

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

name_server = *list*

where *list* is a blank-delimited list of character strings that is used to specify which nameserver(s) are used for the machine. Valid strings are DNS, NIS, and LOCAL. LoadLeveler uses the list to determine when to append a DNS domain name for machine names specified in LoadLeveler commands issued from the machine described in this stanza.

If DNS is specified alone, LoadLeveler will always append the DNS domain name to machine names specified in LoadLeveler commands. If NIS or LOCAL is specified, LoadLeveler will never append a DNS domain name to machine names specified in LoadLeveler commands. If DNS is specified with either NIS or LOCAL, LoadLeveler will always look up the name in the administration file to determine whether to append a DNS domain name. If the name is specified with a trailing period, it doesn't append the domain name.

Examples of Machine Stanzas

Example 1: In this example, the machine is being defined as the central manager.

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

Example 2: This example sets up a submit-only node. Note that the submit-only keyword is set to true:

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

Example 3: In the following example, machine_c is the central manager, has an alias associated with it, and can run parallel PVM jobs:

```
#
machine_c: type = machine
           central_manager = true    # central manager runs here
           schedd_host = true
           alias = brianne
           pvm_root = /u/brianne/loadl/1.2.0/aix32/pvm3
```

Step 2: Specify User Stanzas

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

User stanzas take the following format:

```
                                  User name

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

Figure 7. User Stanzas

priority = *number*

where *number* can be any integer. If not specified, the default is zero. The number specified for priority is referenced as **UserSysprio** in the configuration file. **UserSysprio** can be used in the assignment of job priorities. If the variable **UserSysprio** does not appear in the SYSPRIO expression in the configuration file, the priority numbers for users specified here in the administration file have no effect. See “Step 5: Prioritize the Queue Maintained by the Negotiator” on page 47 for more information about the **UserSysprio** keyword.

default_class = *list*

where *list* is a blank-delimited list of class names used for jobs which do not include a **class** statement in the job command file. If you specify only one default class name, this class is assigned to the job. If you specify a list of default class names, LoadLeveler searches the list to find a class which satisfies the resource limit requirements. If no class satisfies these requirements, LoadLeveler rejects the job.

Suppose a job requests a CPU limit of 10 minutes. Also, suppose the default class list is `default_class = short long`, where `short` is a class for jobs up to five minutes in length and `long` is a class for jobs up to one hour in length. LoadLeveler will select the `long` class for this job because the `short` class does not have sufficient resources.

If no **default_class** is specified in the user stanza, or if there is no user stanza at all, then jobs submitted without a **class** statement are assigned to the **default_class** that appears in the default user stanza. Additionally, if no **default_class** appears in the default user stanza, then an error occurs when a user attempts to submit a job.

maxjobs = *number*

where *number* is the maximum number of jobs a single user can run at any time. If the user stanza does not specify the **maxjobs** or if there is no user stanza at all, the maximum jobs that can be simultaneously run by the user is defined in the default stanza. If no value is found, or the limit found is -1, then no limit is placed on the number of jobs that can simultaneously run for the

user. Regardless of this limit, there is no limit to the number of jobs a user can submit.

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

maxqueued = *number*
where *number* is the maximum number of jobs allowed in the queue for that user. This is the maximum number of jobs which can be either running or being considered to be dispatched by the **negotiator** for that user. You can use this keyword to prevent individual users from dominating the number of jobs that are either running or are being considered to run. If no **maxqueued** is specified in the user stanza, or if there is no user stanza, the maximum number of jobs that can simultaneously be in the queue is defined in the default stanza. If no value is found, or the limit found is -1, then no limit is placed on the number of jobs that can simultaneously be in the job queue for that user. Regardless of this limit, there is no limit to the number of jobs a user can submit.

account =*list*
where *list* is a blank-delimited list of account numbers that identifies the account numbers a user may use when submitting jobs. The default is a null list.

default_group = *group name*
where *group name* is the default group assigned to jobs submitted by the user. If a **default_group** statement does not appear in the user stanza, or if there is no user stanza at all, then jobs submitted by the user without a **group** statement are assigned to the **default_group** that appears in the default user stanza. Additionally, if no **default_group** appears in the default user stanza, then an error occurs when a user attempts to submit a job.

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

Examples of User Stanzas

Example 1: In this example, user `fred` is being provided with a user stanza. His jobs will have a user priority of 100. If he does not specify a job class in his job command file, the default job class **class_a** will be used. In addition, he can have a maximum of 15 jobs running at the same time.

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

Example 2: In this example, user **jane**'s jobs have a user priority of 50, and if she does not specify a job class in her job command file the default job class **small_jobs** is used. This user stanza does not specify the maximum number of jobs that Jane can run at the same time so this value defaults to the value defined in the default stanza. Also, suppose Jane is a member of the primary UNIX group

"staff." Jobs submitted by Jane will use the default LoadLeveler group "staff." Lastly, Jane can use three different account numbers.

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

Step 3: Specify Class Stanzas

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

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

Figure 8. Class Stanzas

priority = *number*
: where *number* can be any integer. If not specified, the default is zero (0). The number specified for priority is referenced as **ClassSysprio** in the configuration file. You can use **ClassSysprio** when assigning job priorities. If the variable **ClassSysprio** does not appear in the SYSPRIO expression, then the priority specified here in the administration file is ignored. See "Step 5: Prioritize the Queue Maintained by the Negotiator" on page 47 for more information about the **ClassSysprio** keyword.

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

nice = *value*
: where *value* is the amount by which the current UNIX *nice* value is incremented. The *nice* value is one factor in a job's run priority. The lower the number, the higher the run priority. If two jobs are running on a machine, the *nice* value determines the percentage of the CPU allocated to each job.

This value ranges from -20 to 20. Values out of this range are placed at the top (or bottom) of the range. For example, if your current *nice* value is 15, and you specify `nice = 10`, the resulting value is 20.

For more information, consult the appropriate UNIX documentaion.

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

exclude_users = *list*
where *list* is a blank-delimited list of users who are *not* permitted to submit jobs of that *class name*. Do not specify both a list of included users and a list of excluded users. Only one of these may be used for any class. The default is that no users are excluded.

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

exclude_groups = *list*
where *list* is a blank-delimited list of groups who are *not* allowed to submit jobs of that *class name*. Do not specify both a list of included groups and a list of excluded groups. Only one of these may be used for any class. The default is that no groups are excluded.

Specify the following keywords in the class stanza when you want jobs submitted to this class to be routed to Network Queuing System (NQS) machines:

NQS_class = true|**false**
When **true**, any job submitted to this class will be routed to an NQS machine.

NQS_submit = *name*
where *name* is the name of the NQS pipe queue to which the job will be routed. When the job is dispatched to LoadLeveler, LoadLeveler will invoke the **qsub** command using the name of this queue. There is no default.

NQS_query = *queue names*
where *queue names* is a blank-delimited list of queue names (including host names if necessary) to be used with the **qstat** command to monitor the job and with the **qdel** command to cancel the job. There is no default.

For more information on routing jobs to machines running NQS, refer to Chapter 7, “Routing Jobs to NQS Machines” on page 121.

Specify the following keyword in the class stanza when you are submitting parallel jobs:

master_node_requirement = true|**false**
where **true** specifies that parallel jobs in this class require the master node feature. For these jobs, LoadLeveler allocates a master node having the **master_node_exclusive = true** setting. If most or all of your parallel jobs

require this feature, you should consider placing the statement **master_node_requirement = true** in your default class stanza. Then, for classes that do not require this feature, you can use the statement **master_node_requirement = false** in their class stanzas to override the default setting. For more information, see the **master_node_exclusive** keyword described in "Step 1: Specify Machine Stanzas" on page 20.

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

class_comment = *"string"*
where *string* is text characterizing the class. This information appears when the user is building a job command file using the GUI and requests Choice information on the classes to which he or she is authorized to submit jobs. The length of the string cannot exceed 1024 characters.

job_cpu_limit = *hardlimit*
These values specify the maximum total CPU time to be used by all processes of a job step. For example, if a job step forks to produce multiple processes, the sum total of CPU consumed by all of the processes is added and controlled by this limit. It is expressed as *hours:minutes:seconds.fraction*. Fractions are rounded to seconds. The following strings can also be used:

rlim_infinity	represents the largest positive number.
unlimited	has same effect as **rlim_infinity**.
copy	use the limit currently active when the job is submitted.

For example:

```
job_cpu_limit = 10000
```

cpu_limit = *hardlimit*
These values specify the maximum CPU time to be used by each individual process of a job step, expressed as *hours:minutes:seconds.fraction*. Fractions are rounded to seconds. The following strings can also be used:

rlim_infinity	represents the largest positive number.
unlimited	has same effect as **rlim_infinity**.
copy	use the limit currently active when the job is submitted.

Some examples:

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

data_limit = *hardlimit*

Specifies the maximum size data segment to be used by each process of the submitted job, expressed 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)

kw kilowords (2^{10} words)

mb megabytes (2^{20} bytes)

mw megawords (2^{20} words)

gb gigabytes (2^{30} bytes)

If no units are specified, then bytes are assumed.

The same shorthand strings can be used to specify an infinite limit or copy the limit from the submitting machine as described for the **cpu_limit** parameter.

Some examples:

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

core_limit = *hardlimit*

Specifies the maximum size of a core file, expressed in the same format as the **data_limit**. The same shorthand strings that were described for **cpu_limit** may be used to specify infinity or to copy limits from the submitting machine. The *hardlimit* parameter is interpreted the same way as explained for **cpu_limit**.

Some examples are:

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

file_limit = *hardlimit*

Specifies the maximum size a file can be created, expressed in the same format as the **data_limit**. The same shorthand strings that were described for **cpu_limit** may be used to specify infinity or to copy limits from the submitting machine. The *hardlimit* parameter is interpreted the same way as explained for **cpu_limit**.

stack_limit = *hardlimit*

Specifies the maximum size for a stack, expressed in the same format as the **data_limit**. The same shorthand strings that were described for **cpu_limit** may be used to specify infinity or to copy limits from the submitting machine. The *hardlimit* parameter is interpreted the same way as explained for **cpu_limit**.

rss_limit = *hardlimit*

Specifies the maximum resident set size, expressed in the same format as the **data_limit**. The same shorthand strings that were described for **cpu_limit** may be used to specify infinity or to copy limits from the submitting machine. The *hardlimit* parameter is interpreted the same way as explained for **cpu_limit**.

Examples of Class Stanzas

Example 1: Creating a Class that Excludes Certain Users:

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

Example 2: Creating a Class for Small-Size Jobs:

```
small:  type=class                 # class for small jobs
        priority=80                # ClassSysprio (max=100)
        cpu_limit=00:02:00         # 2 minute limit
        data_limit=30mb            # max 30 MB data segment
        core_limit=10mb            # max 10 MB core file
        file_limit=50mb            # max file size 50 MB
        stack_limit=10mb           # max stack size 10 MB
        rss_limit=35mb             # max resident set size 35 MB
        include_users = bob sally  # authorized users
```

Example 3: Creating a Class for Medium-Size Jobs:

```
medium: type=class              # class for medium jobs
        priority=70             # ClassSysprio
        cpu_limit=04:00:00      # 4 hour run time limit
        data_limit=80mb         # max 80 MB data segment
        core_limit=30mb         # max 30 MB core file
        file_limit=80mb         # max file size 80 MB
        stack_limit=30mb        # max stack size 30 MB
        rss_limit=100mb         # max resident set size 100 MB
        job_cpu_limit=1800,1200 # hard limit is 30 minutes,
                                 # soft limit is 20 minutes
```

Example 4: Creating a Class for Large-Size Jobs:

```
large:  type=class              # class for large jobs
        priority=60             # ClassSysprio
        cpu_limit=unlimited     # no run time limit
        data_limit=120mb        # max 120 MB data segment
        core_limit=30mb         # max 30 MB core file
        file_limit=120mb        # max file size 120 MB
        stack_limit=unlimited   # unlimited stack size
        rss_limit=150mb         # max resident set size 150 MB
        job_cpu_limit = 3600, 2700 # hard limit 60 minutes
                                    # soft limit 45 minutes
```

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

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

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

Example 6: Creating a Class for PVM Jobs:

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

Example 7: Creating a Class for Master Node Machines:

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

Step 4: Specify Group Stanzas

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

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

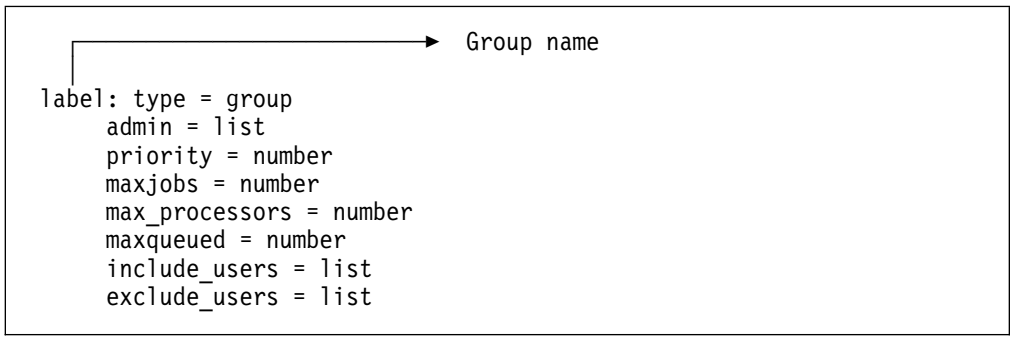


Figure 9. Group Stanzas

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

priority = *number*
where *number* can be any integer. The higher priority numbers result in a better job dispatch order. If the group stanza does not specify a priority or if there is no priority at all, the priority is defined in the default group stanza. If no value is found, the default is zero. The number specified for priority is referenced as **GroupSysprio** in the configuration file. **GroupSysprio** can be used in the assignment of job priorities. If the variable **GroupSysprio** does not appear in the SYSPRIO expression in the configuration file, the priority numbers for group specified here in the administration file have no effect. See "Step 5: Prioritize the Queue Maintained by the Negotiator" on page 47 for more information about the **GroupSysprio** keyword.

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

stanza at all, the maximum number of jobs that can be simultaneously run the group is defined in the default stanza. If no value is found, or the limit found is -1, then no limit is placed on the number of jobs that can be simultaneously run for the group. Regardless of the limit set to running jobs, there is no limit to the number of jobs that a group can submit.

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

maxqueued = *number*
where *number* is the maximum number of jobs allowed in the queue for that group. This prevents groups from flooding the job queue. If no **maxqueued** is specified in the group stanza, or if there is no group stanza, the maximum number of jobs that can simultaneously be in the queue is defined in the default stanza. If no value is found, or the limit found is -1, then no limit is placed on the number of jobs that can simultaneously be in the job queue for that group. Regardless of the limit set to the number of jobs queued, there is no limit to the number of jobs a group can submit.

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

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

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

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

Examples of Group Stanzas

Example 1: In this example, the group name is **department_a**. The jobs issued by users belonging to this group will have a priority of 80. There are three members in this group.

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

Example 2: In this example, the group called **group_a** has five members and these user's jobs have a priority of 100:

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

Configuring LoadLeveler

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

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

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

- **loadl** is the LoadLeveler user ID and the LoadLeveler group ID.
- The home directory of **loadl** is the configuration directory.
- **LoadL_config** is the name of the configuration file.

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

LoadLUserid Specifies the LoadLeveler user ID.

LoadLGroupid Specifies the LoadLeveler group ID.

LoadLConfig Specifies the full path name of the configuration file.

Note that if you change the LoadLeveler user ID to something other than **loadl**, you will have to make sure you have your configuration files on this ID. For more information on **loadl**, see “The LoadLeveler User ID and Group ID” on page 16.

You can also override the **/etc/LoadL.cfg** file. For an example of when you might want to do this, see “Querying Multiple LoadLeveler Clusters” on page 61.

The Configuration Files

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

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

- **Global Configuration File**

  This file is called the **LoadL_config** file and it contains configuration information common to all nodes in the LoadLeveler pool.

- **Local Configuration File**

This book refers to this file as the **LoadL_config.local** file but you can name your local configuration files by some other name. This file contains specific configuration information for an individual node. The **LoadL_config.local** file is in the same format as **LoadL_config** and the information in this file overrides any information specified in **LoadL_config**. It is an optional file that you use to modify information on a local machine. Its full pathname is specified in the **LoadL_config** file by using the **LOCAL_CONFIG** keyword. See "Step 9: Specify Where Files and Directories are Located" on page 51 for more information. "Customizing the Global and Local Configuration Files" on page 41 describes how to tailor this file to suit your needs. Appendix C, "Sample Files" on page 199 contains a sample configuration file.

LoadLeveler assumes a common name space. This means that a user must have the same user ID and login group ID on all machines. For example, if there are two machines in your LoadLeveler cluster, *machine_1* and *machine_2*, user `john` must have the same user ID and login group ID in the **/etc/passwd** file on both machines. If user `john` has user ID 1234 and login group ID 100 on *machine_1*, then user `john` must have the same user ID and login group ID in **/etc/passwd** on *machine_2*. This ensures that the **getuid** system call returns the same user ID on both systems. This common name space is important because it ensures that the job that gets submitted gets run with the same group ID and user ID of the person who submitted the job.

Configuration File Structure and Syntax

The information in both the **LoadL_config** and the **LoadL_config.local** files is in the form of a statement. These statements are made up of *keywords* and *values*. There are three types of configuration file keywords:

- Keywords, described in "Customizing the Global and Local Configuration Files" on page 41
- User-defined keywords, described in "User-Defined Keywords" on page 36.
- Reserved keywords and macros, described in "Reserved Keywords" on page 37

Configuration file statements take one of the following formats:

```
keyword=value
keyword:value
```

Statements that take the following format:

```
keyword=value
```

are used primarily to customize an environment.

Statements that take the following format:

```
keyword:value
```

are used by LoadLeveler to characterize the machine and are known as part of the *machine description*. Every machine in LoadLeveler has its own machine description which is read by the central manager when LoadLeveler is started.

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

User-Defined Keywords

This type of keyword, created and defined by the user, can be named using any combination of letters and numbers. A user-defined keyword is set equal to values where the *value* defines conditions, names files, or sets numeric values. For example, you can create a keyword named **MY_MACHINE** and set it to be equal to the name of your machine named *iron* as follows:

```
MY_MACHINE = iron.ore.met.com
```

User-defined keywords are identified by a dollar sign ($) and enclosed in parenthesis. Any use of the literal **$(MY_MACHINE)** following the definition in the previous example would result in the automatic substitution of **iron.ore.met.com** in place of **$(MY_MACHINE)**.

User-defined definitions may contain references, enclosed in parentheses, to previously defined user-defined keywords, therefore:

```
A = xxx
C = $(A)
```

is a valid expression and the resulting value of **C** is *xxx*. Note that **C** is actually bound to **A**, not to its value, so that

```
A = xxx
C = $(A)
A = yyy
```

is also legal and the resulting value of **C** is *yyy*.

The following are examples of user-defined keywords. These keywords appear in the sample configuration file found in the appendix:

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

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

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

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

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

HighLoad = 1.5
defines the variable **HighLoad** and assigns to it the floating point constant 1.5. This might be used as a saturation level at which no further jobs should be started and assumes only running 1 job on the machine.

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

(MAX_STARTERS * **JobLoad**) + **BackgroundLoad**

LowLoad = $(BackgroundLoad)
defines the variable **LowLoad** and assigns to it the value of **BackgroundLoad**. This might be used as a restart level at which jobs can be started again and assumes only running 1 job on the machine.

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

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

MAIL = *program*
where *program* specifies the path name of a local program you want to use to override the LoadLeveler default mail notification method. You can use this program to, for example, display your own messages to users when a job completes, or to automate tasks such as sending error messages to a network manager. The syntax for the program is the same as it is for standard UNIX mail programs; the command is called with a list of users as arguments, and the mail message is taken from standard input.

KeyboardBusy = KeyboardIdle < (15 * $(MINUTE))
defines the variable **KeyboardBusy** and reassigns to it at each evaluation the Boolean value **true** or **false**, depending on whether the keyboard and mouse have been idle for fifteen minutes.

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

CPU_Busy = LoadAvg >= $(HighLoad)
defines the variable **CPU_Busy** and reassigns to it at each evaluation the Boolean value **true** or **false** depending on whether the Berkeley one-minute load average is equal to or greater than the saturation level of $(HighLoad).

See the configuration files in Appendix C, “Sample Files” on page 199 for additional examples.

Reserved Keywords

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

- Numerical and alphabetic constants
- Mathematical operators
- Reserved macros

Numerical and Alphabetical Constants

Constants may be represented in the following values:

- Boolean expressions that are evaluated as true or false
- Signed integers
- Floating point values
- Strings enclosed in double quotes (" ").

Mathematical Operators

You can use the following C operators in your control function expressions. The operators are listed in order of precedence. All of these operators are evaluated from left to right:

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

Reserved Macros

Reserved macros are evaluated by the LoadLeveler daemons at various stages. LoadLeveler provides the following reserved macros that you can use in your statements:

Cpus
the number of CPU's installed.

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

Disk
the free disk in kilobytes on the filesystem where the executables of the LoadLeveler jobs assigned to this machine are stored. This refers to the file system that is defined by Execute keyword.

domain or **domainname**
dynamically indicates the official name of the domain of the current host machine where the program is running. Whenever a machine name can be specified or one is assumed, a domain name is assigned if none is present. For example, if a job is submitted to host machine **silver** and the domain of the submitting machine is **ore.met.com** and the domain for the machine is not specified elsewhere, then **$(domain)** would have the value **ore.met.com**.

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

host or **hostname**
dynamically indicates the official name of the host machine where the program is running. **host** provides you with the name without the domain name whereas, **hostname** provides you with the domain name.

For example, if a job is submitted to host machine **gold**, then **$(host)** would have the value **gold**.

Initiators
the job initiators both available and in use. Every machine has initiators and supported classes associated with it. The initiator identifies all of the classes that can be run on the particular machine. The class statement in the configuration file defines the classes of jobs that a machine can run. One class

statement can contain multiple initiators of identical or different values. For more information on the class statement, refer to “Step 3: Define LoadLeveler Machine Characteristics” on page 44.

KeyboardIdle
the number of seconds since the keyboard or mouse was last used. It also includes any telnet or interactive activity from any remote machine.

LoadAvg
A measure of the CPU load on the system. The load average is the average of the number of processes ready to run or waiting for disk I/O to complete. LoadLeveler determines this average every 60 seconds. The load average does not map to CPU time.

Machine
the name of the current machine.

Memory
the physical memory installed on the machine in megabytes.

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

State
the state of the **startd**.

Status
the status of the job. This is an integer code corresponding to Idle (I), Running (R), Completed (C), Removed (RM), Remove Pending (RP), Starting (ST), Not Running (NR), Pending (P), System hold (S), user Hold (H), System and user Hold (HS), or Deferred (D).

tilde
the home directory for the LoadLeveler userid.

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

VirtualMemory
the size of available swap space on the machine in kilobytes.

OpSys : $(OPSYS)
indicates the operating system on the host where the program is running. This value is automatically determined and need not be defined in the configuration file.

Arch : $(ARCH)
indicates the system architecture.

Machine : $(hostname)
indicates the name of the current host.

String-valued functions are provided for programs that may want to utilize these control functions to evaluate them.

Time:

Note: You can use the following time macros in the START, SUSPEND, CONTINUE, VACATE, and KILL expressions. If you use these macros in the START expression and you are operating across multiple time zones, unexpected results may occur. This is because the **negotiator** daemon evaluates the START expressions and this evaluation is done in the time zone in which the negotiator resides. Your executing machine also evaluates the START expression and if your executing machine is in a different time zone, the results you may receive may be inconsistent. To prevent this inconsistency from occurring, ensure that both your **negotiator** daemon and your executing machine are in the same time zone.

tm_hour
the number of hours since midnight (0-23).

tm_min
number of minutes after the hour (0-59).

tm_sec
number of seconds after the minute (0-59).

tm_isdst
Daylight Savings Time flag: positive when in effect, zero when not in effect, negative when information is unavailable.

Date:

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

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

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

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

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

These macros take on the type of the expression to which they are bound. They do not require any special characters (parenthesis or dollar sign) to identify them when invoked. Therefore, the **LoadAvg** macro is represented as **LoadAvg** when used in an expression.

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

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

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

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

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

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

Customizing the Global and Local Configuration Files

You can customize the global configuration file, called **LoadL_config**, to specify the common information across all nodes in the LoadLeveler pool. To override the specifications made in this file, you can modify the local configuration file, called **LoadL_config.local** file.

There are many parameters in the configuration file that you can set or you can accept the values provided for you in the global configuration file delivered with LoadLeveler.

To assist you in modifying the parameters, this section is presented in a step-by-step manner that walks you through the customization process. You do not have to perform the steps in the order that they appear here.

Step 1	Define LoadLeveler administrators
Step 2	Manage a job's status using expressions
Step 3	Define LoadLeveler machine characteristics
Step 4	Control the number of jobs that can run on a machine
Step 5	Prioritize the queue maintained by the negotiator
Step 6	Prioritize the order of executing machines maintained by the negotiator
Step 7	Define job accounting
Step 8	Specify alternate central managers
Step 9	Specify where files and directories are located
Step 10	Record and control log files
Step 11	Specify TCP/IP port number
Step 12	Write prolog and epilog programs
Step 13	Enable checkpointing

Step 1: Define LoadLeveler Administrators

LOADL_ADMIN = *list of user name(s) (required)*

where *user name(s)* is a blank-delimited list of those individuals who will have administrative authority. These users are able to invoke the administrator-only commands such as **llctl**, **llfavorjob**, and **llfavoruser**. They can also invoke the administrator-only GUI functions. For more information, see Chapter 9, “Administrative Uses for the Graphical User Interface” on page 165.

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

An administrator on a machine is granted administrative privileges on that machine. It does not grant him administrative privileges on other machines. To be an administrator on all machines in the LoadLeveler pool either specify your

userid in the global configuration file with no entries in the local configuration file or specify your userid in every local configuration file that exists in the LoadLeveler pool.

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

```
LOADL_ADMIN = bob mary
```

Step 2: Manage a Job's Status Using Control Functions

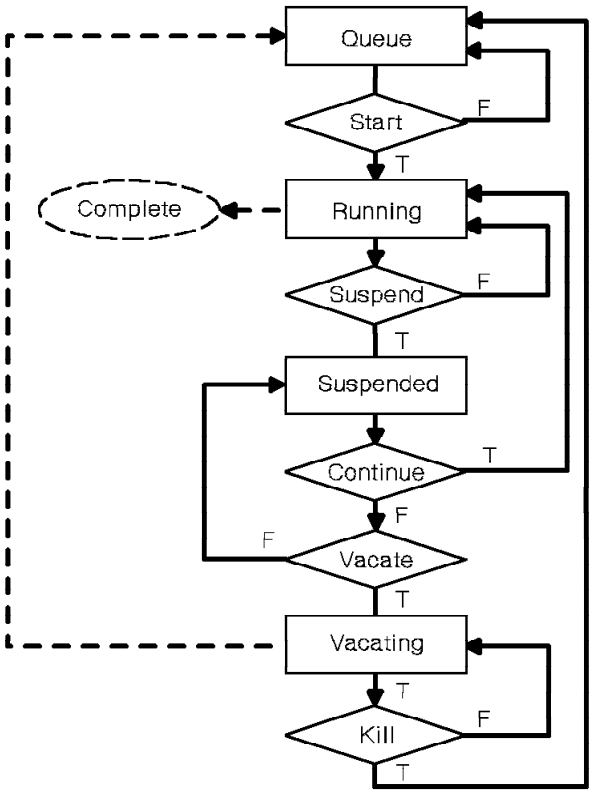
"How Does a Job's Status Change as it is Processed?" on page 7 describes the different job states. You can control when a job enters into a state by using five control functions that you can set to Boolean expressions in the configuration file. These conditions are defined, following normal C conventions, in the following expressions:

- START
- SUSPEND
- CONTINUE
- VACATE
- KILL

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

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

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



START: *expression that evaluates to T or F (true or false)*
Determines whether a machine can run a LoadLeveler job. When the expression evaluates to **T**, LoadLeveler considers dispatching a job to the machine.

SUSPEND: *expression that evaluates to T or F (true or false)*
Determines whether running jobs should be suspended. When **T**, LoadLeveler temporarily suspends jobs currently running on the machine. Suspended LoadLeveler jobs will either be continued or vacated.

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

VACATE: *expression that evaluates to T or F (true or false)*
Determines whether suspended jobs should be vacated. When **T**, suspended LoadLeveler jobs are removed from the machine and placed back into the queue. If a checkpoint was taken, the job restarts from the checkpoint. Otherwise, the job restarts from the beginning.

KILL: *expression that evaluates to T or F (true or false)*
Determines whether or not vacated jobs should be killed and replaced in the queue. It is used to remove a job that is taking too long to vacate. When **T**, vacated LoadLeveler jobs are removed from the machine with no attempt to take checkpoints.

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

Step 3: Define LoadLeveler Machine Characteristics

To define how machines in the LoadLeveler pool will be used, you need to specify machine description macros and machine description control functions. The machine description macros follow:

ARCH = *string (required)*
Indicates the standard architecture of the system. This definition must be consistent among machines of the same type and must be executed the same in the requirements or preferences statements in job command files. The administrator defines the character string for each architecture.

For example, to define a machine as being a RISC System/6000, the macro would look like:

ARCH = R6000

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

ARCH = SPARC

CUSTOM_METRIC = *number*
It specifies an arbitrary number which you can use in the MACHPRIO expression. If you specify neither CUSTOM_METRIC nor CUSTOM_METRIC_COMMAND, CUSTOM_METRIC = 1 is assumed.

CUSTOM_METRIC_COMMAND = *command*
Specifies an executable and any required arguments. The exit code of this command is assigned to CUSTOM_METRIC. If this command does not exit normally, CUSTOM_METRIC is assigned 1. This command is forked every POLLING_FREQUENCY. For more information, refer to Chapter 4, “Interactive Login Sessions” on page 89.

MACHINE_AUTHENTICATE = true|**false**
Specifies whether machine validation is performed. When set to **true**, LoadLeveler only accepts connections from machines specified in the administration file. When set to **false**, LoadLeveler accepts connections from any machine.

MAX_STARTERS = *number (required)*
Specifies the maximum number of job steps that can run simultaneously on a machine. If not specified, the default is the number of elements in the class set. In the parallel context, MAX_STARTERS is the number of jobs that can be initiated from a **startd**.

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

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

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

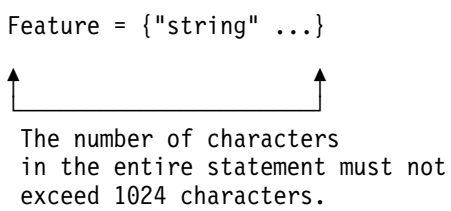
STARTD_RUNS_HERE = true|false
Specifies whether this daemon will run on the host. If you do not want to run this daemon, specify **false**.

X_RUNS_HERE = true|false
Set **X_RUNS_HERE** to **true** if you want to start the keyboard daemon (unles you are running on Sun machine or an HP machine).

Feature = *{"string" ...}* *(optional)*
where *string* is the characteristic to use to match jobs with machines.

You can specify unique characteristics for any machine using this keyword. When evaluating job submissions, LoadLeveler compares any required features specified in the job command file to those specified using this keyword.

You can have a maximum of 1024 characters in the feature statement. Refer to the following diagram for more information:



Example – Using the Feature Keyword

If a machine has licenses for products ABC and XYZ, in the local configuration file, enter the following:

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

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

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

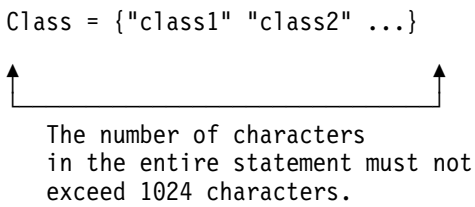
Class = {"*class1***" "***class2***" ...}** | **{"No_Class"}**
where "*class1*" "*class2*" *...* is a blank delimited list of class names. This keyword determines whether a machine will accept jobs of a certain job class.

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

In order for a LoadLeveler job to run on a machine, the machine must have a vacancy for the class of that job. If the machine is configured for only one **No_Class** job and a LoadLeveler job is already running there, then no further LoadLeveler jobs are started on that machine until the current job completes.

Note that you can change the default by modifying the default values specified in the configuration and administration files.

You can have a maximum of 1024 characters in the class statement. Refer to the following diagram for more information:



You can assign multiple classes to the same machine by specifying the classes in the LoadLeveler configuration file (called **LoadL_config**) or in the local configuration file (called **LoadL_config.local**). The classes, themselves, can be identified in the administration file. See "Step 3: Specify Class Stanzas" on page 27 for more information.

Example 1: This example defines the default class:

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

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

Example 2: This example specifies multiple classes. The machine will only run jobs that have either defaulted to, or explicitly requested class **No_Class**. A maximum of two LoadLeveler jobs are permitted to run simultaneously on the machine if MAX_STARTERS is not specified.

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

Example 3: This example specifies multiple classes. The machine will only run a maximum of four LoadLeveler jobs that have either defaulted to, or explicitly requested **No_Class**, **Small**, **Medium**, or **Large** class. A LoadLeveler job with class **IO_bound**, for example, would not be eligible to run here.

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

Example 4: This example specifies multiple classes. The machine will run only LoadLeveler jobs that have explicitly requested class **B** or **D**. Up to three LoadLeveler jobs may run simultaneously: two of class **B** and one of class **D**. A LoadLeveler job with class **No_Class**, for example, would not be eligible to run here.

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

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

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

For more information on classes, refer to "Job Classes" on page 5.

Step 4: Specify How Many Jobs a Machine Can Run

You can specify how many jobs a machine can run in the following ways:

- Specify MAX_STARTERS keyword
- Specify a class statement in the configuration file. This method was already discussed in the last step.

This section discusses how to use the MAX_STARTERS keyword. The syntax is:

MAX_STARTERS = *number*
where *number* defines the maximum number of job steps a machine can run.

- If this keyword is specified without specifying a class statement in the configuration file, this keyword specifies the maximum number of jobs a machine can run. In the following example, the maximum number of jobs that a machine can run is two.

  ```
  MAX_STARTERS = 2
  ```

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

  ```
  MAX_STARTERS = 2
  Class = { "class_a" }
  ```

  then the maximum number of jobs that can be run is one.

- If this keyword is not defined in either the global configuration file or the local configuration file, the maximum number of jobs that the machine can run is equal to the number of classes in the class statement.

For example, if the configuration file contained these statements:

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

the machine would run a maximum of two LoadLeveler jobs simultaneously. In the previous example, the possible combinations of LoadLeveler jobs are:

- A and B
- A and C
- B and B
- B and C
- A, B, or C, running alone.

Step 5: Prioritize the Queue Maintained by the Negotiator

Each job submitted to LoadLeveler is assigned a system priority number, based on the evaluation of the **SYSPRIO** keyword expression in the configuration file. The LoadLeveler system priority number is assigned when the central manager adds the new job to the queue of jobs eligible for dispatch. Once assigned, the LoadLeveler system priority number for a job is never changed (unless jobs for a user swap their **SYSPRIO**). Jobs assigned higher **SYSPRIO** numbers are considered for dispatch before jobs with lower numbers. Refer to “Job Priorities” on page 5 for more information on job priorities.

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

QDate The difference in the UNIX date that the job entered the queue and the UNIX date when **LoadL_negotiator** starts up.

UserSysprio The priority of the user who submitted the job, defined in the user stanza in the administration file. The default is 0.

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

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

UserPrio The user defined priority of the job, specified in the job command file with the **user_priority** keyword. The default is 50.

UserRunningJobs The number of jobs currently running for the user.

GroupRunningJobs the number of jobs currently running for the LoadLeveler group.

It is not recommended that you use **UserPrio** in the **SYSPRIO** expression because jobs for a user are already ordered by **UserPrio** and using **UserPrio** in the **SYSPRIO** expression would be redundant.

You can use the **UserRunningJobs** and **GroupRunningJobs** parameters to prioritize the queue based on current usage. You need to set the **negotiator** parameter **NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL** so that the priorities are adjusted according to current usage rather than usage at submission time.

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

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

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

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

Example 3: This example orders the queue based on the number of jobs a user is currently running. The user who has the fewest jobs running is first in the queue:

```
SYSPRIO : 0 - UserRunningJobs
```

Step 6: Prioritize the Order of Executing Machines Maintained by the Negotiator

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

You can use the **LoadAvg**, **Cpus**, **Speed**, **Memory**, **VirtualMemory**, **Disk**, and **CustomMetric** keywords to define the **MACHPRIO** keyword expression:

LoadAvg the Berkeley one-minute load average of the machine, reported by **LoadL_Startd**.

Cpus the number of active processors of the machine, reported by **LoadL_Startd**.

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

Memory the size of real memory in megabytes of the machine, reported by **LoadL_Startd**.

VirtualMemory the size of available swap space in kilobytes of the machine, reported by **LoadL_Startd**.

Disk the size of free disk space in kilobytes on the filesystem where the executables of jobs assigned to this machine are stored, reported by **LoadL_Startd**.

Example 1: This example orders machines by the Berkeley one-minute load average.

```
MACHPRIO : 0 - (LoadAvg)
```

Therefore, if LoadAvg equals .7, this example would read:

```
MACHPRIO : 0 - (.7)
```

The MACHPRIO would evaluate to -.7.

Example 2: This example orders machines by the Berkeley one-minute load average normalized for machine speed:

```
MACHPRIO : 0 - (1000 * (LoadAvg / (Cpus * Speed)))
```

Therefore, if LoadAvg equals .7, Cpus equals 1, and Speed equals 2, this example would read:

```
MACHPRIO : 0 - (1000 * (.7 / (1 * 2)))
```

This example further evaluates to:

```
MACHPRIO : 0 - (350)
```

The MACHPRIO would evaluate to -350.

Notice that if the speed of the machine were increased to 3, the equation would read:

```
MACHPRIO : 0 - (1000 * (.7 / (1 * 3)))
```

The MACHPRIO would evaluate to approximately -233. Therefore, as the speed of the machine increases, the MACHPRIO also increases.

Example 3: This example orders machines accounting for real memory and available swap space (remembering that Memory is in Mbytes and VirtualMemory is in Kbytes):

```
MACHPRIO : 0 - (10000 * (LoadAvg / (Cpus * Speed))) +
(10 * Memory) + (VirtualMemory / 1000)
```

Example 4: This example allows the administrator control over machine ordering:

```
MACHPRIO : CustomMetric
```

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

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

Note that MasterMachPriority is equal to 1 for nodes which are master nodes (those with **master_node_exclusive = true**). MasterMachPriority is equal to 0 for nodes which are not master nodes. Therefore, the expression in Example 5 gives the master nodes the highest priority. Assigning a high priority to master nodes

may help job scheduling performance for parallel jobs which require master node features.

Step 7: Define Job Accounting

LoadLeveler provides accounting information on completed LoadLeveler jobs. For detailed information on this function, refer to Chapter 5, “Gathering Job Accounting Data” on page 107.

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

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

A_ON	Turns accounting data recording on. If specified without the A_DETAIL flag, total resource consumption for each job step is recorded.
A_OFF	Turns accounting data recording off. This is the default.
A_VALIDATE	Turns accounting validation on.
A_DETAIL	Enables extended accounting. Using this flag causes LoadLeveler to record detail resource consumption by machine and by events for each job step. This flag also enables the -x flag of the **llq** command, permitting users to view resource consumption for active jobs.

For example:

```
ACCT = A_ON A_DETAIL
```

This example specifies that accounting should be turned on and that extended accounting data should be collected and that the -x flag of the **llq** command be enabled.

ACCT_VALIDATION = *$(RELEASEDIR)/llacctval (optional)*
Keyword used to identify the module that will be called to perform account validation. An installation may choose to replace the **llacctval** module with its own accounting routine by specifying the module in this keyword.

GLOBAL_HISTORY = *$(SPOOL) (optional)*
Keyword used to identify the directory that will contain the global history files produced by **llacctmrg** command when no directory is specified as a command argument.

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

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

Step 8: Specify Alternate Central Managers

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

If you selected machines to serve as alternate central managers in the machine stanza(s) of your administration file, then you should set the following keywords in the configuration file.

CENTRAL_MANAGER_HEARTBEAT_INTERVAL = *number*
where *number* is the amount of time in seconds that defines how frequently primary and alternate central managers communicate with each other.

The default is 300 seconds or 5 minutes.

CENTRAL_MANAGER_TIMEOUT = *number*
where *number* is the number of heartbeat intervals that defines how long an alternate central manager will wait without hearing from the primary central manager before declaring that the primary central manager is not operating.

The default is 6.

The following example defines how long the alternate central manager will wait:

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

For more information on central manager backup, refer to Appendix A, “Commonly Asked Questions and Answers” on page 173.

For information on selecting alternate central managers, refer to “Step 1: Specify Machine Stanzas” on page 20

Step 9: Specify Where Files and Directories are Located

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

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

Local Configuration File	**LOCAL_CONFIG =** *pathname*
	defines the pathname of the optional local configuration file containing information specific to a node in the LoadLeveler network. If you are using a distributed file system like NFS, some examples are:
	`LOCAL_CONFIG = $(tilde)/$(host).LoadL_config.local`
	`LOCAL_CONFIG = $(tilde)/LoadL_config.$(host).$(domain)`
	`LOCAL_CONFIG = $(tilde)/LoadL_config.local.$(hostname)`
	If you are using a local file system, an example is:
	`LOCAL_CONFIG = /var/LoadL/LoadL_config.local`
	See “Reserved Macros” on page 38 for information about the **tilde**, **host**, and **domain** macros.

Local Directory	The following subdirectories reside in the local directory. It is possible that the local directory and LoadLeveler's home directory are the same. **EXECUTE =** *local directory* /**execute** *(required)* defines the local directory to store the executables of jobs submitted by other machines. **LOG =** *local directory* /**log** *(required)* defines the local directory to store log files. It is not necessary to keep all the log files created by the various LoadLeveler daemons and programs in one directory but you will probably find it convenient. **SPOOL =** *local directory* /**spool** *(required)* Defines the local directory where LoadLeveler keeps the local job queue and checkpoint files, as well as: **HISTORY = $(SPOOL)/history** *(required)* defines the pathname where a file containing the history of local LoadLeveler jobs is kept.
Release Directory	**RELEASEDIR =** *release directory (required)* defines the directory where all the LoadLeveler software resides. The following subdirectories are created during installation and they reside in the release directory. You can change their locations. **BIN = $(RELEASEDIR)/bin** *(required)* defines the directory where LoadLeveler binaries are kept. **LIB = $(RELEASEDIR)/lib** *(required)* defines the directory where LoadLeveler libraries are kept. **NQS_DIR =** *NQS directory (optional)* defines the directory where NQS commands **qsub**, **qstat**, and **qdel** reside. The default is /**usr**/**bin**.

Step 10: Record and Control Log Files

The LoadLeveler daemons and processes keep log files according to the specifications in the configuration file. A number of macros are used to describe where LoadLeveler maintains the logs and how much information is recorded in each log. These macros, shown in Table 2, are repeated in similar form to specify the pathname of the log file, its maximum length, and the debug flags to be used.

"Controlling Debugging Output" on page 54 describes the events that can be reported through logging controls.

Table 2. Log Control Statements

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

Notes:

1. When coding the *path* for the log files, it is not necessary that all LoadLeveler daemons keep their log files in the same directory, however, you will probably find it a convenient arrangement.
2. There is a maximum length, in bytes, beyond which the various log files cannot grow. Each file is allowed to grow to the specified length and is then saved to an **.old** file. The **.old** files are overwritten each time the log is saved, thus the maximum space devoted to logging for any one program will be twice the maximum length of its log file. The default length is 64KB.

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

   TRUNC_MASTER_LOG_ON_OPEN = true|**false**
   TRUNC_STARTD_LOG_ON_OPEN = true|**false**
   TRUNC_SCHEDD_LOG_ON_OPEN = true|**false**
   TRUNC_KBDD_LOG_ON_OPEN = true|**false**
   TRUNC_STARTER_LOG_ON_OPEN = true|**false**
   TRUNC_COLLECTOR_LOG_ON_OPEN = true|**false**
   TRUNC_NEGOTIATOR_LOG_ON_OPEN = true|**false**
3. LoadLeveler creates temporary log files used by the **starter** daemon. These files are used for synchronization purposes. When a job starts, a **StarterLog.***pid* file is created. When the job ends, this file is appended to the **StarterLog** file.
4. Normally, only those who are installing or debugging LoadLeveler will need to use the debug flags, described in "Controlling Debugging Output" on page 54. The default error logging, obtained by leaving the right side of the debug control statement null, will be sufficient for most installations.

Controlling Debugging Output

Flags control the level of debugging output logged by each program. The following flags are presented here for your information but are used primarily by IBM personnel for debugging purposes:

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

D_CKPT Logs various steps in the checkpointing process.
Logs calls to read and write by the xdr routines.

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

D_EXPR Logs steps in parsing and evaluating control expressions.

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

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

D_NQS Provides more information regarding the processing of NQS files.

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

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

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

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

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

D_SCHEDD Displays how the **schedd** works internally.

D_STARTD Displays how the **startd** works internally.

D_STARTER Displays how the **starter** works internally.

For example,

```
SCHEDD_DEBUG = D_CKPT  D_XDR
```

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

Step 11: Specify TCP/IP Port Numbers

The *port number* is an integer that specifies the port number to use to connect to the specified daemon. You can define these port numbers in the configuration file or the **/etc/services** file or you can accept the defaults. LoadLeveler first looks in the configuration file for these port numbers. If the port number is in the configuration file and is valid, this value is used. If it is an invalid value, the default value is used.

If LoadLeveler does not find the value in the configuration file, it looks in the **/etc/services** file. If the value is not found in this file, the default is used.

The keywords used in the configuration file for specifying the port numbers follow:

COLLECTOR_STREAM_PORT = *port number*
The default is 9612.

MASTER_STREAM_PORT = *port number*
The default is 9616.

NEGOTIATOR_STREAM_PORT = *port number*
The default is 9614.

SCHEDD_STREAM_PORT = *port number*
The default is 9605.

STARTD_STREAM_PORT = *port number*
The default is 9611.

COLLECTOR_DGRAM_PORT = *port number*
The default is 9613.

STARTD_DGRAM_PORT = *port number*
The default is 9615.

MASTER_DGRAM_PORT = *port number*
The default is 9617.

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

```
LoadL_negotiator 9614/tcp   # Negotiator port number
LoadL_collector  9612/tcp   # Collector port number
LoadL_master     9616/tcp   # Master port number
LoadL_schedd     9605/tcp   # Schedd port number
LoadL_startd     9611/tcp   # Startd port number
LoadL_collector  9613/udp   # Collector port number
LoadL_startd     9615/udp   # Startd port number
LoadL_master     9617/udp   # Master port number
```

Step 12: Write Prolog and Epilog Programs

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

Prolog and epilog programs fall into two categories: those that run as the LoadLeveler user ID, and those that run in a user's environment.

You can specify prolog and epilog programs, using the following keywords in the configuration file:

JOB_PROLOG = *pathname*
where *pathname* is the full path name of the prolog program. This program runs under the LoadLeveler user ID.

JOB_EPILOG = *pathname*
where *pathname* is the full path name of the epilog program. This program runs under the LoadLeveler user ID.

JOB_USER_PROLOG = *pathname*
where *pathname* is the full path name of the user prolog program. This program runs under the user's environment.

JOB_USER_EPILOG = *pathname*
where *pathname* is the full path name of the user epilog program. This program runs under the user's environment.

For security reasons, the **JOB_USER_PROLOG** and **JOB_USER_EPILOG** keywords must be coded on the machine where the jobs run and on the machine that schedules the job.

For more information on writing prolog and epilog programs, refer to "Prolog and Epilog APIs" on page 222.

Step 13: Enable Checkpointing

Checkpointing is a method of periodically saving the state of a job so that, if for some reason, the job does not complete, it can be restarted from the saved state. Users can checkpoint their serial jobs. They cannot, however, at this time, checkpoint their parallel jobs.

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

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

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

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

The LoadLeveler administrator can manually force jobs enabled for checkpointing on a machine to be checkpointed and vacated using the **llctl** flush command. Note that no new jobs are redispatched to this host until the **resume** option is issued on the **llctl** command. For more information on this command, refer to "llctl - LoadLeveler Control" on page 134.

Guidelines to Follow When Checkpointing Jobs

Review the following guidelines before submitting a checkpointing job:

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

At checkpoint time, a core dump of the application is created. A new user executable is created from the original executable and the core file. The original execut-

able may be small in size but the new executable can be much larger. This is because the new executable is made up from the original one and from sections of the core file.

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

```
# @ core_limit = unlimited, unlimited
                 Soft limit
                 Hard limit
```

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

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

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

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

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

How to Checkpoint a Job

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

To specify:	Do this:
Exactly when LoadLeveler will checkpoint your job.	Add the following option to your job command file: **checkpoint = user_initiated** You can also select this option on the Build a Job window of the GUI. To do this, code the LoadLeveler **ckpt()** function call in the FORTRAN or C program. Whenever this call is invoked, a checkpoint of the program will be taken.
That LoadLeveler automatically checkpoint your job.	Add the following option to your job command file: **checkpoint = system_initiated** You can also select this option on the Build a Job window of the GUI. For this type of checkpointing to work, system administrators must set certain frequencies in the configuration file to specify how often LoadLeveler would take a checkpoint of the job. The two frequencies are set with the following two keywords: **MIN_CKPT_INTERVAL =** *number* **MAX_CKPT_INTERVAL =** *number* where *number* specifies a period, in seconds, between checkpoints taken for running jobs. The time between checkpoints will be increased after each checkpoint within these limits as follows: • The first checkpoint is taken after a period of time equal to the **MIN_CKPT_INTERVAL** has passed. • The second checkpoint is taken after LoadLeveler waits *twice as long* (**MIN_CKPT_INTERVAL** X 2) • The third checkpoint is taken after LoadLeveler waits twice as long again (**MIN_CKPT_INTERVAL** X 4) before taking the third checkpoint. LoadLeveler continues to double this period until the value of **MAX_CKPT_INTERVAL** has been reached, where it stays for the remainder of the job. A minimum value of 900 (15 minutes) and a maximum value of 7200 (2 hours) are the defaults. The system administrator can set these keyword values globally in the global configuration file so that all machines in the cluster have the same value, or he can specify a different value for each machine by modifying the local configuration files. **Note:** If the job command files you used with previous releases of LoadLeveler state **checkpoint=yes**, they will default to **checkpoint=system_initiated**. **Note:** If you want to have both user initiated and system initiated checkpoint, you would select **system_initiated** as well. In this case, you have to code the **ckpt** function call and specify the keywords.
That your job not be checkpointed	Add the following option to your job command file: **checkpoint = no** You can also select this option on the Build a Job window of the GUI. This option is the default.

Ensure all User's Jobs are Linked to Checkpointing Libraries

All checkpointing jobs must be linked with the LoadLeveler libraries and LoadLeveler checks to make sure the job is linked correctly. Refer to the Memo to Users for any changes or updates to the following information.

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

llcc (for use with C)

llxlf (for use with FORTRAN)

C Example

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

FORTRAN Example

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

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

C Example

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

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

FORTRAN Example

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

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

C Example

Compiling:

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

Linking:

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

FORTRAN Example

Compiling:

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

Linking:

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

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

C Example

Compiling:

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

Linking:

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

FORTRAN Example

Compiling:

f77 -c test01.f

Linking:

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

Additional Administration Topics

This section describes some additional topics which may help your installation:

Querying Multiple LoadLeveler Clusters

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

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

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

You can configure /**etc**/**LoadL.cfg** to point to the “default” configuration files, and you can configure /**etc**/**othercluster.cfg** to point to the configuration files for other clusters which the user can select. The other clusters must be set up as submit-only machines.

For example, you can enter the following query command:

```
$ llq
```

The above command uses the configuration from /**etc**/**LoadL.cfg**. To send a query to the scheduler defined in the configuration file of **etc**/**othercluster.cfg**, enter:

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

If the *othercluster* configuration uses machine authentication checking, that machine must define the submitting machine as a submit-only client.

Setting Limits

A limit is the amount of a resource that a job process is allowed to use. The most common limit is called a CPU limit which limits the amount of CPU a single process can use. For example, if you set the CPU limit to five hours and you have a job composed of five processes, each process can use up to five hours of CPU time, for a total of 25 CPU hours. You can control the amount of CPU used by imposing another limit as well called the job CPU limit. This is the total amount of CPU that the entire job can use. For example, if the CPU limit is set to five hours and a job consists of five processes, the total amount of CPU that can be consumed is 25 hours. However, if you impose a job CPU limit of five hours the entire job (made up of all five processes) cannot consume more than five CPU hours.

If a job consists of a parent process and one or more children processes, only the CPU limit is enforced. In other words, in order for the job CPU limit to be enforced, the parent must wait for the children to terminate before the parent terminates.

The limits that you set in the administration file are used to override system limits. Limits can be set in both the administration file and in the job command file. In the administration file, you can only set hard limits. In the job command file, you can set both soft and hard limits. The *hardlimit* is the higher resource limit and the *softlimit* is the lower resource limit. In general, LoadLeveler handles these limits via the **setrlimit** system call. This system call sets hard and soft limits for a variety of system resources.

A limit pertains only to a single step in a multiple step job. In addition, each assigned processor in a parallel job is considered to be a separate job step for the purpose of limits. Therefore, the limit pertains to all the processes of a parallel job on a single processor, but does not support the sum CPU consumed by all the processors of a parallel job.

Note: Only **job_cpu_limit** is supported on the HP platform.

Setting Hard Limits: You can specify the hardlimit for the following keywords in either the class stanza of the administration file or in the job command file. These limits refer to each process of a job step. Keep in mind that each job step may be made up of several processes.

job_cpu_limit **cpu_limit**

data_limit **core_limit**

file_limit **stack_limit**

rss_limit

You can specify the softlimits for these keywords in the job command file but they cannot be specified in the class stanza of the administration file. Refer to Table 3 for more information on setting the hardlimit:

Table 3. Setting limits

If the hardlimit:	**Then the:**
Is set in both the class stanza and the job command file	Smaller limit is used.
Is not set in either the class stanza or the job command file	User limit set on the machine that runs the job is used.
Is less than the softlimit specified in the job command file	Softlimit is lowered to equal the hardlimit.
Is specified in the job command file	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. Note: If the limit is not defined in the administration file and the hardlimit defined by the user in the job command file is greater than the limit on the executing machine, then the hardlimit is set to the machine limit.

The syntax for setting a limit is:

```
limit_type = hardlimit, softlimit
```

For example:

```
core_limit = hardlimit, softlimit
```

To specify only a hardlimit, you would enter something like:

```
core_limit = 120kb
```

To specify only a softlimit, you would enter something like:

```
core_limit = ,120kb
```

For more information on these keywords, see “Step 3: Specify Class Stanzas” on page 27.

Exceeding Limits: What happens when your job reaches and exceeds either the softlimit or hardlimit depends upon the operating system you are using. For job_cpu_limits only, on processors which support the signal SIGXCPU, the operating system issues a SIGXCPU when the softlimit is reached and a SIGKILL when the hardlimit is reached. On systems that do not support SIGXCPU, LoadLeveler does not distinguish between hard and soft limits. When a softlimit is reached on these platforms, LoadLeveler issues a SIGKILL.

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

If the process reaches the hardlimit, depending upon the operating system, it may be terminated.

Summary of Process Specific Keywords

In the sample configuration files provided with LoadLeveler, certain keyword definitions are grouped according to the daemons or processes that use them. Many of these have already been described but are shown again here in the order they appear in the sample configuration file in Appendix C, "Sample Files" on page 199.

Kbdd Daemon Keywords

You can specify the following **kbdd** keywords in your configuration file:

KBDD = *pathname*
location of **kbdd** executable, **LoadL_kbdd**. See "The Kbdd Daemon" on page 217 for more information.

KBDD_LOG = *pathname*
location of log. See "Step 10: Record and Control Log Files" on page 53.

MAX_KBDD_LOG = *bytes*
limit of log size. See "Step 10: Record and Control Log Files" on page 53.

KBDD_DEBUG = *flags*
Error logging controls. See "Controlling Debugging Output" on page 54.

X_RUNS_HERE = true|**false**
Specifies whether the **kbdd** daemon runs on this host.

Startd Daemon Keywords

You can specify the following **startd** keywords in your configuration file:

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

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

STARTD = *pathname*
where *pathname* is the location of **startd** executable, **LoadL_startd**. See "The Startd Daemon" on page 215 for more information.

STARTD_LOG = *pathname*
location of log. See "Step 10: Record and Control Log Files" on page 53.

MAX_STARTD_LOG = *bytes*
limit of log size. See "Step 10: Record and Control Log Files" on page 53.

STARTD_DEBUG = *flags*
Error logging controls. See "Controlling Debugging Output" on page 54.

POLLING_FREQUENCY = *number*
where *number* specifies the frequency, in seconds, with which the **startd** daemon evaluates the load on the local machine and decides whether to suspend, resume, or abort jobs. On machines that run X (with AIX only) this is

also the minimum interval at which the **kbdd** daemon will report keyboard or mouse activity to the **startd** daemon. A value of 5 is the default.

POLLS_PER_UPDATE = *number*
where *number* specifies how often, in POLLING_FREQUENCY intervals, **startd** daemon updates the central manager. Due to the communication overhead, it is impractical to do this with the frequency defined by the **POLLING_FREQUENCY** keyword. Therefore, the **startd** daemon only updates the central manager every *n*th (where *n* is the number specified for POLLS_PER_UPDATE) local update. Change **POLLS_PER_UPDATE** when changing the **POLLING_FREQUENCY**. A value of 24 is the default.

JOB_ACCT_Q_POLICY = *number*
where *number* is the amount of time in seconds that determines how often the **startd** daemon updates the **schedd** daemon with accounting data of running jobs. This controls the accuracy of the **llq -x** command. The default is 300 seconds.

JOB_LIMIT_POLICY = *number*
where *number* is an amount of time in seconds. The smaller of JOB_LIMIT_POLICY or JOB_ACCT_Q_POLICY is used to control how often the **startd** daemon collects resource consumption data on running jobs and tests for job_cpu_limit. The default for JOB_LIMIT_POLICY is the POLLING_FREQUENCY * POLLS_PER_UPDATE.

Schedd Daemon Keywords

You can specify the following **schedd** daemon keywords in your configuration file:

SCHEDD = *pathname*
location of **schedd** executable, **LoadL_schedd**. See “The Schedd Daemon” on page 214 for more information.

SCHEDD_LOG = *pathname*
location of log. “Step 10: Record and Control Log Files” on page 53.

MAX_SCHEDD_LOG = *bytes*
limit of log size. See “Step 10: Record and Control Log Files” on page 53.

SCHEDD_DEBUG = *flags*
Error logging controls. See “Controlling Debugging Output” on page 54.

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

AFS_GETNEWTOKEN = *programname*
where *programname* is an administrator supplied program that, for example, can be used to refresh an AFS token. The default is to not run a program.

For more information, see “Handling an AFS Token” on page 236.

Collector Daemon Keywords

You can specify the following **collector** daemon keywords in your configuration file.

COLLECTOR = *pathname*
location of **collector** executable, **LoadL_collector**. See “The Collector Daemon” on page 217.

COLLECTOR_LOG = *pathname*
location of log. See “Step 10: Record and Control Log Files” on page 53.

MAX_COLLECTOR_LOG = *bytes*
limit of log size. See “Step 10: Record and Control Log Files” on page 53.

COLLECTOR_DEBUG = *flags*
Error logging controls. See “Controlling Debugging Output” on page 54.

CLIENT_TIMEOUT = *number*
where *number* specifies the maximum time, in seconds, that the **collector** waits to respond to a process over TCP/IP. If the time exceeds the specified amount, the **collector** stops waiting. This amount should be less than **MACHINE_UPDATE_INTERVAL**. The default is 30. The **CLIENT_TIMEOUT** value is used for all LoadLeveler daemons.

Negotiator Daemon Keywords

You can specify the following **negotiator** daemon keywords in your configuration file.

NEGOTIATOR = *pathname*
location of the **negotiator** executable, **LoadL_negotiator**. See “The Negotiator Daemon” on page 217.

NEGOTIATOR_LOG = *pathname*
location of the log. See “Step 10: Record and Control Log Files” on page 53.

MACHINE_UPDATE_INTERVAL = *number*
where *number* specifies the time period, in seconds, during which machines must report to the central manager. Machines that do not report in this number of seconds are considered *down*. The default is 300.

MAX_NEGOTIATOR_LOG = *bytes*
limit of log size. See “Step 10: Record and Control Log Files” on page 53.

NEGOTIATOR_DEBUG = *flags*
Error logging controls. See “Controlling Debugging Output” on page 54.

NEGOTIATOR_INTERVAL = *number*
where *number* specifies the interval, in seconds, at which the **negotiator** daemon updates the status of jobs in the LoadLeveler pool and negotiates with machines that are available to run jobs. The default is 60 seconds.

NEGOTIATOR_PARALLEL_DEFER = *number*
where *number* specifies the amount of time in seconds that defines how long a job stays out of the queue after it fails to get the correct number of processors. The default is NEGOTIATOR_INTERVAL multiplied by 5.

NEGOTIATOR_PARALLEL_HOLD = *number*
where *number* specifies the amount of time in seconds that defines how long a job is given to accumulate processors. The default is NEGOTIATOR_INTERVAL multiplied by 5.

NEGOTIATOR_REDRIVE_PENDING = *number*
where *number* specifies the amount of time in seconds that defines how long a job will remain pending before the **negotiator** resends the start job command to the **schedd**. The default is NEGOTIATOR_INTERVAL multiplied by 3.

NEGOTIATOR_RESCAN_QUEUE = *number*
where *number* specifies the amount of time in seconds that defines how long the **negotiator** waits to rescan the job queue for machines which have bypassed jobs which could not run due to conditions which may change over time, such as user at maxjobs running. The default is NEGOTIATOR_INTERVAL multiplied by 3.

NEGOTIATOR_REMOVE_COMPLETED = *number*
where *number* is the amount of time in seconds that you want the **negotiator** to keep information regarding completed and removed jobs so that you can query this information using the **llq** command. The default is 0 seconds.

NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL = *number*
where *number* specifies the amount of time in seconds between calculation of the **SYSPRIO** values for waiting jobs. A value of 0 (which is the default) means the **SYSPRIO** values are not recalculated. Recalculating the priority can be CPU-intensive; specifying low values for the **NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL** keyword may lead to a heavy CPU load on the **negotiator** if a large number of jobs are running.

You can use this keyword to base the order in which jobs are run on the current number of running jobs for a user or a group. For more information, see “Step 5: Prioritize the Queue Maintained by the Negotiator” on page 47.

Starter Process Keywords

You can specify the following **starter** process keywords in your configuration file.

STARTER = *pathname*
location of **starter** executable, **LoadL_starter**. See “The Starter Process” on page 216 for more information.

STARTER_LOG = *pathname*
Location of the log.

MAX_STARTER_LOG = *bytes*
limit of log size.

STARTER_DEBUG = *flags*
Error logging controls. See “Controlling Debugging Output” on page 54.

MIN_CKPT_INTERVAL = *number*
MAX_CKPT_INTERVAL = *number*
where *number* specifies a period, in seconds, between checkpoints taken for running jobs. The time between checkpoints will be increased after each checkpoint within these limits as follows:

- The first checkpoint is taken after a period of time equal to the **MIN_CKPT_INTERVAL** has passed.

- The second checkpoint is taken after LoadLeveler waits *twice as long* (**MIN_CKPT_INTERVAL** X 2)
- The third checkpoint is taken after LoadLeveler waits twice as long again (**MIN_CKPT_INTERVAL** X 4) before taking the third checkpoint.

LoadLeveler continues to double this period until the value of **MAX_CKPT_INTERVAL** has been reached, where it stays for the remainder of the job.

A minimum value of 900 (15 minutes) and a maximum value of 7200 (2 hours) are the defaults.

Master Daemon Keywords

You can specify the following **master daemon** keywords in your configuration file.

MASTER = *pathname*
location of **master** executable, **LoadL_master**. See "The Master Daemon" on page 213 for more information.

MASTER_LOG = *pathname*
location of log. See "Step 10: Record and Control Log Files" on page 53.

MAX_MASTER_LOG = *bytes*
limit of log size. See "Step 10: Record and Control Log Files" on page 53.

MASTER_DEBUG = *flags*
Error logging controls. See "Controlling Debugging Output" on page 54

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

OBITUARY_LOG_LENGTH = *number*
where *number* specifies the number of lines from the end of the file that are appended to the Master_Log. The Central Manager mails this log to the LoadLeveler administrators when one of the daemons dies. The default is 25.

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

Summary of Administration File Keywords

The following table contains a brief description of the keywords you can use in the administration file. This table serves only as a reference. These keywords are described in more detail throughout this chapter. (For a starting point, see "Customizing the Administration File" on page 20.)

Keyword	Stanza	Description
account	User and Group	A list of account numbers that identifies the account numbers a user may use when submitting jobs.
admin	Group and Class	A list of administrators for a group or class.
alias	Machine stanza	Lists one or more alias names to associate with the machine name.
central_manager	Machine	When **true**, this designates the machine as the LoadLeveler central manager.
class_comment	Class	Text characterizing the class
core_limit	Class	Specifies the maximum size core file a job can create.
cpu_limit	Class	Specifies the maximum CPU time to be used by the job.
cpu_speed_scale	Machine	Determines whether CPU time is normalized according to machine speed.
data_limit	Class	Specifies the maximum size data segment to use by the job.
default_class	User	A class name that is the default value assigned to jobs submitted by users for which no class statement appears.
default_group	User	A group name to which the user belongs.
exclude_groups	Class	A list of groups names identifying those who cannot submit jobs of a particular class.
exclude_users	Class and Group	A list of user names identifying those who cannot submit jobs of a particular class or who are not members of the group.
file_limit	Class	Specifies the maximum size of a file that a job can create.
include_groups	Class	A list of groups names identifying those who can submit jobs of a particular class.
include_users	Class and Group	A list of user names identifying those who can submit jobs of a particular class or who do belong to the group.
job_cpu_limit	Class	The maximum amount of cpu limit an individual job step can use per processor.
master_node_exclusive	Machine	When **true**, this machine is used only as a master node for parallel jobs.
master_node_requirement	Class	When **true**, jobs in this class have the requirement that they run on a master node having the **master_node_exclusive** setting.
maxjobs	Group and User	Maximum number of jobs this group or user can have running simultaneously.
max_jobs_scheduled	Machine	The maximum number of jobs that this machine can run.
max_processors	User, Class, and Group	Defines the maximum number of processors a user can request for a parallel job.
maxqueued	Group and User	Defines the maximum number of jobs a single group or user can have queued at the same time.
name_server	Machine	A list of nameservers used for a machine.
nice	Class	Increments the *nice* value of a job.
NQS_class	Class	When **true**, any job submitted to this class is routed to an NQS machine.
NQS_query	Class	A list of queue names to use to monitor and cancel jobs.

Keyword	Stanza	Description
NQS_submit	Class	A name that identifies the name of the NQS pipe queue to which the job will be routed.
priority	User, Class, and Group	A number that identifies the priority of the appropriate user, class, or group.
pvm_root	Machine	A directory in which PVM 3.3 is installed.
rm_host	Machine	Can be **true** or **false** and it defines whether or not the machine is an SP system machine which will be communication with the resource manager.
rss_limit	Class	Specifies the maximum resident set size for a job.
schedd_host	Machine	When **true**, this machine is used to help submit-only machines access LoadLeveler hosts that run LoadLeveler jobs.
speed	Machine	Defines the weight associated with the machine.
stack_limit	Class	Specifies the maximum size for a stack.
submit_only	Machine	When **true**, this designates this as a submit-only machine.
type	All	The type of stanza. Can be either *user*, *class*, or *machine*.

Summary of Configuration File Keywords and Macros

The following tables contain a brief description of the keywords you can use in the configuration file.

The term *configuration file keywords* refers to keywords, user-defined keywords, and macros. A summary table is provided for each of the three types of configuration file keywords.

Keywords

The following table serves only as a reference. These keywords are described in more detail in this chapter. (For a starting point, see "Customizing the Global and Local Configuration Files" on page 41.)

Keyword	Description
ACCT	Defines if the accounting function is turned on.
ACCT_VALIDATION	Identifies module that will be called to perform account validation.
ADMIN_FILE	Points to the administration file containing user, class, and machine list stanzas.
AFS_GETNEWTOKEN	The filter which can be used to renew an AFS token.
ARCH	Indicates the standard architecture of the system.
BIN	Defines the directory where LoadLeveler binaries are kept.
CENTRAL_MANAGER_HEARTBEAT_INTERVAL	The amount of time in seconds that defines how frequently primary and alternate central manager communicate with each other.
CENTRAL_MANAGER_TIMEOUT	The number of heartbeat intervals that defines how long an alternate central manager will wait before declaring that the primary central manager is not operating.
Class	Specifies what class of jobs may be run on the machine.
CLIENT_TIMEOUT	The maximum time, in seconds, that the **collector** waits to respond to a process over TCP/IP.

Keyword	Description
COLLECTOR	Location of the **collector** executable (**LoadL_collector**).
COLLECTOR_STREAM_PORT	Number that defines the port number to use when connecting to the daemon.
COLLECTOR_DGRAM_PORT	Number that defines the port number to use when connecting to the daemon.
CONTINUE	Continue expression. Determines if a job should continue.
CUSTOM_METRIC	Specifies a number which you can use in MACHPRIO expression.
CUSTOM_METRIC_COMMAND	Specifies an executable and any required arguments.
EXECUTE	Defines the local directory to store the executable checkpoints of jobs submitted by other machines.
Feature	A string specifying unique characteristics of a machine. When a job is submitted, LoadLeveler copmares these features to those specified in the job command file.
GLOBAL_HISTORY	Identifies the directory that will contain the global history files.
HISTORY	Defines the pathname where a file containing the history of local LoadLeveler jobs is kept.
JOB_ACCT_Q_POLICY	The amount of time in seconds that determines how often the **starts** daemon updates the **schedd** daemon with accounting data of running jobs.
JOB_EPILOG	Pathname of the epilog program.
JOB_LIMIT_POLICY	Defines the amount of time in seconds that determines how often LoadLeveler checks to see if job_cpu_limit has been exceeded.
JOB_PROLOG	Pathname of the prolog program.
JOB_USER_EPILOG	Pathname of the user epilog program.
JOB_USER_PROLOG	Pathname of the user prolog program.
KBDD	KBDD expression. Location of **kbdd** executable (**Loadl_kbdd**).
KILL	Kill expression. Determines if vacated jobs should be killed.
LIB	Defines the directory where LoadLeveler libraries are kept.
LOADL_ADMIN	Defines LoadLeveler administrators.
LOCAL_CONFIG	Defines the pathname of the optional local configuration file containing information specific to a node in the LoadLeveler network.
LOG	Defines the local directory to store log files.
MACHINE_AUTHENTICATE	Specifies whether machine validation is performed.
MACHINE_UPDATE_INTERVAL	The time, in seconds, during which machines must report to the central manager.
MACHPRIO	Machine priority expression
MASTER	Location of the **master** executable (**LoadL_master**).
MASTER_DGRAM_PORT	Number that defines the port number to use when connecting to the daemon.
MASTER_STREAM_PORT	Number that defines the port number to use when connecting to the daemon.
MAX_STARTERS	Specifies the maximum number of jobs that can run simultaneously.
MAX_CKPT_INTERVAL	The maximum number of seconds between checkpoints for running jobs.
MIN_CKPT_INTERVAL	The minimum number of seconds between checkpoints for running jobs.

Keyword	Description
NEGOTIATOR	Location of the **negotiator** executable (**LoadL_negotiator**).
NEGOTIATOR_INTERVAL	The time interval, in seconds, at which the **negotiator** daemon updates the status of jobs in the LoadLeveler pool and negotiates with machines that are available to run jobs.
NEGOTIATOR_PARALLEL_DEFER	The length of time that defines how long a job is given to accumulate processors.
NEGOTIATOR_PARALLEL_HOLD	The length of time a job attempts to collect machines before releasing them.
NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL	the amount of time in seconds between calculation of the **SYSPRIO** values for waiting jobs.
NEGOTIATOR_REDRIVE_PENDING	The length of time a job will remain in Pending state before it is redrived by the **negotiator**.
NEGOTIATOR_REMOVE_COMPLETED	The amount of time the negotiator keeps information on completed and removed jobs.
NEGOTIATOR_RESCAN_QUEUE	
Rescans job queue for machines that have temporarily non-runnable jobs.	
NEGOTIATOR_STREAM_PORT	Number that defines the port number to use when connecting to the daemon.
NQS_DIR	Defines the directory where NQS commands reside.
POLLING_FREQUENCY	The frequency in seconds the **startd** daemon uses to evaluate the load on the local machine and to decide whether to suspend, resume, or abort jobs.
OBITUARY_LOG_LENGTH	The number of lines from the ned of the file that are appended to the Master_Log.
POLLS_PER_UPDATE	The frequency, in POLLING_FREQUENCY intervals, with which the **startd** daemon updates the central manager.
PUBLISH_OBITUARIES	When set to **true**, this specifies that the **master** daemon send mail to the administrator(s) when any daemon it manages dies abnormally.
RELEASEDIR	Defines the directory where all the LoadLeveler software resides.
RESTARTS_PER_HOUR	The number of times the **master** daemon attempts to restart a daemon that dies abnormally.
SCHEDD	Location of the **schedd** executable (**LoadL_schedd**).
SCHEDD_RUNS_HERE	Specifies whether this daemon will run on the host.
SCHEDD_STREAM_PORT	Number that defines the port number to use when connecting to the daemon.
SPOOL	Defines the local directory where LoadLeveler keeps the local job queue and checkpoint files.
START	Start expression. Determines if a machine can run a job.
STARTD	Location of the **startd** executable (**LoadL_startd**).
STARTER	Location of the **starter** executable (**LoadL_starter**).
STARTD_RUNS_HERE	Specifies whether this daemon will run on the host.
START_DAEMONS	Specifies whether to start the daemons on the machine.
STARTD_DGRAM_PORT	Number that defines the port number to use when connecting to the daemon.
STARTD_STREAM_PORT	Number that defines the port number to use when connecting to the daemon.
SUBMIT_FILTER	Specifies the program you want to run to filter a job script when the job is submitted.

Keyword	Description
SUSPEND	Suspend expresson. Determines if a job should be suspended.
SYSPRIO	System priority expression
TRUNC_COLLECTOR_LOG_ON_OPEN	If **true**, it specifies the log file be started new with every invocation of the daemon.
TRUNC_KBDD_LOG_ON_OPEN	If **true**, it specifies the log file be started new with every invocation of the daemon.
TRUNC_MASTER_LOG_ON_OPEN	If **true**, it specifies the log file be started new with every invocation of the daemon.
TRUNC_NEGOTIATOR_LOG_ON_OPEN	If **true**, it specifies the log file be started new with every invocation of the daemon.
TRUNC_SCHEDD_LOG_ON_OPEN	If **true**, it specifies the log file be started new with every invocation of the daemon.
TRUNC_STARTD_LOG_ON_OPEN	If **true**, it specifies the log file be started new with every invocation of the daemon.
TRUNC_STARTER_LOG_ON_OPEN	If **true**, it specifies the log file be started new with every invocation of the daemon.
USE_X_IDLE_EXTENSION	If your X server supports the XGetIdleTime protocol extension, set this keyword to **true**. If you are using a Sun machine, ignore this keyword.
VACATE	The vacate expression. Determines whether suspended jobs should be vacated.
X_RUNS_HERE	Set **X_RUNS_HERE** to **true** if you want to start the keyboard daemon (unles you are running on Sun machine or an HP machine).

User-Defined Keywords

The following table serves only as a reference. These keywords are described in more detail in “User-Defined Keywords” on page 36.

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

Keyword	Description
mail	Specifies a local program you want to use in place of the LoadLeveler default mail notification method.
MINUTE	Defines the variable **MINUTE** and assigns to it a constant integer value.
StateTimer	Defines the variable **StateTimer** and reassigns to it at each evaluation the number of seconds since the current state was entered.

Reserved Macros

The following table serves only as a reference. These keywords are described in more detail in “Reserved Macros” on page 38.

Keyword	Description
ARCH or Arch	Indicates the standard architecture of the system.
ClassSysprio	The job priority for the class.
Cpus	number of CPU's installed.
CurrentTime	UNIX date that includes the current system time, in seconds, since January 1, 1970.
Disk	Free disk in megabytes on the filesystem where checkpoints are stored.
domain or domainname	Dynamically indicates the official name of the domain of the current host machine where the program is running.
EnteredCurrentState	Value of CurrentTime when the current state was entered.
GroupRunningJobs	The number of jobs currently running for the LoadLeveler group.
GroupSysprio	The job priority for the group.
host or hostname	Dynamically indicates the official name of the host machine where the program is running. **host** returns the name without the domain while **hostname** returns the fully qualified name.
initiators	Job initiators, both available and in use.
KeyboardIdle	Number of seconds since the keyboard or mouse was last used.
LoadAvg	Berkeley one-minute load average.
Machine	Name of the current machine.
Memory	Physical memory installed on the machine in megabytes.
OpSys	Indicates the operating system on the host where the program is running.
QDate	Difference in seconds between when LoadLeveler (specifically the negotiator daemon) comes up and when the job is submitted using llsubmit.
State	State of the **startd**. Can be "none", "busy", "running", "idle", "suspend", "flush", or "drain".
Status	Status of the job; an integer code corresponding to Idle (I), Running (R), Completed (C), Removed (X), Pending (P), System_hold (S), User_hold (U), or Deferred (D).
tilde	Dynamically defines the pathname of the LoadLeveler home directory.
tm_hour	Number of hours since midnight (0-23).
tm_isdst	Daylight Savings Time flag: positive when in effect, zero when not in effect, negative when information is unavailable.
tm_mday	Number of the day of the month (1-31).
tm_min	Number of minutes after the hour (0-59).
tm_mon	Number of months since January (0-11).
tm_sec	Number of seconds after the minute (0-59).
tm_wday	Number of days since Sunday (0-6).
tm_yday	Number of days since January 1 (0-365).

Keyword	Description
tm_year	Number of years since 1900 (0-99).
UserPrio	User defined priority of a job.
UserRunningJobs	The number of jobs currently running for the user.
UserSysprio	The priority of the user who submitted the job.
VirtualMemory	The size of the available swap space on the machine in kilobytes should be vacated.

Chapter 3. Submitting Parallel Jobs

Users can use LoadLeveler to schedule parallel batch jobs that have been written using Parallel Virtual Machine (PVM) 3.3 (a public domain package) or any general parallel programming language (such as the IBM Parallel Environment Library* 1.2.1 or the Parallel Virtual Machine Extended (PVMe)* 1.3.1) which uses the LoadLeveler parallel application programming interfaces. These interfaces can be used by any application to retrieve a list of machines from LoadLeveler and to start jobs using LoadLeveler's **startd** daemon. Refer to Appendix E, "LoadLeveler's Application Programming Interfaces" on page 219 for more information on these interfaces.

This chapter discusses what the system administrator needs to do to allow end users to submit parallel jobs. It also provides a sample scenario illustrating how to set up your environment to handle parallel batch job submission.

Migration Issues for Submitting Parallel Jobs

In this section, "LoadLeveler 1.2.x" refers to LoadLeveler release 1.2.0 or later.

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

- In previous releases of LoadLeveler, users could only submit parallel jobs to the RS/6000 Scalable POWERparallel (SP) system. In LoadLeveler 1.2.x, LoadLeveler can schedule parallel jobs to any combination of supported platforms as long as the platforms are at the same LoadLeveler release level and the parallel programming environment is supported on the platform.
- In previous releases of LoadLeveler, when submitting a parallel job to the SP, LoadLeveler did not select the parallel machines or start the parallel jobs. LoadLeveler reserved machines through the resource manager and started the partition manager. The partition manager would then pick up the reserved machines from the resource manager and start the parallel jobs. In LoadLeveler 1.2.x, the **negotiator** daemon selects the parallel machines and starts the parallel jobs. Therefore, it is a new requirement that all parallel machines run a **startd** daemon. It is no longer required that the resource manager be run at all. However you must run the resource manager if you want to use the high-performance switch in user mode or if you want the resource manager to coordinate machine usage between interactive and batch jobs on the SP system. Serial and parallel jobs may now be configured to run on the same machines.
- In previous releases of LoadLeveler, the LoadLeveler configuration file keyword **POOL_ID** was required. LoadLeveler used to query the resource manager for the number of available parallel machines in the POOL_ID. This number was then matched against the user parallel machine requirement. If enough machines were available, LoadLeveler would reserve the machines through the resource manager. In LoadLeveler 1.2.x, POOL_ID is not used to query the resource manager, therefore the POOL_ID keyword is no longer allowed.
- In previous releases of LoadLeveler, the keyword **parallelmachines** was required in the job command file in order to specify the number of parallel machines needed to run the job. In LoadLeveler 1.2.x, this keyword has been replaced with keywords in the following list. Note that not all of the keywords are required in every circumstance:

© Copyright IBM Corp. 1993, 1995

– **min_processors**

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

– **max_processors**

  Specifies the maximum number of processors requested for this parallel job. The default is **min_processors**. If you want to accept the default, you do not have to specify this keyword.

  The administrator may limit the number of processors that a user may request by specifying the **max_processors** keyword in the class stanza of the administration file.

– **job_type**

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

– **parallel_path**

  Specifies the path that should be used when starting a PVM 3.3 slave process. This is used for PVM 3.3 only. It will be used for the PVM 3.3 **ep** keyword.

– **requirements**

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

  - Adapter

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

  - Pool

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

- In LoadLeveler 1.2.x, the **negotiator** daemon selects the parallel machines through *temporary reservation*. This means that when a parallel job enters the system, the central manager scans a list of machines with resources matching the job requirements. If sufficient machines to schedule the parallel job are not available, the central manager temporarily reserves as many machines as it can. As machines meeting the job requirements become available, these machines are added to the list of reserved machines. When the machines requirement has been meet, the job is dispatched.

  It is possible that a sufficient number of machines never become available. Therefore, the central manager limits the amount of time the machines may be held in reserve. When this limit has been met, all reserved machines are released and the parallel job is placed on a deferred queue where it waits to reenter the LoadLeveler queue with its original system priority. How long a job

attempts to collect machines before releasing them and how long a job waits on the temporary queue are both configurable by specifying the following two keywords in the configuration file:

NEGOTIATOR_PARALLEL_DEFER
Specifies the amount of time in seconds that defines how long a job stays out of the queue after it fails to get the correct number of processors. The default is NEGOTIATOR_INTERVAL multiplied by 5.

NEGOTIATOR_PARALLEL_HOLD
Specifies the amount of time in seconds that defines how long a job is given to accumulate processors. The default is NEGOTIATOR_INTERVAL multiplied by 5.

LoadLeveler machines that are configured to run on the SP system with the resource manager will poll the resource manager to identify whether or not they have been dedicated to another client. If this is the case, the central manager will update the machine status to reserved and will not attempt to schedule any more jobs on that machine.

Restrictions and Limitations for Submitting Parallel Jobs

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

- Vacate, suspend, flush and checkpointing are not supported.
- PVMe 1.1.3 and PVM 3.3, dynamic allocation and de-allocation of parallel machines are not supported.
- The **negotiator** will allocate a machine once and only once per job, regardless of the number of classes available.
- PVMe 1.1.3 and PVM 3.3 should be restricted to one parallel class for each, per machine. This is mandatory for PVM 3.3 because PVM requires that each user be allowed to run only one instance of PVM per machine. Since PVMe 1.1.3 only runs over the high-performance switch in user mode, which is always allocated as dedicated, all other PVMe jobs will be blocked on those allocated machines.

Setting Up to Allow Users to Submit Parallel Jobs

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

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

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

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

As was described earlier, the process of submitting parallel jobs to the SP system differs in LoadLeveler 1.2.x from that of previous releases. To use the resource manager to schedule parallel jobs on the SP system, edit the LoadLeveler administration file and add the keyword **rm_host = true** to every SP system machine stanza. When you set **rm_host** to **true**, LoadLeveler does the following:

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

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

Checklist: Common Set Up Problems

This section presents a list of common problems found in setting up parallel jobs:

- Jobs that need resource manager adapters remain in an Idle state:
  - LoadL machines have **rm_host=false**
  - LoadL machines have **rm_host=true**:
    - LoadLeveler has not been globally reconfigured since you added **rm_host=true**
    - Resource manager is not running (use **jm_status -P** to check)
    - Machines are not specified as a general pool members or batch pool members in **jmd_config** (use **jm_status -P** to check)
- Jobs remain in a Pending state:
  - Nameserver is not consistent. Compare results of **host** *machine_name* and **host** *IP_address*

Setting Up to Allow Users to Submit PVM Jobs

If users will be submitting PVM jobs, your installation must first obtain and install PVM. PVM is a public domain package distributed through electronic mail by Oak Ridge National Labs. PVM must be at the 3.3 level or above. To obtain information on PVM, issue the following:

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

LoadLeveler expects to find PVM installed in ~**loadl**/**pvm3**. You can override this using the **pvm_root** entry in the machine stanza. The value of **pvm_root** is used to set the environment variable **$(PVM_ROOT)** which PVM requires. For example:

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

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

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

The following is an example of a class statement that you can add to your configuration file:

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

Note:

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

PVM Problem Determination

If LoadLeveler is to manage PVM jobs on a machine for a user, that user should not attempt to run PVM jobs on that machine outside of LoadLeveler control. Because of PVM restrictions, only a single PVM daemon per user per machine is permitted. If a user tries to run PVM jobs without using LoadLeveler and LoadLeveler later attempts to start a job for that user on the same machine, LoadLeveler may not be able to start PVM for the job. This will cause the LoadLeveler job to be cancelled.

If a PVM job submitted through LoadLeveler is rejected, it is probably because PVM was not correctly terminated the last time it ran on the rejecting machine. LoadLeveler attempts to handle this by making sure that it cleans up PVM jobs when they complete, but remember that you may need to clean up after the job yourself. If a machine refuses to start a PVM job, check the following:

- See if there is a process with the name **pvmd** running on the machine in question under the id of the user whose job will not start. Stop the process by issuing:

  ```
  ps -ef | grep pvmd
  kill -TERM <pid>
  ```

Do not use either of the following variations to stop the daemon because this will prevent **pvmd** from cleaning up and jobs will still not start:

```
kill -9 <pid>
kill -KILL <pid>
```

- If there is no **pvmd** process running, see if there is a file called **/tmp/pvmd.**<**userid**> where <userid> is the id of the user whose job will not start. If so, remove it.

Sample Parallel Job Submission Scenarios

The following sample scenario walks you through the process of submitting a parallel job. This scenario assumes the following:

- The name of the parallel job is `my_parallel_job`.
- The user wants to submit parallel jobs to the SP system only.

Task 1: Decide Where Parallel Jobs Will Run

In this example, the user only wants to submit parallel jobs to the SP system. To do this, follow these steps:

1. Edit the LoadLeveler administration file and add the keyword **rm_host = true** to every SP system machine stanza. For example, the following machine stanza illustrates the use of **rm_host**:

```
sp2_machine1: type = machine
              submit_only = false
              rm_host = true
```

Figure 10. SP Machine Stanza Specifying the rm_host Keyword

2. Edit the local LoadLeveler configuration files for every SP machine and add the parallel classes. For example:

```
class = {"Serial" "Serial" "POE" "POE" "PVMe" "PVM3"}
```

Note that PVMe and PVM3 are only listed once. It is highly recommended that you restrict both these classes to one case only per machine.

Task 2: Submit the Parallel Jobs Using llsubmit

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

- **min_processors**

  Specifies the minimum number of processors requested for this parallel job. The default is 1. The syntax is:

  ```
  min_processors = number
  ```

- **max_processors**

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

```
max_processors = number
```

- **job_type**

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

  ```
  job_type = string
  ```

  where:

  pvm3 For PVM 3.3 jobs. This is not for PVMe jobs.

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

  serial For serial jobs. This is the default.

- **parallel_path**

  Specifies the path that should be used when starting a PVM 3.3 slave process. This is used for PVM 3.3 only and will be used for the PVM 3.3 **ep** keyword.

  For example:

  ```
  parallel_path = /home/userid/cmds/pvm33/$PVM_ARCH
  ```

  $PVM_ARCH will be replaced by the architecture of the machine, as defined by PVM 3.3. This will specify the path to be searched for executables when the user's job issues a **pvm_spawn()** command.

- **requirements**

  List of requirements which a remote machine must meet to execute any job steps. The requirements specified for the job are the requirements for all of the machines. "requirements" on page 192 lists and describes all of the requirements supported. This section only lists the parallel requirements supported:

  – Adapter

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

    The syntax is:

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

    Where `adapter_name` can be one of the following:

    hps_ip Refers to the high-performance switch in IP mode.

    hps_user Refers to the high-performance switch in user mode.

    ethernet Refers to Ethernet. This is the default.

    fddi Refers to Fiber Distributed Data Interface (FDDI)

    tokenring Refers to Token Ring

    fcs Fiber Channel Standards

    For example:

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

  – Pool

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

```
Pool == number
```

For example:

```
Pool == 4
```

You can specify multiple requirements in a single **requirements** statement. The syntax is:

```
requirements = Boolean expression
```

For example,

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

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

Job Command File Examples: In these examples, the parallel job requires eight parallel machines to run but can run with a maximum of ten machines. Therefore, the job command file would look similar to:

```
#
# @ output = my_parallel_job.$(cluster).$(process).out
# @ error  = my_parallel_job.$(cluster).$(process).err
# @ min_processors = 8
# @ max_processors = 10
```

Figure 11. Sample Job Command File

- IBM AIX Parallel Operating Environment (POE) 1.2.1 – The next two examples show you, in addition to the statements shown in Figure 11, what your job command file might look like when using POE 1.2.1:
  - POE 1.2.1 Example 1:

    ```
    # @ executable   = /usr/bin/poe
    # @ arguments    = /home/userid/my_parallel_job -euilib us
    # @ job_type     = parallel
    # @ requirements = (Adapter == "hps_user")
    # @ class        = POE
    # @ queue
    ```

  - POE 1.2.1 Example 2:

    ```
    # @ job_type     = parallel
    # @ requirements = (Adapter == "hps_ip" && Pool == 4)
    # @ class        = POE
    # @ queue
    /usr/bin/poe /home/userid/my_parallel_job -infolevel 2
    ```

  Note the following requirements for POE jobs:

  - You must specify the POE partition manager as the executable (or invoke it from the job command file).
  - You must pass the parallel job as an argument using the full path name.
  - If you are running in user space mode, you must use the POE flag **-euilid us** (the default is **ip**).

– The job must have a *job_type* of **parallel**.

- Parallel Virtual Machine Extended (PVMe) 1.3.1 – The next example shows you, in addition to the statements shown in Figure 11 on page 84, what your job command file might look like when using PVMe 1.3.1:

```
# @ job_type     = parallel
# @ requirements = (Adapter == "hps_user")
# @ class        = PVMe
# @ queue
/usr/lpp/pvm3/pvmd3e <daemon options> -exec my_parallel_job <program args>
```

  Note the following requirements for PVMe jobs:

  – The PVMe daemon must be invoked from the job command file.

  – You must pass the parallel job as an **-exec** argument using the full path name.

  – You must specify a user space of **Adapter**.

  – The job must have a *job_type* of **parallel**.

- Parallel Virtual Machine (PVM) 3.3 – The next example shows you, in addition to the statements shown in Figure 11 on page 84, what your job command file might look like when using When using the PVM 3.3:

```
# @ executable    = my_parallel_job
# @ job_type      = pvm3
# @ parallel_path = /home/userid/cmds/pvm/$PVM_ARCH
# @ class         = PVM3
# @ requirements  = (Pool == 4)
# @ queue
```

  Note the following requirements for PVM 3.3 jobs:

  – You must specify the parallel executable as the executable.

  – A machine with the specified adapter is selected, but the adapter is not used. (The adapter address is not passed on by the resource manager.)

  – The job must have a *job_type* of **pvm3**.

Task 3: Obtain Status of Parallel Jobs

Both end users and LoadLeveler administrators can obtain status of parallel jobs in the same way as they obtain status of serial jobs – either by using the **llq** command or by viewing the Jobs window on the graphical user interface (GUI). By issuing **llq -l**, of by using the **Job Details** selection in the GUI, users get a list of machines allocated to the parallel job. The following is sample output:

```
=============== Job Step k1bn09.hpssl.kgn.ibm.com.1.0 ===============
        Job Step Id: k1bn09.hpssl.kgn.ibm.com 1.0
           Job Name: k1bn09.hpssl.kgn.ibm.com.1
          Step Name: 0
  Structure Version: 8
              Owner: lluser
         Queue Date: Mon Mar  6 11:11:22 1995
             Status: Running
    Completion Date: (not completed)
    Completion Code: (not completed)
      User Priority: 50
       user_sysprio: 0
      class_sysprio: 0
      group_sysprio: 0
    System Priority: -2481
          q_sysprio: -2481
      Notifications: Always
        User Number: 1
 Virtual Image Size: 2 kilobytes
         Checkpoint: no
     Hold Job Until: (no hold)
                Cmd: test_pvme.cmd
               Args:
                 In: /dev/null
                Out: my_parallel_job.1.0.out
                Err: my_parallel_job.1.0.err
Initial Working Directory: /u/lluser/test_parallel
         Dependency:
       Requirements: (Arch == "R6000") && (Adapter == "hps_user") <<<<<<<<<<<
        Preferences:
           Job Type: General Parallel                              <<<<<<<<<<<
     Min Processors: 2                                             <<<<<<<<<<<
     Max Processors: 2                                             <<<<<<<<<<<
Allocated host count: 2                                            <<<<<<<<<<<
Allocated host names: k1bn09.hpssl.kgn.ibm.com                     <<<<<<<<<<<
                      k1bn12.hpssl.kgn.ibm.com
    Submitting host: k1bn09.hpssl.kgn.ibm.com
        Notify User: lluser@k1bn09.hpssl.kgn.ibm.com
              Shell: /bin/ksh
           LL_group: No_Group
              Class: PVME
            Account: NONE
         Unix_group: staff
```

As an administrator, you can create a class for parallel jobs. Then, when users check the status of their parallel jobs, they can identify their parallel jobs by the class field on the Jobs window.

Task 4: Obtain Status of Machines

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

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

Task 5: Cancel Parallel Jobs

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

Submit Problem Checklist

- For POE:
  - Specify the POE partition manager as the executable. Do *not* specify the parallel job as the executable.
  - Pass the parallel job as an argument to POE.
  - The parallel job must exist and must be specified as a full path name.
  - If the job runs in user space, specify the flag **-euilib us**.
  - Specify the correct adapter (when needed).

- Specify a POE job only once in the job command file.
- Compile only with the supported level of POE.
- Specify only **parallel** as the *job_type*.

- For PVMe:
  - Do *not* invoke the PVMe daemon from the **executable** keyword.
  - Pass the parallel job to PVMe as an **-exec** argument.
  - The parallel job must exist and must be specified as a full path name.
  - If the job runs in user space, the user must specify a user space adapter.
  - Do *not* specify adapter **hps_ip**.
  - Specify a PVMe job only once in the job command file.
  - Compile only with the supported level of PVMe.
  - Specify only **parallel** as the *job_type*.
- For PVM:
  - Specify the parallel job as the executable. Do *not* specify PVM as the executable.
  - If the job runs in user space, do not specify a resource manager adapter.
  - Compile only with the supported level of PVM.
  - Specify only **pvm3** as the *job_type*.

Chapter 4. Interactive Login Sessions

Interactive Session Support (ISS) is used to distribute logins and application sessions across a pool of servers in a manner which is completely transparent to users and applications. Without ISS, users wishing to gain access to a pool of shared computing resources from TCP/IP applications must request a connection to a specific server within the pool by supplying a machine name or internet address which identifies the particular server. Traditionally there has been no facility for leveling of the load on the shared pool, and the distribution of logged in users across the shared pool is arbitrary.

When ISS is used, ISS interfaces with a TCP/IP nameserver to translate machine names to internet addresses. In addition, ISS assists the nameserver by monitoring the load on each of the servers in the pool, periodically modifying its recommendation to the nameserver as the load changes. Instead of specifying the machine name of a particular server, the user specifies the name of a pool which has been set up by the administrator. The advantage is that ISS can ensure that the user is connected to the server which has the lightest load of all the available servers in the specified pool. This efficiently balances the user load across all the available servers. Instructions on how to set up pools appear in "Using Interactive Session Support."

For some applications which require access to stored data, it is necessary to ensure that all servers within a pool have equal access to the necessary data. This is ensured by using a system such as NFS or by using a shared disk architecture.

You can configure ISS to suit your environment in terms of frequency of access, total number of users and types of access (e.g. short queries, long-running queries or CPU-intensive loads). Typically, ISS provides the internet address of the least-loaded server in a pool to the requesting user or application. ISS monitors a pool's activity metrics (as configured) and distributes user and application connection requests evenly across the pool's servers.

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

Using Interactive Session Support

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

© Copyright IBM Corp. 1993, 1995

Step 1: Install ISS

You need to install ISS on the machine which is running the TCP/IP domain nameserver which has authority over the subdomain over which interactive login sessions are to be leveled. For more information about TCP/IP domains and how to configure the nameserver, see “Configure the TCP/IP nameserver” on page 97.

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

If you plan to use LoadLeveler pools within ISS, then you need to create the userid **loadl** and group **loadl**.

Note: For more information on installing ISS, refer to the *Memo to Users*, which is included on the product tape.

Step 2: Determine How Interactive Sessions Will Be Achieved

In order to determine how end users will obtain interactive sessions, you need to determine which of the following three methods of pool allocation you will make available to users. Keep in mind that you can make all three methods available or only a subset. Refer to the following table to determine which method best suits your needs:

Select this method:	If these circumstances apply:	See:
LoadLeveler Configuration	There is an existing LoadLeveler configuration in place and you want to define the pool by using the machine descriptions available in the LoadLeveler administration file. This file contains stanzas that describe the characteristic of each machine and the type of workload that it supports.	“Using LoadLeveler configuration” on page 91.
ISS Custom Configuration	You want to use ISS independently of LoadLeveler and you want to define the pool by supplying a list of machine names and/or internet addresses of the machines in the pool by modifying the ISS configuration file.	See “Using ISS custom configuration” on page 91.
ISS Round-Robin Configuration	You want to use ISS independently of LoadLeveler and you want to define the pool by supplying a list of machine names and/or internet addresses of the machines in the pool by modifying the ISS configuration file.	“Using ISS round-robin configuration” on page 91.

These methods are not mutually exclusive so you can use them in combination. All of the methods require the servers to be divided up into *pools*. A pool is a group of servers which are equally suitable for a particular purpose. A pool allows the load created by TCP/IP connected applications to be spread across a number of servers. This provides better performance than would be achieved by using a single server.

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

Using LoadLeveler configuration

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

When you use ISS in this way, LoadLeveler must be running because ISS uses LoadLeveler to determine the load on each of the servers in the pool. ISS uses the metrics configured in LoadLeveler to determine the loading and availability of servers in the pool. You should configure LoadLeveler by specifying the machine descriptions and then configure ISS, using the same classes, requirements and preferences used in the LoadLeveler files. See “Step 3: Configure ISS” on page 92 for more details.

Using ISS custom configuration

You can use ISS independently of LoadLeveler. If you do this you can use any metric that you choose, and you define the metric in the ISS configuration file. This metric can be anything that can be executed on each of the servers and which will return a numeric result. Other parameters in the ISS configuration file determine how this result is interpreted. For example, where the metric is such that higher numbers represent higher loading, you would set POLICY to MIN, which indicates to ISS that it should minimize the metric. Alternatively, where higher numbers represent lower loading you would set POLICY to MAX, so that ISS maximizes the metric. See the description of the POLICY keyword in “Step 3: Configure ISS” on page 92 for more details. The following example illustrates a custom metric which counts the number of processes running on a system:

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

When using ISS in this way, it is independent of LoadLeveler and LoadLeveler does not have to be running. You configure ISS by defining pools as described in “Step 3: Configure ISS” on page 92. These pools are not related to any other LoadLeveler configuration information.

Using ISS round-robin configuration

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

When using ISS in this way, it is independent of LoadLeveler and LoadLeveler does not have to be running. You configure ISS by defining pools as described in “Step 3: Configure ISS” on page 92. These pools are not related to any other LoadLeveler configuration information.

There are advantages and disadvantages in using the round-robin method and its use is more appropriate in certain environments than in others. The main advantage is that ISS does not have to poll the servers, thus avoiding the overhead on the servers which would otherwise be introduced by the polling. The main disad-

vantage of round-robin scheduling is that ISS does not notice if a particular server is getting very busy, and will continue to schedule it periodically as the server to which new users of the pool connect. Providing the client load has a reasonably uniform profile in terms of frequency of access and duration of access, round-robin scheduling works very well.

Step 3: Configure ISS

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

Configure the ISS Configuration File

ISS has its own configuration file which defines the pools of servers and the metrics to use for each pool. A single ISS configuration file can contain definitions for LoadLeveler pools, ISS custom pools, and ISS round-robin pools. The name of the configuration file is supplied as a command line argument when ISS is run and you can name the file anything you want. An example configuration file is located in **/usr/lpp/LoadL/iss/Iss_config**. Copy the example file and edit it to specify your pools and their characteristics. You may have a variety of configuration files for different workloads and other requirements.

Syntax of the ISS Configuration File: The ISS configuration file is an ASCII file consisting of a series of keywords, each followed by one or more parameters. Each keyword must appear at the start of a line, and is followed by the parameters which relate to it. Blank lines are ignored and lines that begin with semicolons (;) and hash-symbols (#) are treated as comments. Keywords are case insensitive. The case of parameters is preserved, since they may contain custom metrics or hostnames.

The majority of keywords relate to a pool definition. There are two keywords, however, that do not relate to any particular pool. They are **ISS_TRACE** and **NAMED_BOOTFILE**, which can appear at the start of any line in the configuration file.

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

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

Table 4. Parameter significance for different metrics			
Keyword	**Metric**		
	LOADLEVELER	**CUSTOM**	**ROUNDROBIN**
CLASS	**mandatory**	ignored	ignored
NAME	**mandatory**	**mandatory**	**mandatory**
POLICY	ignored	**mandatory**	ignored
POLL	**mandatory**	**mandatory**	**mandatory**
PREFERENCES	optional	ignored	ignored
REQUIREMENTS	optional	ignored	ignored
SERVERS	ignored	**mandatory**	**mandatory**

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

Now would be a good time to refer to the sample configuration file supplied with ISS.

ISS Configuration File Keywords

This section lists the ISS configuration file keywords.

CLASS

Use this keyword when setting the METRIC keyword to LOADLEVELER. The value assigned to it must be a valid LoadLeveler class. This keyword allows you to restrict login sessions to those machines that support this class. The syntax is:

```
CLASS <LoadLeveler_Class>
```

For example, to set this keyword to the class **Fast_Response**, enter the following line:

```
CLASS Fast_Response
```

If a pool's METRIC keyword is not set to LOADLEVELER, this keyword is ignored.

NAME

The NAME keyword defines the name users will specify to connect to the pool of machines. A NAME entry in the configuration file marks the start of the configuration parameters for a new pool. Therefore, the NAME line must be the first non-comment line in the definition of a pool. The syntax is:

```
NAME <pool_name>
```

The parameter given to NAME is the smallest part of the fully specified name of the pool. For example, the pool name might be:

```
dept_pool.sp.site.corp.com
```

In this example, **site.corp.com** is the TCP/IP nameserver domain in which the client machines and the nameserver machine are situated, and **sp.site.corp.com** is the sub-domain over which the nameserver has authority. The NAME entry in the configuration file would be:

```
NAME dept_pool
```

The NAME keyword is required and must appear first when defining a pool. It can be any string that you choose but it must conform to the requirements for valid TCP/IP machine names, such as not containing whitespace or certain punctuation characters.

NAMED_BOOTFILE

The NAMED_BOOTFILE defines the path and filename of the bootfile used when the name daemon was started. The file generally used is **/etc/named.boot**, and where this is so, there is no need to specify the NAMED_BOOTFILE keyword. Where a different bootfile has been used to start the name daemon, this keyword must be provided in order for ISS to locate the nameserver data that it requires. The syntax is:

```
NAMED_BOOTFILE <filename>
```

POLICY

This keyword indicates whether the metric function specified in a METRIC CUSTOM entry is to be minimized or maximized. The syntax is:

```
POLICY { MIN | MAX }
```

For example, a metric based on the number of active users would regard the smallest value as the best and, therefore, this type of function would have the following entry:

```
POLICY MIN
```

A metric function based on CPU idle time, on the other hand, would regard the maximum value as best and, therefore, the policy would be specified as the following:

```
POLICY MAX
```

If METRIC is set to LOADLEVELER or ROUNDROBIN, this keyword is ignored. When METRIC is set to CUSTOM, this keyword is required.

POLL

The POLL keyword indicates the time in seconds between measurements of the load on each of the machines in the pool, and between updates of the name daemon by ISS. You can use this keyword to optimize the overall responsiveness of the pool in specific user workload scenarios. The syntax is:

```
POLL <poll_period>
```

For example, to initiate a poll of the pool every 30 seconds, enter the following:

```
POLL 30
```

This keyword is always required, regardless of the type of metric used.

PREFERENCES

This keyword is only used in conjunction with LoadLeveler. The value assigned to it must be a string containing valid LoadLeveler preferences, and allows you to bias login sessions to those machines that have the preferred characteristics in the LoadLeveler configuration. Refer to Appendix B, “LoadLeveler Job Command File Keywords” on page 183 for information on how to create a Boolean expression containing preferred characteristics.

If you do not set METRIC to LOADLEVELER, this keyword is ignored. The syntax is:

```
PREFERENCES <Boolean expression>
```

REQUIREMENTS

This keyword is only used in conjunction with LoadLeveler. The value assigned to it must be a string containing valid LoadLeveler requirements. It allows you to restrict login sessions to those machines that have the required characteristics in the LoadLeveler configuration. Refer to Appendix B, "LoadLeveler Job Command File Keywords" on page 183 for information on how to create a Boolean expression containing required characteristics.

If you do not set METRIC to LOADLEVELER, this keyword is ignored. The syntax is:

```
REQUIREMENTS <Boolean expression>
```

SERVERS

The SERVERS keyword marks the start of a list of machine names or internet addresses (in dotted decimal notation), specifying the machines which you want to include in the pool and which are to be accessed via the pool name given in the preceding NAME entry. Where possible, you should use machine names because they make administration simpler but you can use machine names and internet addresses interchangeably. Separate the list with blanks and continue the list on subsequent lines if required. If you use multiple lines, only issue the keyword on the first line.

The syntax is:

```
SERVERS <server1> [<server2>...]
```

For example, for 3 machines in an example pool, the SERVERS entry might look like this:

```
SERVERS 1.2.3.10 deptserver 1.2.3.12
```

There is no requirement for machines to be within the same internet subnet groups. It is only necessary that the machines are accessible from the client and the nameserver machines. You can define overlapping pools, such that the same machine name or internet address may appear on the SERVERS line of more than one pool defined in the configuration file.

This keyword is requried for CUSTOM and ROUNDROBIN. It is ignored when used with the LOADLEVELER metric.

ISS_TRACE

Place this keyword at the start of any line in the configuration file to cause ISS to produce trace output. This output is sent to **stdout** and can be useful for testing a new configuration and helping to resolve any problems.

Used in connection with the ability to reconfigure ISS while it is still running, you can use this keyword dynamically to turn trace on and off. See "Step 4: Reconfigure ISS While it is Running" on page 101.

The syntax is:

```
ISS_TRACE
```

This keyword is optional.

METRIC

This keyword controls which of the three methods of operation ISS uses to level login sessions across the pool to which this keyword relates (see "Using Interactive Session Support" on page 89 for details).

The syntax is:

```
METRIC { LOADLEVELER | CUSTOM <string> | ROUNDROBIN }
```

Valid values are:

LOADLEVELER
This value specifies that ISS is to treat this pool as a LoadLeveler pool. This means that the pool consists of those machines defined in LoadLeveler and the metric used will be the metric being used by LoadLeveler.

To configure a pool to use a LoadLeveler metric, add the following line in the ISS configuration file:

```
METRIC LOADLEVELER
```

When the METRIC keyword is set to LOADLEVELER, the keywords POLICY and SERVERS are ignored for this pool. The keyword POLL is mandatory. In addition, the keywords CLASS, REQUIREMENTS and PREFERENCES are optional.

CUSTOM <**string**>
This value specifies that ISS is to use a custom metric for this pool. The metric used is defined by the <string> supplied. ISS sends the metric string to each machine in the pool using a remote shell (**rsh**) and collects the returned value. It then uses this value to determine which machine would be the best one for new users to be connected to, based on the load-leveling policy specified by the POLICY keyword. This polling is repeated at intervals specified in the ISS configuration file by the POLL keyword.

The string is one or more UNIX commands, probably separated by the '|' pipe-through operator. The end result of these commands must be an integer, between 0 and 99,999 inclusive, sent to standard output. The number must be a non-negative integer.

It is useful to test new metric strings by typing them from the command line before putting them in the configuration file. For example:

```
rsh machine metric
```

Be careful when using quotation marks to ensure that the desired portion of the string is executed on the remote machine. The following examples show some useful metrics that you can use.

- the number of users logged-on,
- the number of processes running,
- the number of rsh daemons (rshd) in action,
- the percentage CPU utilization of the machine either instantaneously or over an averaging period,
- the percentage idle time of the machine
- the number of application instances of a certain sort which are in operation,
- the number of users connected via telnet (telnetd).

The metric string must fit on one line in the configuration file after the METRIC and CUSTOM keywords. If the load-leveling algorithm is highly complex, you should encode the algorithm in a shell script or perhaps an application which is stored on each of the remote machines. The custom metric will then be an invocation of that script or program which returns the computed metric value to standard output as an integer between 0 and 99,999 inclusive.

Some examples of custom metrics which have proven highly appropriate in a number of applications of ISS are described below. Note that all of these require POLICY MIN unless otherwise stated.

```
METRIC CUSTOM 'ps -ef | grep rshd | wc -l'
```

This metric returns the number of rsh requests currently active on the remote machine. It is desirable to minimize this value. The quotation marks ensure that the whole line is sent to the remote machine, rather than just the first command in the pipeline, as in the following example,

```
METRIC CUSTOM ps -ef | wc -l
```

where the **wc -l** actually runs on the nameserver machine. This is not necessarily a problem but there could be performance implications if a large volume of information has to be sent across the network.

```
METRIC CUSTOM uptime | awk '{print $NF}'
```

This metric returns the number of runnable processes in the scheduler queue on the remote machine. This value should be minimized. Be careful when using quotation marks to ensure that there are single quotation marks to bind the commands together and there are single quotation marks around the awk 'program'. The solution in this instance is:

```
METRIC CUSTOM ‘uptime | awk ‘“‘“‘{print $NF}‘“‘“‘
```

The final example uses **vmstat** to give the system idle time during a one second sample period. This value should be set as high as possible so POLICY should be set to MAX.

```
METRIC CUSTOM ‘vmstat 1 2 | tail -1’ | awk ‘{print $16}’
```

Note that when METRIC is set to CUSTOM <string>, the keywords CLASS, PREFERENCES and REQUIREMENTS are ignored for this pool and the keywords SERVERS, POLL and POLICY are required.

ROUNDROBIN

This value specifies that ISS is to use round-robin scheduling for this pool. This means that users requests are automatically rotated around the machines in the pool irrespective of their loading, at time intervals specified by the value of the keyword POLL. The round-robin metric does not poll the machines in the pool. Instead, it cycles around the machines in the pool in the order in which they appear in the SERVERS list.

To configure a pool for round-robin scheduling, add the following metric line in the configuration file:

```
METRIC ROUNDROBIN
```

Note that when METRIC is set to ROUNDROBIN, the keywords CLASS, PREFERENCES and REQUIREMENTS are ignored for this pool and the keywords SERVERS, and POLL are required.

Configure the TCP/IP nameserver

This section describes how to set up the TCP/IP nameserver part of the ISS system.

Select an appropriate configuration: The first task you need to accomplish is to designate a machine to act as the nameserver for the domain which contains the servers in the ISS pools. The choice of machine and the definition of the domain depend on whether there is an existing nameserver in use and the size of the

domain which it serves. Refer to the following table to help you decide which machine in your network to use to act as the nameserver:

If these circumstances apply:	Then:
Your organization already has a nameserver in use	You can add the pool servers to a domain served by this existing nameserver or you can add them as an additional sub-domain which that machine will serve.
Your organization either does not have an existing nameserver in use or wants to use a different machine to act as the nameserver. You might want to use this alternative machine in order to isolate the ISS nameserver from the site nameserver.	You can add the pool servers as a new sub-domain to this new nameserver and modify the client machines so that they go in the first instance to the new nameserver. You could also add the pool servers as a new sub-domain to this new nameserver and have an existing nameserver pass on requests for the new sub-domain. This approach is best suited for large sites because if your site has a large number of users, it may not be appropriate to add the ISS pools to the site nameserver because ISS causes the nameserver to reload its database once every poll period. For large named data files the overhead of doing this could be prohibitive. In such sites the pool servers should be added to a separate sub-domain, and this sub-domain should be administered by a separate nameserver machine. This does not necessarily mean that you have to modify the many client workstations' **/etc/resolv.conf** files, since the site nameserver can be configured to pass on anything for the newly introduced ISS sub-domain. This allows the client workstations to remain unchanged, directing all requests for name resolution to the site nameserver.

The following instructions, see “Configure a new nameserver,” describe how to set up a sub-domain for the pool servers and how to introduce a machine to act as a nameserver for that sub-domain. In addition, the instructions illustrate how to modify the client workstations so that they use the new nameserver and how to set up the new nameserver to pass requests for names outside the new sub-domain to an existing nameserver.

If you do not want to use this approach to setting up a pool server, you can still follow these instructions but you can ignore the section on modifying the client machines.

If you are using an existing nameserver, you do not have to set up a sub-domain and you can add the server names to the existing nameserver's data file.

If you will be adding the pool servers as a new sub-domain to a new nameserver and you want to use an existing nameserver to pass on requests for the new sub-domain, you also need to add an entry to the existing nameserver so that it passes on requests for the new sub-domain to the new nameserver.

Configure a new nameserver: The TCP/IP nameserver is set up to have authority over a sub-domain of the main domain to which the server machines, the nameserver machine, and the client machines belong. For example, you might use the sub-domain **sp.site.corp.com** to denote the domain of authority of the ISS system, with all client machines and the nameserver machine and the pool machines all being configured as part of the main domain **site.corp.com**. Note that the pool machines are not designated as being part of the **sp.site.corp.com** sub-domain. It is not necessary to change the server machines in order for ISS to provide load-leveling over those machines.

There are three files which determine the behavior of the TCP/IP nameserver. These files are described in the following list:

- **boot file**

  This file is normally called the **named.boot** file. You can supply the name of this file as a command line parameter when the **named** daemon is started. If you don't supply the name, then the **named** daemon uses the /**etc/named.boot** file. Use the root id to copy the ISS sample file provided to you and modify it. Take care not to destroy any existing **named.boot** file. There are three essential lines in this file which you have to configure according to the particular environment in which you will use the ISS system:

  ```
  domain  sp.site.corp.com
  primary sp.site.corp.com       /u/loadl/named.data
  primary in-addr.arpa            /u/loadl/named.rev
  ```

  The `sp.site.corp.com` string is the sub-domain that we are creating for the server machines. You should modify the pathnames to the files **named.data** and **named.rev** to match the path that has been chosen for installing the ISS files. For this example, the LoadLeveler home directory (/u/loadl) has been used. The sub-domain name string should be set correctly, with an appropriate sub-domain name, and the main domain to which the client and server machines belong.

- **reverse data file**

  This file is normally called the **named.rev** file and it is identified in the boot file. You should copy the reverse data file to the chosen directory (for example, /**u**/**loadl**). Its full pathname must appear in the boot file.

  The main purpose of this file is to ensure that reverse name resolution requests (dotted decimal to machine name conversion) which are outside the domain of the ISS nameserver are passed on to the main site nameserver for resolving. For example, you might want anything which begins with "1.2.", but which has not been resolved by the ISS nameserver, to be resolved by the local site nameserver; so you would have the following line:

  ```
  2.1     9999999 IN      NS      nameserver.site.corp.com.
  ```

  In the Start of Authority (SOA) stanza of this file, the ISS nameserver machine name is specified, including the sub-domain name. Note that the ISS nameserver machine is not itself configured (in its TCP/IP configuration) as part of the sub-domain, we are just telling the **named** daemon that it is for the benefit of ISS. You should change this line as appropriate to match the local environment.

- **data file**

  This file is normally called the **named.data** file. You should copy this file to the chosen directory (for example, /**u**/**loadl**). Its full pathname must appear in the boot file.

  The purpose of this file is to tell the **named** daemon what dotted-decimal internet address to return in response to a request to convert a machine name into an internet address ("forward" resolution). This is the file in which you declare the names for the pools over which ISS is presiding. ISS searches only the data file for the names of ISS pools. ISS does not search any other files specified by $INCLUDE statements in the data file.

The data file also contains the information to permit hierarchical name resolution. Any name which the ISS name daemon cannot resolved is passed to the site nameserver.

You need to modify several lines in this file: In this example, the name of the machine on which the nameserver and ISS are running is `poolserver`:

```
@         9999999 IN      SOA     poolserver.sp.site.corp.com.
                                  root.poolserver.sp.site.corp.com. (
```

You should adjust the entries in the SOA stanza to match the local environment, and the name of the ISS nameserver machine and the sub-domain name.

```
                9999999 IN      NS      poolserver
```

This Name Server (NS) stanza has the machine name of the ISS nameserver machine. Update this stanza if necessary to reflect the correct name.

The next two stanzas indicate the way that unresolved name resolution requests are passed up to the nameserver with authority over the domain of which the ISS system is a sub-domain.

```
site.corp.com.              9999999 IN   NS   nameserver.site.corp.com.
nameserver.site.corp.com.   9999999 IN   A    1.2.3.4
```

The following lines declare the names of the pools associated with the groups of servers over which ISS has control. The names are given without their sub-domain name and the internet address given should initially be set to the address of one of the machines in the pool.

```
tpserver     9999999    IN  A    1.2.3.4
dbclust      9999999    IN  A    1.2.3.4
```

Once you edit all of these files, you can start the name daemon on the ISS nameserver machine with the following command (issued by the root user):

```
/etc/named
```

or

```
startsrc -s named
```

Configure the Client Machines

If you are not setting up a sub-domain for the pool servers and you don't want to introduce a new machine to act as a nameserver for that sub-domain and if you do not need to modify client machines to interact with the new nameserver, you can ignore this section. If your selected approach dictates that you need to configure the client machines, follow the instructions presented in this section. These instructions illustrate how to set up the TCP/IP configuration on the the client machines so that the ISS nameserver is the main nameserver and the site nameserver is the backup nameserver. You can do this using SMIT for AIX platforms. For other platforms, use any appropriate configuration tool. Using SMIT, select these choices:

- Communications Applications and Services
- TCP/IP
- Further Configuration
- Name Resolution
- Domain Nameserver (/etc/resolv.conf)
- Add a Nameserver.

When you are finished, ensure that the /**etc/resolv.conf** has the nameserver entries in the following order:

1. The ISS nameserver machine should be first.
2. The site nameserver machine should be second.

The ISS nameserver is used for name resolution under normal conditions. If the ISS nameserver does not respond, then the backup nameserver is used to perform name resolution. Note that when the backup nameserver is being used, the performance may be affected.

To test the operation of the ISS nameserver, make a name resolution request to the nameserver from a client machine. For example, from the name **myserver**, type:

```
host myserver
```

This returns the dotted decimal internet address for the name that is specified in the **named.data** file.

Step 4: Reconfigure ISS While it is Running

To reconfigure ISS while it is still running, edit the configuration file that was used when ISS was started and send a SIGHUP signal to the ISS process, as follows:

```
kill -HUP <iss_pid>
```

where <iss_pid> is the process id of the **iss** process, which can be determined by issuing the following command:

```
ps -ef | grep iss
```

ISS then re-reads the configuration file and alters its behavior in accordance with the new pool definitions provided. This causes ISS to temporarily suspend leveling of TCP/IP connections but does not prevent new connections to pools from being established. All such requests received during reconfiguration are resolved to the machine address which was current for each pool when the above command was issued. Reconfiguration is very quick and should not be noticeable to users.

Pool names which are deleted from the configuration file during reconfiguration (with ISS either running or stopped) will still be present in the nameserver datafile but will not be actively leveled. They will always resolve to the machine address which was current for that pool when the configuration was changed.

Step 5: Start ISS

You can run ISS in the foreground or background and you can run it using `nohup` if desired. You can start ISS after ensuring that the TCP/IP name daemon is running and is correctly resolving names from the client machines. To start ISS, as root, enter the following:

```
iss <filename>
```

where <filename> is the name of the ISS configuration file. ISS prints an opening message. If it does not find any errors in the ISS configuration file, it begins to transparently level TCP/IP connections across the pools defined in the configuration file. If there are problems with the pool definitions in the configuration file, a message is generated and ISS exits. If this occurs, modify the configuration file and try again. Refer to the sample configuration file for guidance.

Should ISS close down without cleaning-up, it will detect this when it is restarted and will automatically clean-up any ISS resources that were not used.

Step 6: Establish an Interactive Session

An interactive session can be established with any standard AIX TCP/IP application such as **telnet, rsh, ftp, or rlogin**. When the application expects you to provide the machine name or internet address of the remote host to which you want to connect, enter instead the name of the pool of machines to which you want to connect. For example:

```
telnet dept_pool
```

where *dept_pool* is the name of the ISS defined pool. By using the *pool name* the user is automatically connected to a least-loaded node in that pool, using the leveling metric defined in the configuration file.

Applications that specify TCP/IP machine names or internet addresses for client server connections should also specify the *pool name* in their TCP/IP configuration files.

Depending on the way your site nameserver is set up and the relationship between the TCP/IP domains in which the client and server machines are located, you may need to specify the pool name more fully. For example, *dept_pool.sp* or even *dept_pool.sp.site.corp.com* This may be necessary if your client is served by AFS.

Step 7: End an Interactive Session

Interactive sessions are ended in the same manner as they would be in the absence of ISS. For example, to end an ftp session, type **quit** at the ftp prompt or to end a telnet session type the escape character followed by **quit**.

Step 8: Stop ISS

To stop ISS, issue a SIGINT signal. For example:

```
kill -INT <iss_pid>
```

where <iss_pid> is the process id of the **iss** process, which can be determined by issuing the following command:

```
ps -ef | grep iss
```

This gives ISS time to clean up its shared system resources and close down gracefully. Should ISS close down without cleaning-up, it will detect this when it is restarted, and will automatically clean-up any ISS resources that were not used.

When ISS is not running, access requests directed to the pool names will be resolved to the machine address which was current for each pool when ISS was terminated.

Troubleshooting

Using the ISS trace facility

If ISS does not appear to be working as you would expect, then try running it with its trace facility enabled. You can enable ISS trace by inserting the keyword ISS_TRACE into the ISS configuration file. The position of the keyword in the file is not important, except that it should be on a line by itself and should be at the start of that line. Tracing will start as soon as ISS encounters the keyword in the file, so if you wish to see what happens when ISS parses your configuration file, put it at the top. If you just want tracing of events following the parsing of the configuration file then you could put it at the bottom.

Setting this keyword in the configuration file causes ISS to write trace messages to stdout. Examine the trace that is produced and compare it with the problems described in this section.

An example of the use of the ISS_TRACE keyword in the configuration file follows:

```
ISS_TRACE
NAME    My_Pool
METRIC  LOADLEVELER
CLASS   No_Class
 ..etc..
```

Note that ISS can be reconfigured while running using SIGHUP. Therefore, you can add the ISS_TRACE keyword to the configuration file and issue:

```
kill -1 <iss pid>
```

in order to turn ISS trace on. You can turn it off by removing ISS_TRACE from the configuration file and issuing a further SIGHUP.

Problem: name <pool_name> NOT FOUND

You may encounter this message when attempting to connect to a machine identified by an ISS pool name. Try and specify the pool name more fully. For example:

```
dept_pool.sp
```

or

```
dept_pool.sp.site.corp.com
```

When resolving addresses, all client machines should refer to the ISS nameserver before any other nameservers in **/etc/resolv.conf**. For AIX systems, adding a nameserver using `smit` will add that nameserver to the end of the **/etc/resolv.conf** file. Therefore, either delete existing entries and then add all desired nameservers in the appropriate order or edit **/etc/resolv.conf** manually.

Problem: processes are left running after ISS is stopped

When ISS is configured with a CUSTOM METRIC, it generates a number of child processes, one for each server node controlled by a CUSTOM METRIC. These processes disappear when ISS exits but in rare circumstances, orphans could remain. You can recognize these processes by their process names, which are in the form `iss_server.<pool_name>.<machine#>`, For example,

```
iss_server.dept_pool.2
```

ISS removes these orphans when it is started up again. You can kill them if you do not plan to restart ISS in the near future. Another way to kill them is to start ISS and stop it almost immediately to let it remove the processes.

Problem: ISS does not start up

ISS uses a temporary file called /**tmp**/**iss.keys** to hold keys for shared memory and system semaphores. If ISS has problems cleaning-up during a restart, it might mean that the /**tmp**/**iss.keys** file is no longer synchronized with the system resources. This is unlikely to happen, but if ISS seems to be having problems with clean-up on restart, take the following steps:

1. Enter `ipcs` to list shared memory segments and semaphores.
2. Examine the listing for shared memory segments or semaphores with keys starting from **f000**, and remove them using the `ipcrm` command. To remove a shared memory segment, enter:

   `ipcrm -m` *id_number*

   where *id_number* is the ID number of the memory segment that corresponds to the key starting from **f000**.

   To remove semaphores, enter:

   `ipcrm -s` *id_number*

   where *id_number* is the ID number of the semaphore that corresponds to the key starting from **f000**.
3. Delete /**tmp**/**iss.keys**, if it exists.

Problem: All interactive sessions are assigned to one machine

If this situation occurs, it is probably because:

- One machine genuinely has a consistently light load relative to other machines in the pool. This may be because other machines in the pool are supporting some other activity, which is not under the control of LoadLeveler or ISS.
- A high load on the pool is causing the machines to not respond to ISS. When a machine does not respond, ISS avoids that machine when making its next recommendation. If all machines are too busy to respond, then ISS will defer a recommendation until at least one machine responds. In the meantime, all traffic will be directed to the last machine which was recommended.
- There may be a problem with ISS configuration or file permissions, which is preventing ISS from operating satisfactorily.

If this condition does arise, check that there are no warnings being printed (to stderr) which indicate that one or more hosts are not responding. See “Problem: Warning: server <internet address> not responding” on page 105 for more details.

Check that the load on the machines is not skewed to the extent that the metric being used to level load across the pool would genuinely result in the same machine being nominated every time. You can do this by looking at the amount of idle time each CPU is experiencing (using vmstat or sar), or by looking at the list of active processes (using ps). If using a LoadLeveler pool, you can use **llstatus** or **xloadl** to test the availability of machines which support the Class, Requirements and Preferences defined for the pool.

Also, turn on ISS_TRACE to determine what ISS is doing. See “Using the ISS trace facility” on page 103 for details.

Problem: Warning: server <internet address> not responding

If you see this warning occasionally, don't worry. It is quite normal for it to occur occasionally, but it should not occur consistently. Frequent occurrences (i.e. approximately every poll period, as defined in the configuration file) indicate that a machine is either not responding because it is too busy or because ISS cannot contact it.

Check that ISS is being run by root.

Check that the root user on the machine on which ISS is running has rsh permissions on all the machines in the pool, and in particular on the machine to which the warning relates. Do this by checking the **.rhosts** file on each of the machines, and look for an entry which names the ISS machine and root. For example,

```
issmachine.site.corp.com root
```

If you are using AFS, you may need to ensure that the **rcommands** such as rsh and rlogin that you have are AFS compatible.

Problem: llinitiate: configuration error - no default group

This error message occurs if there is no stanza for the loadl user in the **LoadL_admin** file in the LOADL_HOME directory, or if there is a stanza but it has no default group assignment.

Ensure that such a user stanza exists and that the default group is set.

Problem: ISS trace shows "llinitiate()" and nothing further happens

This problem is caused by LoadLeveler not responding to ISS. This can happen if, for example, the **LoadL_admin** file has been changed. The LoadLeveler configuration needs to be refreshed by using the **llctl** command.

Problem: ISS trace shows "Cannot rename <file 1> to <file 2> Permission denied"

Ensure that ISS is being run by root, and check the permissions on the files named in the error message.

ISS Configuration File

The following is the ISS configuration file called **lss_config** which is found in the install directory **/usr/lpp/LoadL/iss/lss_config**:

```
#
# Iss_config
#
# Sample configuration file for Interactive Session Support
#

# Uncomment the next line to turn the ISS trace facility on
#ISS_TRACE

# Uncomment the next line to provide a non-default named bootfile
# inserting the fully pathed filename of your bootfile in place
# of <filename>
#NAMED_BOOTFILE <filename>

# Example Pool definitions follow

# Example LoadLeveler pool
#
NAME            ISS_LoadL_pool
# update every 20 seconds
POLL            20
METRIC          LOADLEVELER
CLASS           No_Class
REQUIREMENTS    (Arch == "R6000")
PREFERENCES

# Example ISS custom metric pool
#
NAME            ISS_Custom_pool
# update every 10 seconds
POLL            10
METRIC          CUSTOM 'ps -ef | wc -l'
SERVERS         <server1> <server2> <server3>
                <server4>
# minimise the metric
POLICY          MIN

# Example ISS round-robin metric pool
#
NAME            ISS_RR_pool
# update every 5 seconds
POLL            5
METRIC          ROUNDROBIN
SERVERS         <server1> <server2>
                <server3> <server4>
```

Chapter 5. Gathering Job Accounting Data

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

Collecting Job Resource Data on Serial and Parallel Jobs

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

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

You can also view resource consumption information on serial and parallel jobs that are still running by specifying the following keywords in the configuration file:

JOB_ACCT_Q_POLICY = *number*
where *number* is the amount of time in seconds that determines how frequently resource consumption data is collected and stored on disk. The default is 300 seconds or 5 minutes.

JOB_LIMIT_POLICY = *number*
where *number* is the amount of time in seconds that determines how frequently the process table is examined and job_cpu_limit is tested.

Collecting Job Resource Data Based on Machines

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

To enable recording of resources by machine, you need to specify the following in the LoadLeveler configuration file.

ACCT = A_ON A_DETAIL

The machine's speed is part of the data collected. With this information, an installation can develop a charge back program which can charge more or less for resources consumed by a job on different machines. For more information on a machine's speed, refer to the machine stanza information. See “Step 1: Specify Machine Stanzas” on page 20.

© Copyright IBM Corp. 1993, 1995

Collecting Job Resource Data Based on Events

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

llctl capture -g *eventname*
where *eventname* is any string of continuous characters (no whitespace is allowed) that defines the event about which you are collecting accounting data. For example, if you were collecting accounting data on the *graveyard* work shift, your command could look like:

llctl capture -g *graveyard*

Using this command allows you to obtain a snapshot of all data up to and including that moment in time. If you wanted to capture this type of information on a regular basis, you could set up a crontab entry to invoke this command regularly. For example:

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

For more information on the **llctl** command, refer to "llctl - LoadLeveler Control" on page 134. For more information on the collection of accounting records, see "llq - Query Job Status" on page 145.

Collecting Job Resource Information Based on User Accounts

If your installation is interested in keeping track of resources used on an account basis, you can require all users to specify an account number in their job command files. They can specify this account number with the **account_no** keyword which is explained in detail in Appendix B, "LoadLeveler Job Command File Keywords" on page 183.

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

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

- User name

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

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

Collecting the Accounting Information and Storing it into Files

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

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

LoadLeveler accounting uses the following types of files:

- The local history file which is local to each **schedd** machine is where job resource information is first recorded. These files are usually named *history* and are located in the spool directory of each **schedd** machine, but you may specify an alternate name with the **HISTORY** keyword in either the global or local configuration file. For more information, refer to the “Step 7: Define Job Accounting” on page 50.
- The global history file is a combination of the history files from some or all of the machines in the LoadLeveler cluster merged together. The command **llacctmrg** is used to collect files together into a global file. As the files are collected from each machine, the local history file for that machine is reset to contain no data. The file is named *globalhist.yymmddhhmm*. You may specify the directory in which to place the file when you invoke the **llacctmrg** command or you can specify the directory with the **GLOBAL_HISTORY** keyword in the configuration file. The default value set up in the sample configuration file is the local spool directory:

GLOBAL_HISTORY = *$(SPOOL) (optional)*

Accounting Reports

You can produce three types of reports using either the local or global history file. These reports are called the *short*, *long*, and *extended* versions. As their names imply, the short version of the report is a brief listing of the resources used by LoadLeveler jobs. The long version provides more comprehensive detail with summarized resource usage and the extended version of the report provides the comprehensive detail with detailed resource usage. If you do not specify a report type, you will receive the default short version.

The short report displays the number of jobs along with the total CPU usage according to user, class, group, and account number. The extended version of the report displays all of the data collected for every job. See the **llsummary** command, "llsummary - Returns job resource information" on page 157, for examples of the short and extended versions of the report.

For information the accounting Application Programming Interfaces, refer to Appendix E, "LoadLeveler's Application Programming Interfaces" on page 219.

Sample Job Accounting Scenario

The following sample scenario walks you through the process of collecting account data. You can perform all of the steps or just the ones that apply to your situation.

Task 1: Update the Configuration File

Edit the configuration file according to the following table:

Edit this keyword:	To:
GLOBAL_HISTORY	Specify a directory in which to place the global history files.
ACCT	Turn accounting and account validation on and off and specify detailed accounting.
ACCT_VALIDATION	Specify the account validation routine.
Note: See "Step 7: Define Job Accounting" on page 50 for more information on these keywords.	

Task 2: Merge Multiple Files Collected From Each Machine Into One File

You can accomplish this step using either the **llacctmrg** command or the graphical user interface:

- Using **llacctmrg**

  See "llacctmrg - Collects machine history files" on page 128 for the syntax of this command.

- Using the Graphical User Interface

  Select a machine from the Machines window

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

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

  Press **OK**

  ▲ The window closes and you return to the main window.

Task 3: Report Job Information on all the Jobs in the History File

You can accomplish this step using either the **llsummary** command or the graphical user interface:

- Using **llsummary**

  See "llsummary - Returns job resource information" on page 157 for the syntax of this command.

- Using the Graphical User Interface

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

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

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

  – A short, long, or extended version of the output. The short version is the default version.

  – Start and end date ranges for the report. If no date is specified, the default is to report all of the data in the report.

  – The name of the input data file.

  – The name of the output data file.

  Press **OK**

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

Task 4: Using Account Numbers and Setting Up Account Validation

1. Specify the following keyword in the user stanza in the administration file:

   account = *list*
   where *list* is a blank delimited list of account numbers a user may use when submitting jobs.

2. Instruct users to associate an account number with their job.

   - Using the job command file

     Add the **account_no** keyword to the job command file. See Appendix B, "LoadLeveler Job Command File Keywords" on page 183 for details.

   - Using the Graphical User Interface

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

     ▲ The Build a Job window appears.

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

     Press OK

     ▲ The window closes and you return to the main window.

3. Specify the **ACCT_VALIDATION** keyword in the configuration file that identifies the module that will be called to perform account validation. The default module is called **llacctval**. You can replace this module with your installation's own accounting routine by specifying a new module with this keyword.

Task 5: Specifying Machines and Their Weights

To specify weights to associate with the machines, specify the following keyword in the machine's **machine stanza** in the administration file:

speed = *number*
where *number* defines the weight. The higher numbers correspond with a greater weight. The default weight is 1.0.

Chapter 6. Error Logging Facility

LoadLeveler performs error recording by using the AIX error logging facility. **You must run LoadLeveler on AIX Release 4.1 in order to use this facility.**

In addition, LoadLeveler daemons and processes also keep log files. For more information on these LoadLeveler log files, see "Step 10: Record and Control Log Files" on page 53.

Error Logging Overview

Error logging is the writing of information to persistent storage to be used for debugging purposes. This type of logging is for subsystems that perform a service or function on behalf of an end user. The subsystem does not communicate directly with the end user and, therefore, needs to log events to some storage location. The events that are logged are primarily error events.

LoadLeveler error logging uses AIX error log facilities to report events on a per machine basis. The intent is to have the AIX error log be the starting point for diagnosing system problems. For more information on AIX error log facilities, see *IBM AIX Version 4.1 Problem Solving Guide and Reference*, SC23-2606, and *IBM General Concepts and Procedures for RISC System/6000*, GC23-2202. For more information on Error Log facilities available with the SP, see *IBM RS/6000 Scalable POWERparallel SP Diagnosis and Messages Guide*, SC23-3866.

Error log entries include a "`DETECTING MODULE`" string that identifies software component, module name, module level, and line of code or function that detected the event that was logged. The information is formatted depending on the logging facility the user is viewing. For example the AIX Error Log facility information appears:

```
DETECTING MODULE
LPP=LPP name Fn=filename SID_level_of_the_file L#=Line number
```

How Error Log Events Are Classified

The following table shows the mapping of LoadLeveler error log suffixes to syslog priorities and AIX Error Log error types.

Table 5. LoadLeveler Error Label Suffixes Mapped to AIX Error Log

Error Label Suffix	AIX Error Log Error Type	AIX Error Log Description
EM	PEND	The loss of availability of a device is imminent.
ER	PERM	No recovery from this condition. A permanent error occurred.
ST	UNKN	It is not possible to determine the severity of the error.
TR	UNKN	It is not possible to determine the severity of the error.
RE	TEMP	Condition was recovered after several unsuccessful attempts.
DE	UNKN	It is not possible to determine the severity of the error.

© Copyright IBM Corp. 1993, 1995

What Events are Recorded

The following events are recorded by the LoadLeveler error logging facility:

- Errors occurring during the initial start up of LoadLeveler daemons
- Installation and customization errors
- The restarts per hour limit being exceeded
- **LoadL_master** detecting the crash of a daemon
- Changes in the state of a daemon
- LoadLeveler truncating the log files

Viewing LoadLeveler Error Log Reports

Enter the following command to view the LoadLeveler error reports for a machine:

```
errpt -a -N LoadLeveler
```

Clearing All LoadLeveler Error Log Entries

Enter the following command to clear all LoadLeveler entries from the error logs of a machine:

```
errclear -N LoadLeveler 0
```

Error Notification

You can be notified of a LoadLeveler error when it occurs by using the AIX Error Notification Facility. This facility will perform an ODM method defined by the administrator when a particular error occurs or a particular process fails. *IBM General Concepts and Procedures for RISC System/6000* (GC23-2202) explains how to use the AIX Error Notification Facility.

Sample Error Log

The following is a sample LoadLeveler error report:

```
LABEL:          LL_TRUNCATE_ST
IDENTIFIER:     05D868E9

Date/Time:       Fri Mar 31 18:07:43
Sequence Number: 271
Machine Id:      000001801000
Node Id:         ll2
Class:           S
Type:            UNKN
Resource Name:   LoadLeveler

Description
SOFTWARE

Probable Causes
APPLICATION PROGRAM

User Causes
FILE NEEDS REORGANIZATION

        Recommended Actions
        NO ACTION NECESSARY

Detail Data
DETECTING MODULE
LPP=LoadL,Fn=dprintf_config.c,SID=1.2,L#=392,
DIAGNOSTIC EXPLANATION
Logfile "/u/loadl/log/StartLog" was truncated
```

Figure 12. Sample LoadLeveler Error Log

Possible Causes for LoadLeveler Errors

Table 6 describes the possible causes for some common problems reported by the error logging facility:

Table 6 (Page 1 of 6). Possible Causes for LoadLeveler Failure

Label: LL_INIT_ER
Error Type: PERM
Diagnostic Explanation: LoadL_master must be run as root

Explanation: The ownership and the permissions of **LoadL_master** file are incorrect

Cause: This error ocurrs for one of the following reasons:

- The **LoadL_master** file is not owned by root
- The **LoadL_master** file does not have Set User ID permission (the “s” bit)

Action: Perform the following actions:

- Issue **chown root LoadL_master**
- **chmod u+s LoadL_master**
- Restart LoadLeveler

Table 6 (Page 2 of 6). Possible Causes for LoadLeveler Failure

Label: LL_INIT_ER
Error Type: PERM
Diagnostic Explanation: master: could not set the real uid to ROOT, rc=nn

Explanation: Unable to set the real Uid to 0 (root).
Cause: The setuid system call failed for an authorized caller.
Action: Contact the system administrator

Label: LL_INIT_ER
Error Type: PERM
Diagnostic Explanation: Error setting group id to nn. errno = nn.

Explanation: **LoadL_master** was unable to set the group ID.
Cause: The setguid() call failed.
Action: Verify that the following are true:

- **LoadLGroupid** is defined correctly in **/etc/LoadL.cfg**
- The group ID reported in the error description is defined on the system.

Label: LL_START_ER
Error Type: PERM
Diagnostic Explanation: LoadLeveler cannot continue; this host not in machine list

Explanation: You attempted to start LoadLeveler on a machine that is not defined in the **LoadL_admin** file, and MACHINE_AUTHENTICATE = TRUE is set in the **LoadL_config** file.
Cause: See "Explanation."
Action: Do one of the following:

- Define the machine in the **LoadL_admin** file
- Set MACHINE_AUTHENTICATE = FALSE in the **LoadL_config** file

Then, restart LoadLeveler on the machine where the problem occurred.

Label: LL_START_ER
Error Type: PERM
Diagnostic Explanation: Cannot start LoadLeveler. Must be the designated LoadLeveler administrator

Explanation: A user other than the user defined as the LoadLeveler administrator attempted to start LoadLeveler.
Cause: The **LOADL_ADMIN** keyword does not contain the user that attempted to start LoadLeveler.
Action: Add the user to the **LoadL_admin** file.

Table 6 (Page 3 of 6). Possible Causes for LoadLeveler Failure

Label: LL_START_ER
Error Type: PERM
Diagnostic Explanation: START_DAEMONS flag was set to *value*. Exiting.

Explanation: **LoadL_master** could not start the LoadLeveler daemons.
Cause: The **START_DAEMONS** keyword was not defined or was set to False.
Action: Set START_DAEMONS = true.

Label: LL_START_ER
Error Type: PERM
Diagnostic Explanation: Cannot get address of *hostname* from nameserver; LoadLeveler not starting.

Explanation: LoadLeveler could not resolve the hostname of the machine defined as the central manager.
Cause: gethostbyname() failed.
Action: Follow standard procedures for resolving nameserver problems.

Label: LL_INSTL_CUST_ER
Error Type: PERM
Diagnostic Explanation: *keyword* not specified in config file.
Admin file not specified in config! Cannot continue.

Explanation: The specified keyword is not defined correctly.
Cause: This error occurs for one of the following reasons:
- The keyword value is equal to a blank
- The keyword is misspelled
- The keyword is commented out
- The keyword is missing

Action: Define the keyword correctly.

Label: LL_INSTL_CUST_ER
Error Type: PERM
Diagnostic Explanation: *keyword*=*value* can not execute

Explanation: LoadLeveler was unable to execute the specified keyword (binary).
Cause: This error occurs for one of the following reasons:
- The binary does not have execute permission
- The binary name was misspelled
- The binary does not exist

Action: Correct the problem and reissue the keyword.

Table 6 (Page 4 of 6). Possible Causes for LoadLeveler Failure

Label: LL_INSTL_CUST_ER
Error Type: PERM
Diagnostic Explanation: Cannot open file *filename*

Explanation: LoadLeveler was unable to open the specified file.
Cause: This error occurs for one of the following reasons:
- Incorrect permissions are set for the file
- The file does not exist

Action: Correct the problem with file.

Label: LL_RESTART_ER
Error Type: PERM
Diagnostic Explanation: Exceeded nn restarts / hour

Explanation: **LoadL_master** exceeded the maximum allowed restarts for a daemon, as defined by the **RESTARTS_PER_HOUR** keyword.
Cause: The **LoadL_master** daemon died after exceeding the **RESTARTS_PER_HOUR** limit.
Action: Check the LoadLeveler error logs and the administrator mail for more information.

Label: LL_CRASH_ER
Error Type: PERM
Diagnostic Explanation: The *daemon_name* (process *pid*) died

Explanation: The specified daemon died.
Cause: This error occurs for one of the following reasons:
- The daemon received a signal
- The daemon experienced an unrecoverable error

Action: Check the LoadLeveler error logs for more information.

Label: LL_INFO_ST
Error Type: UNKN
Diagnostic Explanation: Started *daemon_name*, pid and pgroup=*pid*

Explanation: The specified daemon started.
Action: None. This is an informational message.

Table 6 (Page 5 of 6). Possible Causes for LoadLeveler Failure

Label: LL_INFO_ST
Error Type: UNKN
Diagnostic Explanation: Got SHUTDOWN command.
Got RECONFIG command.

Explanation: **LoadL_master** received the specified commands.
Cause: The **llctl** command was issued with “stop” or “reconfig”.
Action: None. This is an informational message.

Label: LL_INFO_ST
Error Type: UNKN
Diagnostic Explanation: Alternate central manager *cent_manager* will become active central manager. Switching central manager to *cent_manager*.

Explanation: LoadLeveler performed central manager recovery.
Cause: The backup central manager was unable to contact the primary central manager.
Action: Check the LoadLeveler error logs and the administrator mail for more information.

Label: LL_INFO_ST
Error Type: UNKN
Diagnostic Explanation: Alternate central manager *cent_manager* will return to stand-by state.

Explanation: LoadLeveler performed central manager recovery.
Cause: The backup central manager gave control back to the primary central manager.
Action: None. This is an informational message.

Label: LL_INFO_ST
Error Type: UNKN
Diagnostic Explanation: *machine* is unable to serve as alternate central manager.

Explanation: The machine specified cannot be the alternate central manager because the appropriate negotiator and collector keywords are not defined for this machine, or they are defined incorrectly.
Cause: The specified machine was defined as the central manager.
Action: Define the appropriate keywords.

Table 6 (Page 6 of 6). Possible Causes for LoadLeveler Failure

Label: LL_TRUNCATE_ST
Error Type: UNKN
Diagnostic Explanation: Logfile *filename* was truncated.

Explanation: LoadLeveler truncated the specified log file because the file exceeded the size defined in the **LoadL_config** file.
Action: None. This is an informational message.

Chapter 7. Routing Jobs to NQS Machines

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

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

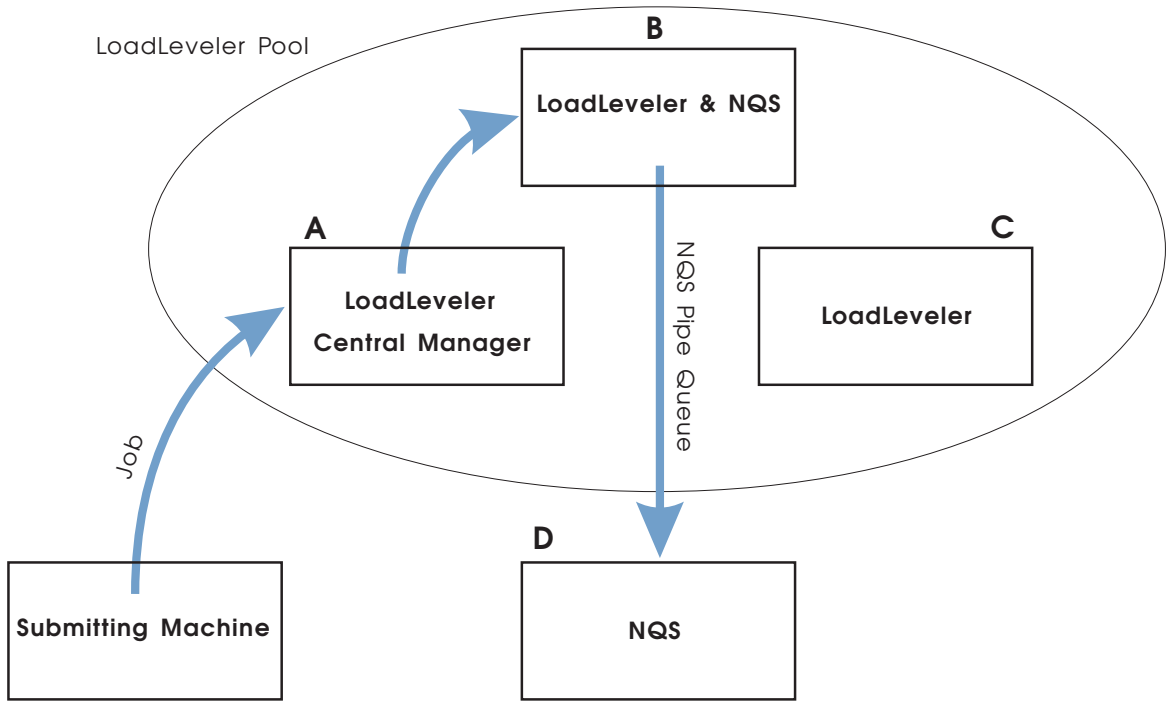


Figure 13. Environment illustrating jobs being routed to NQS machines.

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

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

Setting Up the NQS Environment

Setting up the NQS environment involves the following:

- Install NQS on each node that an NQS class is defined.

  In the previous diagram, this would be machine B.

- Create an NQS pipe queue on the LoadLeveler machine whose destination is the NQS batch queue on the machine designated to run the NQS jobs.

© Copyright IBM Corp. 1993, 1995

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

- Create an NQS batch queue on the machine designated to run the NQS jobs.

  In the previous diagram, this would be machine D.

Designating Machines to Which Jobs Will be Routed

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

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

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

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

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

   You can set up multiple classes to access different machines.

2. Modify the local configuration file on the machines that you want to accept this class of jobs.

3. Add the **NQS_DIR** keyword to the **LoadL_admin** file:

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

Sample Routing Jobs to NQS Machines Scenario

The following example walks you through the process of setting up your environment to route jobs to machines that run NQS.

Assume Figure 13 on page 121 depicts your environment. You have three machines in the cluster named A, B, and C. Outside of the cluster, you have machine D running NQS.

Task 1: Modify the Administration File

After setting up your NQS environment, modify the **LoadL_admin** file by defining the class **NQS** including the following stanzas:

```
NQS:
 type = class
 NQS_class = true
 NQS_submit = pipe_a
 NQS_query = queue@chevy.kgn.ibm.com
```

Task 2: Modify the Configuration File

Modify the **LoadL_config.local** on the machine(s) that you want to accept this class of jobs. In this example, you would modify machine B's **LoadL_config.local** file. To do this, add a class statement similar to:

```
CLASS = {"NQS" "a" "b" ....}
```

where `NQS` is the name of the class of jobs that will be routed to the machines that run NQS, and `a` and `b` are names of additional classes.

Task 3: Submit the Jobs

After you perform the previous tasks, users can route their jobs to machines running NQS using the **llsubmit** command. The job command file must specify the **class** keyword. For example:

```
class = NQS
```

The job command file must also contain the shell script to be submitted to the NQS node. NQS accepts only shell scripts, binaries are not allowed. All options in the command file pertaining to scheduling the job will be used by LoadLeveler to schedule the job. When the job is dispatched to the node running the specified NQS class, the LoadLeveler options pertaining to the runtime environment are converted to NQS options and the job is submitted to the specified NQS queue.

LoadLeveler command file options are used as follows:

arguments	error message generated and job not submitted
checkpoint	error message generated and job not submitted
class	used only for LoadLeveler scheduling
core_limit	converted to **-lc** option
cpu_limit	converted to **-lt** option
data_limit	converted to **-ld** option
environment	if COPY_ALL is specified, the option is converted to **-x**, otherwise error message generated and job not submitted
error	converted to **-e**
executable	error message generated and job not submitted
file_limit	converted to **-lf** option
hold	used only for LoadLeveler scheduling
image_size	error message generated and job not submitted
initialdir	error message generated and job not submitted
input	error message generated and job not submitted
notification	If the option specified is
	always converted to **-mb**and **-me** options
	error converted to **-me** option
	start converted to **-mb** option
	never ignored
	complete converted to **-me** option

notify_user	converted to **-mu** option
output	converted to **-o** option
preferences	used only for LoadLeveler scheduling
queue	places one copy of job in the LoadLeveler queue
requirements	used only for LoadLeveler scheduling
restart	If the option specified is
	yes ignored
	no converted to **-nr** option
rss_limit	converted to **-lw** option
shell	converted to **-s** option
stack_limit	converted to **-ls** option
start_date	used only for LoadLeveler scheduling
user_priority	used only for LoadLeveler scheduling

Users can also submit an NQS script. In this case, any NQS options in the script are used to schedule the job and once dispatched by LoadLeveler, the file is sent to NQS unmodified.

LoadLeveler schedules these jobs the same as it schedules other jobs. When the job is dispatched, LoadLeveler determines whether or not it is running in an NQS class. If it is, an NQS command **qsub** is issued.

LoadLeveler monitors the job by periodically invoking a **qstat** command. A **qstat** command is first issued for the pipe queue on the local host. If the request id is not found, a **qstat** is issued for each queue listed in the NQS_query class keyword. If the request id is still not found, starter marks the job as complete.

When a job is sent to an NQS class, **llsubmit** saves the following environment variables:

- HOME
- LOGNAME
- MAIL
- PATH
- SHELL
- TZ
- USER

When LoadLeveler dispatches the job, these environment variables are installed so that they are available to **qsub**. **llsubmit** also saves the name of the current directory (pwd) and the current value of the user file create mask (umask).

Task 4: Obtain Status of NQS Jobs

Users can obtain status of NQS jobs in the same way as they obtain status of LoadLeveler jobs - either by using the **llq** command or by viewing the Jobs window on the graphical user interface. The users can identify the NQS jobs by the class field on the Jobs window.

LoadLeveler monitors the job until **qstat** shows the job is no longer in either queue. If LoadLeveler loses contact with NQS, a message is written to the shadow log file and the jobs associated with those queues will be considered complete.

NQS does not provide job accounting. Therefore, the only accounting information LoadLeveler will have is the total time for the job.

LoadLeveler will not send mail when the job completes. The LoadLeveler notification option is translated to the appropriate NQS flag (me or mb) and NQS will send the mail.

Task 5: Cancel NQS Jobs

Users can cancel NQS jobs using the LoadLeveler **llcancel** command. All they need to know is the LoadLeveler job id for the NQS job. Once they submit their request to cancel the job, LoadLeveler forwards their request to the appropriate node and a **qdel** will be issued for the job for the queue listed in the the **NQS_submit** and **NQS_query** keywords.

NQS Scripts

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

a	startdate
e	error
ke	ignored
ko	ignored
lc	core_limit
ld	data_limit
lf	file_limit
lm	rss_limit
lM	ignored
ln	ignored
ls	stack_limit
lt	cpu_limit
lT	ignored
lv	ignored
lw	ignored

mb	notification (always)
me	notification (complete)
mu	notify_user
nr	restart = no
o	output
p	user_priority
q	class
r	ignored
re	ignored
ro	ignored
s	shell
x	environment = copyall
z	suppresses messages but not mail

Chapter 8. LoadLeveler Commands

LoadLeveler provides two types of commands: those that are available to all users of LoadLeveler, and those that are reserved for use by LoadLeveler administrators. The administrator commands can operate on the entire LoadLeveler job queue and all machines configured. The user commands affect only those jobs submitted by that user.

Some user commands, such as **llhold**, include functions that can only be performed by an administrator identified by the **LOADL_ADMIN** keyword in the configuration file. Refer to “Step 1: Define LoadLeveler Administrators” on page 41 for more information.

© Copyright IBM Corp. 1993, 1995

llacctmrg - Collects machine history files

Purpose

Collects all individual machine history files together into a single file specified as a parameter. The default is to collect data from all the machines identified in the **LoadL_admin** file but you can also specify a list of machines as a parameter to gather information on a subset of machines.

Syntax

```
llacctmrg -H -v -h hostlist -d directory
```

where

-H Provides a short usage message.

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

-h *hostlist* A blank delimited list of machines from which to collect data. The default is all machines listed in LoadL_admin.

-d *directory* Directory to hold the new global history file. If not specified, the directory specified in the **GLOBAL_HISTORY** keyword in the configuration file is used.

Notes

llacctmrg collects all individual machine accounting files together into a single file. It defaults to collecting information from all machines identified in the LoadL_admin file. You can also get information on a subset of machines by providing a list of machines as a parameter to the command.

When the **llacctmrg** command ends, accounting information regarding the job is stored in a file called *globalhist.YYMMDDHHmm*:

YY indicates the year

MM indicates the month

DD indicates the day

HH indicates the hour

mm indicates the minute.

Information such as the amount of resources consumed by the job as well as other job related data is stored in this file.

Data on processes which fork child processes will be included in the file only if the parent process waits for the child process to end. Therefore, complete data may not be collected for jobs which are not composed of simple parent/child processes. For example, if a LoadLeveler job invokes an rsh command to execute some function on another machine, the resources consumed on the other machine will not be collected as part of the accounting data.

Examples

The following example collects data from machines named `mars` and `pluto`.

```
llacctmrg -h mars pluto
```

llcancel - Cancel a Submitted Job

Purpose

Cancels one or more jobs from the LoadLeveler queue.

Syntax

```
llcancel -H -v -q -u userlist -h hostlist host.jobid.stepid.
```

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

jobid Is the job id assigned to the job when it was submitted using the **llsubmit** command.

.*stepid* (delimited by dot) Is the step id assigned to the job when it was submitted using the **llsubmit** command. The default is to include all members of the job.

-u *userlist* Is a blank-delimited list of users. When used with the **-h** option, only the user's jobs monitored on the machines in the *hostlist* are cancelled. When used alone, only the user's jobs monitored by the machine issuing the command are cancelled.

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

Notes

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

When the **-h** flag is specified by a non-administrator, all jobs submitted from the machines in *hostlist* by the user issuing the command are cancelled.

When the **-h** flag is specified by an administrator, all jobs submitted by the administrator are canceled, unless the **-u** is also specified, in which case all jobs both submitted by users in *userlist* and monitored on machines in *hostlist* are cancelled.

Group administrators and class administrators cannot use the **-h** flag or the **-u** flag.

When cancelling a job from a submit-only machine, you must specify the machine name that scheduled the job. For example, if you submitted the job from machine

A, a submit-only machine, and machine B, a scheduling machine, scheduled the job to run, you must specify machine B's name in the cancel command.

Group administrators and class administrators cannot use the **-h** flag or the **-u** flag.

Examples

This example cancels the job step 3 that is part of the job 18 that runs on the machine named bronze:

```
llcancel bronze.18.3
```

This example cancels all the job steps that are a part of job 8 that are running on the machine named gold.

```
llcancel gold.8
```

llclass - Query Class Information

Purpose

Returns information about classes.

Syntax

```
llclass -H -v -l classlist
```

where

-H Provides a short usage message.

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

-l Specifies that a long listing be generated for each class for which status is requested. If **-l** is *not* specified, then the standard listing is generated.

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

Notes

If you have more than a few classes configured for LoadLeveler, consider redirecting the output to a file when you use the **-l** flag.

Each machine periodically updates the central manager with a snapshot of its environment. Since the information returned by **llclass** is a collection of these snapshots, all taken at varying times, the total picture may not be completely consistent.

Examples

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

```
llclass -l silver gold
```

Results

Standard Listing

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

MaxJobCPU The CPU limit for all the processes in a job of this class.

MaxProcCPU The CPU limit for processes in this class.

Free Slots The number of free slots for new jobs.

Max Slots The total number of slots available in this class.

Description The description of this class.

The Long Listing

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

Name	The name of the class
Priority	The system priority of this class relative to other classes.
admin	The list of administrators of this class.
NQS_class	Indicates whether this class is a gateway for an NQS system.
NQS_submit	The NQS queue where the job will be sumbitted.
NQS_query	The NQS queues to query where the job has been dispatched.
max_processors	The maximum number of processors than can be used for parallel jobs.
class_comment	The text supplied by the administrator describing this class.
job_cpu_limit	The CPU limit for all processes in a job of this class.
cpu_limit	The CPU limit for processes in this class.
data_limit	The maxmimum size of the data area used for processes in this class.
core_limit	The maximum core size.
file_limit	The maximum file size.
stack_limit	The maximum stack limit.
rss_limit	The maximum rss limit.
nice	The *nice* value of jobs in this class.
free	The number of free slots available to new jobs.
maximum	The total number of slots available in this class.

llctl - LoadLeveler Control

Purpose

Controls LoadLeveler daemons on all members of the LoadLeveler pool. This command sends a message to the central manager requesting that action be taken on the members of the LoadLeveler pool.

Syntax

```
llctl -H -q -h host recycle
           -g       reconfig
                    start
                    stop
                    purge list_of_machines
                    drain [schedd] [startd]
                    flush
                    suspend
                    resume [schedd] [startd]
                    version
                    capture eventname
```

where the flags have the following meanings:

-H Provides a short usage message.

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

-h *host* Indicates that the command should apply to only this machine in the LoadLeveler pool. If neither **-h** nor **-g** is specified, the default is the machine on which the **llctl** command is issued.

-g Indicates that the command should apply globally to all machines in the **LoadL_admin** file.

Flags must be specified before any of the following keywords are specified.

recycle Stops all LoadLeveler daemons and restarts them.

reconfig Forces all daemons to reread the configuration files. When a reconfig is issued, the machine does not accept any more jobs to be scheduled. The machine does wait for jobs that are running to complete but it does not schedule jobs that are currently in the queue. Once the running jobs have finished and their status has been sent, the machine re-reads the configuration file.

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

stop Stops the LoadLeveler daemons on the specified machine.

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

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

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

drain When issued alone, it allows no more LoadLeveler jobs to begin running on this machine or jobs to be submitted through this machine. When issued as **drain schedd**, it drains the **schedd** machine. When issued as **drain startd**, it drains the **startd** machine.

flush Terminates running jobs on this machine and sends them back to the system queue to await redispatch. No new jobs are redispatched to this machine until **resume** is issued. Forces a checkpoint if jobs are enabled for checkpointing. However, the checkpoint gets cancelled if it does not complete within a five minute period.

suspend Suspends all jobs on this machine.

resume Resumes all jobs on this machine. When issued alone, it resumes all jobs on the machine. When issued as **resume schedd**, it resumes the jobs on the **schedd** machine. When issued as **resume startd**, it resumes the jobs on the **startd** machine.

version Displays version and release data at the screen.

capture Captures accounting data for all jobs running on the designated machines and associates the name *eventname* with the data.

Examples

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

```
llctl -h iron stop
```

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

```
llctl -g start
```

This example causes the LoadLeveler daemons on machine *iron* to re-read the configuration files, which may contain new configuration information for the *iron* machine:

```
llctl -h iron reconfig
```

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

```
llctl -h iron capture day
```

You could capture accounting information on all the machines in the LoadLeveler pool by using the **-g** option or you could collect accounting information on the local machine by simply issuing the following:

```
llctl capture day
```

Capturing information on the local machine is the default. For more information, see “Collecting Job Resource Data Based on Events” on page 108.

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

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

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

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

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

llfavorjob - Reorder System Queue by Job

Purpose

Sets specified jobs to a higher system priority than all jobs that are not favored. If this command is issued against jobs that are already running, it has no effect. If the job vacates, however, and returns to the queue, the job gets re-ordered with the new priority.

This command also *unfavors* previously favored job(s), restoring the original priority, when you specify the **-u** flag.

Syntax

```
llfavorjob -H -u -v host.jobid.stepid
```

where

-H	Provides a short usage message.
-u	Unfavors previously favored jobs, requeuing them according to their original priority levels.
-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 machine to which the job was submitted (delimited by dot). The default is the local machine.
jobid	Is the job id assigned to the job by LoadLeveler when it was submitted using the **llsubmit** command.
.stepid	(delimited by dot) Is the job step id assigned to the job by LoadLeveler when it was submitted using the **llsubmit** command. The default is to include all members of the job.

Notes

If more than one job is affected by this command, then the jobs are ordered by the **SYSPRIO** expression and are scanned before the not favored jobs. However, favored jobs which do not match the job requirements with available machines may run after not favored jobs. This command remains in effect until reversed with the **-u** option.

Examples

This example assigns jobs 12.4 on the machine *iron* and 8.2 on *zinc* the highest priorities in the system, with the first job submitted being the highest.

```
llfavorjob iron.12.4 zinc.8.2
```

This example unfavors jobs 12.4 on the machine *iron* and 8.2 on the machine *zinc*.

```
llfavorjob -u iron.12.4 zinc.8.2
```

llfavoruser - Reorder System Queue by User

Purpose

Sets a user's job(s) to the highest priority in the system, regardless of the current setting of the job priority, or SYSPRIO. Jobs already running are not affected.

This command also *unfavors* the user's job(s), restoring the original priority, when you specify the **-u** flag.

Syntax

```
llfavoruser -H -v -u userlist
```

where

-H	Provides a short usage message.
-u	Unfavors previously favored users, requeuing their job(s) according to their original priority level(s). If **-u** is **not** specified, the user's job(s) are favored.
-v	Outputs the name of the command, release number, service level, service level date, and operating system used to build the command.
userlist	Is a blank-delimited list of users whose jobs are given the highest priority.

Notes

The user's jobs still remain ordered by **UserPrio** (which may cause jobs for the user to swap **SYSPRIO**). If more than one user is affected by this command, the jobs of favored users are ordered by **SYSPRIO** and are scanned before the jobs of not favored users. However, jobs of favored users which do not match job requirements with available machines may run after jobs of not favored users.

Examples

This example grants highest priority to all queued jobs submitted by users `ellen` and `fred` according to their time of submission.

```
llfavoruser ellen fred
```

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

```
llfavoruser -u ellen fred
```

Although this example includes the list of users, it does not have to.

llhold - Hold or Release a Submitted Job

Purpose

Places jobs in user hold or system hold and releases jobs from both types of hold. Users can only move their own jobs into and out of user hold. Only LoadLeveler administrators move jobs into and release them from system hold.

Syntax

```
llhold -H -v -q -s -u userlist -h hostlist host.jobid.stepid
                -r
```

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.

-s Puts job(s) in system hold. Only a system administrator can use this option.

If neither **-s** nor **-r** is specified, LoadLeveler puts the job(s) in user hold.

-r Releases a job from hold. A job in user hold is released unless it is also in system hold, where it remains. A job in system hold is released unless it is also in user hold, where it remains.

Only an administrator can release jobs from system hold. Only an administrator or the owner of a job can release it from user hold.

If neither **-s** nor **-r** is specified, LoadLeveler puts the job(s) in user hold.

-u *userlist* Is a blank-delimited list of users. When used with the **-h** option, only the user's jobs monitored on the machines in the *hostlist* are held or released. When used alone, only the user's jobs monitored on the **schedd** machine are held or released.

-h *hostlist* Is a blank-delimited list of machine names. All jobs monitored on machines in this list are held or released. When issued with the **-u** option, the *userlist* is used to further select jobs for holding or releasing.

When issued by a non-administrator, this option only acts upon jobs that user has submitted to the machines in *hostlist*.

When issued by an administrator, all jobs monitored on the machines are acted upon unless the **-u** option is also used. In that case, the *userlist* is also part of the selection process,

and only jobs both submitted by users in *userlist* and monitored on the machines in the *hostlist* are acted upon.

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

If the job was submitted from a submit-only machine, this is the name of the **schedd** machine that sent the job to the negotiator.

jobid Is the job id assigned to the job when it was submitted using the **llsubmit** command.

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

Notes

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

To ensure a job is released from both system hold and user hold, the administrator must issue the command with **-r** specified to release it from system hold. The administrator or the submitting user can reissue the command to release the job from user hold.

This command will fail if:

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

Examples

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

```
llhold 23.0 19.1
```

This example releases job 23, job step 0, job 19, job step 1, and job 20, job step 3 fron a hold state.

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

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

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

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

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

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

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

llinit - Initializes machines in the LoadLeveler Pool

Purpose

Initializes a new machine as a member of the LoadLeveler hardware resource pool. This program runs once on each machine during the installation process. It must be run by the user ID you have defined as the LoadLeveler user ID. The log, spool, and execute directories are created with the correct modes and ownerships. The **LoadL_config** and **LoadL_admin** files are copied from LoadLeveler's release directory to LoadLeveler's home directory. **LoadL_config.local** is copied from LoadLeveler's release directory to LoadLeveler's local directory.

Syntax

```
llinit -H -quiet -help -prompt -local pathname -release pathname
       -cm machine -debug
```

-H Provides a short usage message.

-quiet
Quiet mode. Information messages do not display. Error messages always display.

-help Displays a brief explanation of flags.

-prompt
Prompts or leads you through a set of questions that help you to complete the **llinit** command.

-local *pathname* Where *pathname* is the local directory on which to create the spool, execute, and log sub-directories. The default, if this flag is not used, is the home directory.

There must be a unique local directory for each LoadLeveler pool member.

-release *pathname* Where *pathname* is the release directory, where the LoadLeveler bin, lib, man, include, and samples subdirectories are located. The default, if this flag is not used, is the /**usr**/**lpp**/**LoadL**/**nfs** directory.

-cm *machine* Where *machine* is the central manager machine, where the **negotiator** and **collector** daemons run.

-debug Displays a large amount of messages, tracing the path through **llinit** during execution. This is intended for debugging purposes only.

Notes

llinit initializes a new machine as a member of the LoadLeveler resource pool by doing the following:

- Creates the following LoadLeveler subdirectories with permissions.
  - Spool subdirectory with permissions set to 700.
  - Execute subdirectory with permissions set to 1777.
  - Log subdirectory with permissions set to 775.

- Copies the **LoadL_admin** and the **LoadL_config** files from the release directory samples subdirectory into the loadl home directory.
- Copies the **LoadL_config.local** file from the release directory samples subdirectory into the local directory.
- Creates symbolic links from the loadl home directory to the spool, execute, and log subdirectories and the **LoadL_config.local** file in the local directory (if home and local directories are not identical).
- Create symbolic links from the home directory to the bin, lib, man, samples, and include subdirectories in the release directory.
- Creates symbolic links for the **llcc**, **llxlf**, and **llxlp** commands in the release directory.
- Updates the **LoadL_config** with the release directory name.
- Updates the **LoadL_admin** with the central manager machine name.

Before running **llinit** ensure that your HOME environment variable is set to LoadLeveler's home directory.

NOTE: To run llinit you must have:

- Write privileges in the LoadLeveler home directory
- Write privileges in the LoadLeveler release directory
- Write privileges in the LoadLeveler local directory.

For more information on **llinit**, see the LoadLeveler *Memo To Users*.

Examples

The following example initializes a machine, assigning /**var**/**loadl** as the local directory, /**usr**/**lpp**/**LoadL**/**nfs** as the release directory, and the machine named **bronze** as the central manager.

```
llinit -local /var/loadl -release /usr/lpp/LoadL/nfs -cm bronze
```

llprio - Changes the Priority of Submitted Job Steps

Purpose

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

Note: llprio does not affect a job step that is running, even if its priority is lower than other jobs steps. Note that the priority is not the UNIX *nice* priority.

Syntax

```
llprio -H -v -q  +integer  host.jobid.stepid
                 -integer
                 -p integer
```

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 step, 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 machine to which the job step was submitted (delimited by dot). The default is the local machine.

If the job step was submitted from a submit-only machine, this is the name of the machine where the **schedd** daemon that sent the job to the negotiator resides.

jobid Is the job id assigned to the job when it was submitted using the **llsubmit** command.

.*stepid* (delimited by dot) Is the job step id assigned to the job when it was submitted using the **llsubmit** command.

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

Examples

This example raises the priority of job 4, job step 1 submitted to machine bronze by a value of 25.

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

This example sets the priority of job 18, job step 4 submitted to machine silver to 100, the highest possible value.

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

llq - Query Job Status

Purpose

Returns information about jobs that have been dispatched.

Syntax

```
llq -H -v -x -s -l joblist -u userlist -h hostlist -c classlist
```

where

-H Provides a short usage message.

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

-x Provides extended information about the selected job. If the **-x** flag is used with the **-h**, **-s**, or **-u** flag, an error message is generated.

CPU usage and other resource consumption information on active jobs can only be reported using the -x flag if the LoadLeveler administrator has enabled it by specifying A_ON and A_DETAIL for the ACCT macro in the LoadLeveler configuration file.

Normally, **llq** connects with the central manager to obtain job information. When you specify **-x**, **llq** connects to the **schedd** machine that received the specified job to get extended job information.

When specified without **-l**, CPU usage for active jobs is reported in the short format. Using **-x** can produce a very long report and can cause excess network traffic.

-s Provides information why a selected list of jobs remain in Idle state. Along with this flag, users must specify a list of jobs. The user can also optionally supply a list of machines to be considered when determining why the job(s) cannot run. If a list of machines is not provided, the default is the list of machines in the LoadLeveler pool. For each job, **llq** determines why the job remains in Idle state instead of running.

-l Specifies that a long listing be generated for each job for which status is requested. Detailed information can only be displayed for jobs belonging to the user issuing the **llq** command. If status is requested for a job which does not belong to the user, an error message is generated and no further output is displayed. Administrators can always display detailed information about any job. Fields included in the long listing are shown in “Results” on page 147. Only the owner of a job and the administrator can use this option.

If **-l** is *not* specified, then the standard listing is generated.

joblist is composed of *host.jobid.stepid* where:

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

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

-u *userlist* Is a blank-delimited list of users. When used with the **-h** option, only the user's jobs monitored on the machines in the *hostlist* are queried. When used alone, only the user's jobs monitored on the **schedd** machine are queried.

-h *hostlist* Is a blank-delimited list of machines. If the **-s** flag is not specified, all jobs monitored on machines in this list are queried. If the **-s** flag is specified, the list of machines is considered when determining why a job remains in Idle state. When issued with the **-u** option, the *userlist* is used to further select jobs for querying.

-c *classlist* Is a blank-delimited list of classes. If the **-s** flag is specified, this value is ignored. When used with **-h**, only those jobs monitored on the machines in the *hostlist* are queried. When used alone, only the classes specified on the **schedd** machine are queried.

Notes

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

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

Examples

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

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

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

```
llq 12
```

Results

Standard Listing

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

Id job identifier presented as *host.jobid.stepid* number. The job id may be shortened if the job id is name which would be too long to fit into the standard format. The name is shortened by removing the domain name from the host portion of the job id. In this case, a dash "-" is added to the shortened name to indicate that it was shortened. To see the full job id, run **llq** with the **-l** flag.

Owner userid of the job submitter.

Submitted date and time of job submission.

ST current job status, 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.
- *D* indicates a parallel job is deferred. Job has been placed in a temporary (deferred) queue while waiting to re-enter the LoadLeveler queue at its original priority. A parallel job enters a deferred queue when LoadLeveler cannot allocate the appropriate nodes for the job in a time period specified by the administrator.
- *C* indicates job completed. This only occurs when multiple jobs have been submitted from a job command file.
- *RM* indicates job removed.
- *RP* indicates remove pending, meaning the job is about to be removed.
- *S* indicates job has been put in system hold. See “llhold - Hold or Release a Submitted Job” on page 139.
- *H* indicates job has been put in user hold. See “llhold - Hold or Release a Submitted Job” on page 139
- *SH* indicates job has been put in user hold and system hold. .

PRI user priority of the job, where the values are defined in the **llprio** command. See “llprio - Changes the Priority of Submitted Job Steps” on page 143.

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 machine.
- *Idle* indicates job is not running or is scheduled to start running.
- *Pending* indicates that the job has been dispatched to a machine but is not yet running.
- *Deferred* indicates that the job has been temporarily put on hold until certain conditions have been met.
- *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.
- *Remove Pending* indicates the job is about to be removed.
- *System Hold* indicates job has been put in system hold. See "llhold - Hold or Release a Submitted Job" on page 139
- *User Hold* indicates job has been put in user hold. See "llhold - Hold or Release a Submitted Job" on page 139.

Completion Date date and time job was completed.

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

user_sysprio user system priority of the job, where the value is defined in the administration file.

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

group_sysprio group priority of the job, where the value is defined in the administration files.

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

Notifications notification status for the job, where:

- *always* notification is sent through mail for all four job status categories below.
- *complete* notification is sent through mail only when job completes.
- *error* notification is sent through mail only when job terminates abnormally.

never notification is never sent.

start notification is sent through mail only upon starting or restarting job.

Virtual Image Size of the executable that was submitted.

Checkpoint checkpoint status (yes or no)

Restart restart status (yes or no)

Hold Job Until job is 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. Appears only when the **-x** option is specified.

In file to be used for stdin.

Out file to be used for stdout.

Err file to be used for stderr.

Init Working Directory directory from which the job is run. The relative directory from which the stdio files are accessed, if appropriate.

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.

Other fields found when issuing **llq -x -l** are:

Account No account number specified in the job command file.

Group login group name.

maxrss Maximum resident set sized utilized.

ixrss Amount of shared memory used.

idrss Amount of unshared memory used.

isrss	Integral unshared stack used.
minflt	# Page faults (re-claimed).
majflt	# Page faults (I/O required).
nswap	# times swapped out.
inblock	# times file system performed input.
oublock	# times file system performed output.
msgsnd	# of IPC messages sent.
msgrcv	# of IPC messages received.
nsignals	# of signals delivered.
nvcsw	# of context switches due to voluntarily giving up processor.
nivcsw	# of involuntary context switches.

llstatus - Query Machine Status

Purpose

Returns status information about machines in the LoadLeveler pool. It does not provide status on any NQS machine.

Syntax

```
llstatus -H -v -l hostlist
```

where

-H Provides a short usage message.

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

-l Specifies that a long listing be generated for each machine for which status is requested. If **-l** is *not* specified, the standard list, described below, is generated.

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

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 machines named silver and gold.

```
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.

Schedd state of the daemon.

InQ number of jobs in the queue that were scheduled from this machine.

Act number of jobs that the schedd has dispatched.

Startd state of the daemon which can be one of the following:

- *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* all jobs in this machine have been suspended by the administrator.

Run number of jobs running on this machine.

LdAvg Berkeley one-minute load average on this machine.

Idle number of seconds since keyboard or mouse activity in a login session was detected. Highest number displayed is 9999.

Arch hardware architecture of machine as listed in configuration file.

OpSys operating system on this machine.

The Long Listing

The long listing, 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.

KILL the expression, defined following C conventions in the configuration file, that evaluates to true or false (T/F). This determines whether running jobs should be killed on this machine.

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.

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

Running number of jobs submitted to this machine that are 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:

- *Avail* The machine is available to receive and dispatch jobs. This state applies only to machines with the **schedd** daemon.
- *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.
- *Drained* The **schedd** and the **startd** daemons have been drained by the administrator. No new jobs can start until the administrator resumes these two daemons.
- *Draining* The **schedd** and the **startd** daemons have been drained by the administrator but some jobs are still running. The state of the machine will remain this way until all running jobs complete at which

time it will change to **drain**. No new jobs will be started.

Idle indicates that the job is not running or is scheduled to run or is starting to run. This state applies only to machines with the **startd** daemon.

Notrun The job cannot be run because a dependency cannot be satisfied.

Pending indicates that the job has been dispatched to a machine but it has not yet started to run.

Removed This job has been logically removed but is not physically removed from the queue yet. This only occurs when multiple jobs have been submitted to run from the job command file.

Running The job is running on a machine.

Starting The job is waiting to be started.

Terminated The completed or removed job is being removed from the queue.

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

Virtual Memory available swap space, in kilobytes, 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.

LoadAvg Berkely one-minute load average on machine.

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 all features on this machine.

Memory physical memory, in megabytes, on this machine.

Max_Starters maximum number of jobs that can run simultaneously on this machine.

Config Time Stamp
date and time of last (re)configuration.

Cpus number of CPUs on this machine.

Speed speed associated with the machine.

Subnet TCP/IP subnet that this machine resides on.

CustomMetric number that indicates the order of the machines for scheduling purposes.

ScheddRunning number of jobs running on this machine.

Pending number of jobs in this state on this machine.

Starting number of jobs in this state on this machine.

Idle	number of jobs in this state on this machine.
Unexpanded	number of jobs in this state on this machine.
Held	number of jobs in this state on this machine.
Removed	number of jobs in this state on this machine.
RemovedPending	number of jobs in this state on this machine.
Completed	number of jobs in this state on this machine.
DependentNotRun	number of jobs in this state on this machine.
Total Jobs	number of total jobs submitted to this machine.
ScheddState	state of the schedd on this machine.
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.

For information on writing a program to filter job scripts when they are submitted, see "User Exit: Filtering a Job Script" on page 237.

Syntax

```
llsubmit -H -v -q cmdfile
                  -
```

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 job command file named *qtrlyrun.cmd* is submitted.

```
llsubmit qtrlyrun.cmd
```

llsummary - Returns job resource information

Purpose

Returns job resource information regarding completed jobs.

Syntax

```
llsummary -H -v -x -l -s MM/DD/YY to MM/DD/YY -e MM/DD/YY to MM/DD/YY
          -u user -c class -g group -G unixgroup -a allocated
          -r report -j jobid -d section file
```

where

-H Provides a short usage message

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

-x Provides extended information. Using **-x** can produce a very long report. This option is meaninful only when used with the **-l** option. You must enable the recording of accounting data in order to collect information with the **-x** flag. To do this, specify `ACCT=A_ON A_DETAIL` in your **LoadL_config** file. For more information, see "Step 7: Define Job Accounting" on page 50.

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

-s Specifies a range for the start date (queue date) for accounting data to be included in this report. The format for entering the date is either `MM/DD/YY` (where `MM` is month, `DD` is day, and `YY` is year), or is a string of digits representing the number of seconds since 1970. The default is to report all the data in the report.

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

-u *user* Specifies the user ID for whom accounting data is reported.

-c *class* Specifies the class for which accounting data is reported.

-g *group* Specifies the LoadLeveler group for which accounting data reported is reported.

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

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

-r *report* Specifies the report type. You can choose one or more of the following:

- **resource** Provides CPU usage.
- **avgthroughput** Provides average queue time, run time, and CPU time
- **maxthroughput** Provides maximum queue time, run time, and CPU time
- **minthroughput** Provides minimum queue time, run time, and CPU time
- **throughput** Selects all thorughput reports
- **numeric** Reports CPU times in seconds rather than hours, minutes, and seconds

-d *section* Specifies the category (data section) for which you want to generate a report. You can specify one or more of the following:

user
group
unixgroup
class
account
day
week
month
jobid
jobname
allocated

-j *jobid* The job ID for which accounting data is reported.

file The file containing the accounting data. If not specified, the default is the local history file on the machine from which the command was issued.

Reports

This command provides both a short and long version of output:

The Short Listing

The short listing is generated when you do not specify **-l**. **-r**, or **-d** with **llsummary**. This report includes summaries of the following data:

- Number of jobs, Total CPU usage, per user.
- Number of jobs, Total CPU usage, per class.
- Number of jobs, Total CPU usage, per group.
- Number of jobs, Total CPU usage, per account number.

The following is an example of the short version:

```
    Name   Jobs   Steps        Job Cpu    Starter Cpu     Leverage
 brels4d      1       1     0+00:08:00     0+00:00:00  (undefined)
   TOTAL      1       1     0+00:08:00     0+00:00:00  (undefined)

   Class   Jobs   Steps        Job Cpu    Starter Cpu     Leverage
No_Class      1       1     0+00:08:00     0+00:00:00  (undefined)
   TOTAL      1       1     0+00:08:00     0+00:00:00  (undefined)

   Group   Jobs   Steps        Job Cpu    Starter Cpu     Leverage
    NONE      1       1     0+00:08:00     0+00:00:00  (undefined)
   TOTAL      1       1     0+00:08:00     0+00:00:00  (undefined)

 Account   Jobs   Steps        Job Cpu    Starter Cpu     Leverage
    NONE      1       1     0+00:08:00     0+00:00:00  (undefined)
   TOTAL      1       1     0+00:08:00     0+00:00:00  (undefined)
```

The explanations for the fields in the short report follow:

Name Userid submitting jobs.

Class Class specified or defaulted for the jobs.

Group User's login group.

Account Account number specified for the jobs.

Jobs Count of the total number of jobs submitted by this user, class, group, or account.

Starter CPU Total CPU time consumed by LoadLeveler starter processes on behalf of the user jobs.

Job CPU Total CPU time consumed by user's jobs.

Leverage Ratio of job CPU to starter CPU.

The Long Listing

The long listing is returned by **llsummary** when you specify the **-l** option:

llsummary

```
================== Job charlie 1 =================
             Job Id: charlie 1
           Job Name: charlie.1
              Owner: brels4d
              Group: staff
    Submitting host: charlie
  Submitting Userid: 18039
 Submitting Groupid: 100
    Number of steps: 1
------------------------- Step charlie 1.0 -----------------
                  Job Step Id: charlie 1.0
                    Step Name: 0
                   Queue Date: Mon Aug  1 22:59:27 1994
          Job Step Dependency:
                       Status: Completed
               Completion Date: Mon Aug  1 23:11:39 1994
               Completion Code: 0
                   Start Count: 1
                 User Priority: 0
                 Notifications: Always
           Virtual Image Size: (not recorded)
                    Checkpoint: no
                Hold Job Until: (no hold)
                           Cmd:
                          Args:
                           Env: LOADL_CORESIZE = 1024
                            In: /dev/null
                           Out: job4.out.1
                           Err: job4.err.1
    Initial Working Directory: /u/brels4d/cmr/samples
                Requirements: (Arch == "R6000") && (OpSys == "AIX32")
                 Preferences:
                    Job Type: Serial
          Allocated host name: charlie
                  Notify User: brels4d@charlie
                        Shell: /bin/ksh
                        Class: No_Class
           Job Cpu Hard Limit: 0 seconds
           Job Cpu Soft Limit: 0 seconds
```

```
                Account: NONE
        Job tracking exit:
         Job tracking arg:
        Starter User Time:   0+00:00:00.180000
      Starter System Time:   0+00:00:00.450000
       Starter Total Time:   0+00:00:00.630000
           Starter maxrss: 720
            Starter ixrss: 4224
            Starter idrss: 4496
            Starter isrss: 0
           Starter minflt: 244
           Starter majflt: 61
            Starter nswap: 0
          Starter inblock: 0
          Starter oublock: 0
           Starter msgsnd: 14
           Starter msgrcv: 14
         Starter nsignals: 4
            Starter nvcsw: 119
           Starter nivcsw: 0
             Job User Time:   0+00:08:00.001040
           Job System Time:   0+00:00:00.000640
            Job Total Time:   0+00:08:00.001680
                Job maxrss: 588
                 Job ixrss: 394304
                 Job idrss: 4426568
                 Job isrss: 0
                Job minflt: 770
                Job majflt: 14
                 Job nswap: 0
               Job inblock: 0
               Job oublock: 0
                Job msgsnd: 4
                Job msgrcv: 3
              Job nsignals: 9
                 Job nvcsw: 72
                Job nivcsw: 0
```

When you specify the **-x** option in conjunction with the **-l** option on the **llsummary** command, the long report resembles the following:

llsummary

```
================== Job charlie 1 =================
             Job Id: charlie 1
           Job Name: charlie.1
              Owner: brels4d
              Group: staff
    Submitting host: charlie
  Submitting Userid: 18039
 Submitting Groupid: 100
    Number of steps: 1
-------------------------- Step charlie 1.0 -----------------
                  Job Step Id: charlie 1.0
                    Step Name: 0
                   Queue Date: Mon Aug  1 22:59:27 1994
          Job Step Dependency:
                       Status: Completed
               Completion Date: Mon Aug  1 23:11:39 1994
               Completion Code: 0
                   Start Count: 1
                 User Priority: 0
                 Notifications: Always
           Virtual Image Size: (not recorded)
                    Checkpoint: no
                Hold Job Until: (no hold)
                           Cmd:
                          Args:
                           Env: LOADL_CORESIZE = 1024
                            In: /dev/null
                           Out: job4.out.1
                           Err: job4.err.1
    Initial Working Directory: /u/brels4d/cmr/samples
                Requirements: (Arch == "R6000") && (OpSys == "AIX32")
                 Preferences:
                    Job Type: Serial
          Allocated host name: charlie
                  Notify User: brels4d@charlie
                        Shell: /bin/ksh
                        Class: No_Class
           Job Cpu Hard Limit: 0 seconds
           Job Cpu Soft Limit: 0 seconds
                      Account: NONE
            Job tracking exit:
             Job tracking arg:
------------------------------------ Detail for charlie.1.0 ------
                         Running host: charlie
                        Machine speed: 2.234560
                                Event: System
                           Event name: started
                         Time of Event: Mon Aug  1 23:02:56 1994
                     Starter User Time:   0+00:00:00.000000
                   Starter System Time:   0+00:00:00.000000
                    Starter Total Time:   0+00:00:00.000000
                        Starter maxrss: 0
                         Starter ixrss: 0
                         Starter idrss: 0
                         Starter isrss: 0
                        Starter minflt: 0
                        Starter majflt: 0
                         Starter nswap: 0
                       Starter inblock: 0
                       Starter oublock: 0
                        Starter msgsnd: 0
                        Starter msgrcv: 0
                      Starter nsignals: 0
                         Starter nvcsw: 0
```

```
                    Starter nivcsw: 0
                     Job User Time:   0+00:00:00.000000
                   Job System Time:   0+00:00:00.000000
                    Job Total Time:   0+00:00:00.000000
                        Job maxrss: 0
                         Job ixrss: 0
                         Job idrss: 0
                         Job isrss: 0
                        Job minflt: 0
                        Job majflt: 0
                         Job nswap: 0
                       Job inblock: 0
                       Job oublock: 0
                        Job msgsnd: 0
                        Job msgrcv: 0
                      Job nsignals: 0
                         Job nvcsw: 0
                        Job nivcsw: 0
                             Event: System
                        Event name: completed
                     Time of Event: Mon Aug  1 23:11:39 1994
                  Starter User Time:   0+00:00:00.180000
                Starter System Time:   0+00:00:00.450000
                 Starter Total Time:   0+00:00:00.630000
                     Starter maxrss: 720
                      Starter ixrss: 4224
                      Starter idrss: 4496
                      Starter isrss: 0
                     Starter minflt: 244
                     Starter majflt: 61
                      Starter nswap: 0
                    Starter inblock: 0
                    Starter oublock: 0
                     Starter msgsnd: 14
                     Starter msgrcv: 14
                   Starter nsignals: 4
                      Starter nvcsw: 119
                     Starter nivcsw: 0
                      Job User Time:   0+00:08:00.001040
                    Job System Time:   0+00:00:00.000640
                     Job Total Time:   0+00:08:00.001680
                         Job maxrss: 588
                          Job ixrss: 394304
                          Job idrss: 4426568
                          Job isrss: 0
                         Job minflt: 770
                         Job majflt: 14
                          Job nswap: 0
                        Job inblock: 0
                        Job oublock: 0
                         Job msgsnd: 4
                         Job msgrcv: 3
                       Job nsignals: 9
                          Job nvcsw: 72
                         Job nivcsw: 0
```

Examples

The following example requests summary reports (short listing) of all the jobs submitted on your machine between the days of September 12, 1994 and October 12, 1994:

```
llsummary -s 09/12/94 to 10/12/94
```

llsummary

Chapter 9. Administrative Uses for the Graphical User Interface

The end user can perform many tasks more efficiently and faster using the graphical user interface (GUI) but there are certain tasks that end users cannot perform unless they have the proper authority. If you are defined as a LoadLeveler administrator in the LoadLeveler configuration file then you are immediately granted administrative authority and can perform the administrative tasks discussed in this section. To find out how to grant someone administrative authority, see “Step 1: Define LoadLeveler Administrators” on page 41.

You can access the administrative commands that act on LoadLeveler through the **Admin** pulldown menu on both the Jobs and the Machines pane of the GUI. The **Admin** pulldown menu on the Jobs window corresponds to the command options available in the **llhold**, **llfavoruser**, and **llfavorjob** commands. The **Admin** pulldown menu on the Machines window corresponds to the command options available in the **llctl** command.

© Copyright IBM Corp. 1993, 1995

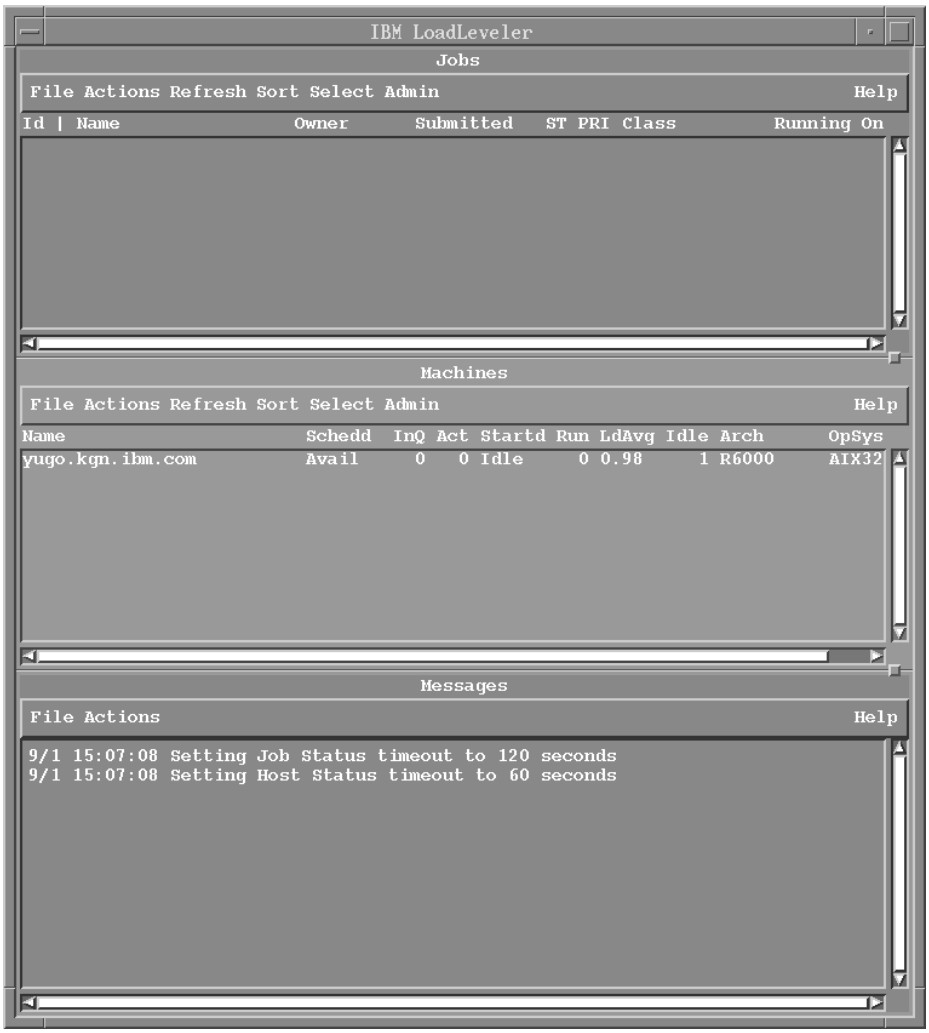


Figure 14. Main Window of the LoadLeveler GUI

The LoadLeveler GUI has three panes: one for job status with pull-down menus for job-related commands, one for machine status with pull-down menus for machine-related commands, and one for messages and logs. There are a variety of facilities available that allow you to sort and select the items displayed.

To invoke the LoadLeveler GUI, enter:

xloadl to invoke it in the foreground

xloadl & to invoke it in the background.

The LoadLeveler GUI differs from the Motif standard in some ways:

- Accelerators or mnemonics do not appear on the menu bars.

- Submerged windows do not necessarily rise to the top when refreshed.

The GUI resources are defined in the following files:

Xloadl_so for submit-only machines

Xloadl for all other machines.

For information on customizing your GUI, refer to Appendix B of the LoadLeveler User's Guide.

Job Related Administrative Actions

You access the administrative commands that act on jobs through the **Admin** pulldown menu in the jobs pane of the GUI.

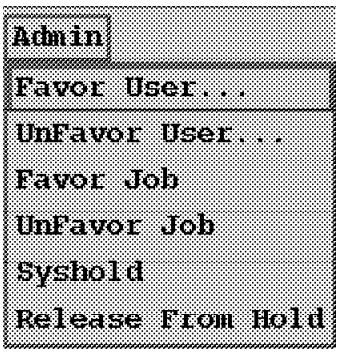


Figure 15. Administrator Commands: Job Pulldown

Using this pulldown menu, you can perform the following tasks:

Favor Users Allows you to favor users. This means that you can select one or more users whose jobs you want to move up in the job queue. This corresponds to the **llfavoruser** command.

Select **Admin** from the Jobs window

Select **Favor User**

▲The **Order by User** window appears.

Type in the name of the user for whom you want to favor their jobs.

Press OK

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

Select **Admin** from the Jobs window

Select **Unfavor User**

▲The **Order by User** window appears.

Type in the name of the user for whom you want to unfavor their jobs.

Press OK

Favor Jobs Allows you to select a job that you want to favor. This corresponds to the **llfavorjob** command.

Select one or more jobs from the Jobs window

Select **Admin** from the Jobs window

Select **Favor Jobs**

▲The selected jobs are favored.

Unfavor Jobs Allows you select a job that you want to unfavor. This corresponds to the **llfavorjob** command.

Select one or more jobs from the Jobs window

Select **Admin** from the Jobs window

Select **Unfavor Jobs**

▲Unfavors the jobs that you previously selected.

Syshold Allows you to place a system hold on a job. This corresponds to the **llhold** command.

Select a job from the Jobs window

Select **Admin** pulldown menu from the Jobs window

Select **Syshold** to place a system hold on the job.

Release From Hold Allows you to release the system hold on a job. This corresponds to the **llhold** command.

Select a job from the Jobs window

Select **Admin** pulldown menu from the Jobs window

Select **Release From Hold** to release the system hold on the job.

Machine Related Administrative Actions

You access the administrative commands that act on machines through the **Admin** pulldown menu in the machines pane of the GUI. You can perform these same actions using the **llctl** command. Refer to “llctl - LoadLeveler Control” on page 134 for information on this command and its options.

Using the GUI pulldown menu, you can perform the tasks described in this section.

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

Select **Admin** from the Machines window.

Select **Start All**

Start LoadLeveler Allows you to start LoadLeveler on selected machines.

Select one or more machines on which you want to start LoadLeveler.

Select **Admin** from the Machines window.

Select **Start LoadLeveler**

Stop LoadLeveler Allows you to stop LoadLeveler on selected machines.

Select on or more machines on which you want to stop LoadLeveler.

Select **Admin** from the Machines window.

Select **Stop LoadLeveler**.

Stop All Stops LoadLeveler on all machines listed in machine stanzas. Use this option when specifying alternate central managers.

Select **Admin** from the Machines window.

Select **Stop All**

reconfig forces all daemons to reread the configuration files.

Select the machine on which you want to operate.

Select **Admin** from the Machines window.

Select **reconfig**.

recycle stops all LoadLeveler daemons and restarts them.

Select the machine on which you want to operate.

Select **Admin** from the Machines window.

Select **recycle**.

drain allows no more LoadLeveler jobs to begin running on this machine but it does allow running jobs to complete.

Select the machine on which you want to operate.

Select **Admin** from the Machines window.

Select **drain**.

A cascading menu allows you to select either **daemons**, **schedd**, or **startd**. If you select **daemons**, both machines will be drained. If you select **schedd**, only the machine running this daemon will be drained. If you select **startd**, only the machine running this daemon will be drained.

flush terminates running jobs on this host and sends them back to the system queue to await redispatch. No new jobs are redispatched to this machine until **resume** is issued. Forces a checkpoint if jobs are enabled for checkpointing.

Select the machine on which you want to operate.

Select **Admin** from the Machines window.

Select **flush**.

suspend suspends all jobs on this host.

Select the machine on which you want to operate.

Select **Admin** from the Machines window.

Select **suspend**.

resume resumes all jobs on this machine.

Select the machine on which you want to operate.

Select **Admin** from the Machines window.

Select **resume**.

Capture Data collects information on the machines selected.

Select the machine on which you want to operate.

Select **Admin** from the Machines window.

Select **Capture Data**.

Collect Account Data collects accounting data on the machines selected.

Select the machine on which you want to operate.

Select **Admin** from the Machines window.

Select **Collect Account Data**.

A window appears prompting you to enter the name of the directory in which you want the collected data stored.

Create Account Report creates an accounting report for you.

Select **Admin** → **Create Account Report...**

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

A window appears prompting you to enter the following information:

- A short, long, or extended version of the output. The short version is the default.
- The user ID
- The class name
- The LoadL (LoadLeveler) group name
- The UNIX group name
- The Allocated host
- The job ID
- The report Type
- The section
- A start and end date for the report. If no date is specified, the default is to report all of the data in the report.
- The name of the input data file.

- The name of the output data file. This is the same as stdout.

Press **OK**

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

version displays version and release data for LoadLeveler on the machines selected in an information window.

Select the machine on which you want to operate.

Select **Admin** from the Machines window.

Select **version**.

Appendix A. Commonly Asked Questions and Answers

While using LoadLeveler, you may encounter problems or have questions that are not discussed or answered in the descriptive sections of this document or the User's Guide. This appendix provides diagnostic information as well as some helpful hints that can help you to solve these problems and answer your questions.

Refer to “Step 10: Record and Control Log Files” on page 53 for information on setting up error logs.

Why Won't My Job Run?

If you submitted your job and it is in the LoadLeveler queue but has not run, issue **llq -s** first to help diagnose the problem. If you need more help diagnosing the problem, refer to the following table:

Reason why Your job may not be running:	**To help determine the problem, check the following:**
Job requires specific machine, operating system, or other resource.	• Does the resource exist in the LoadLeveler pool? If yes, wait until it becomes available. Check the GUI to compare the job requirements to the machine details, especially **Arch**, **OpSys**, and **Class**. Ensure that the spelling and capitalization matches.
Job requires specific job class	• Is the class defined in the administration file? If yes, • Is there a machine in the pool that supports that class? If yes, you need to wait until the machine becomes available to run your job.
The maximum number of jobs are already running on the machine	Wait until the machine finishes a job to schedule yours.

The start expression evaluates to false.	Examine the configuration files (both **LoadL_config** and **LoadL_config.local**) to determine the **START** control function expression used by LoadLeveler to start a job. As a problem determination measure, set the START and SUSPEND values, as shown in this example:
	`START: T`
	`SUSPEND: F`

The priority of your job is lower than the priority of other jobs.	You cannot affect the system priority given to this job by the negotiator daemon but you can try to change your user priority to move this job ahead of other jobs you previously submitted using the **llprio** command or the GUI.
The information the central manager has about machines and jobs may not be current.	Wait a few minutes for the central manager to be updated and then the job may be dispatched. This time limit (a few minutes) depends upon the polling frequency and polls per update set in the **LoadL_config** file. The default polling frequency is five minutes.
You do not have the same userid on all the machines in the pool.	To run jobs on any machine in the pool, you have to have the same userid on every machine in the pool. If you do not have a userid on one machine, your jobs will not be scheduled to that machine.

You can use the **llq** command to query the status of your job or the **llstatus** command to query the status of machines in the pool. Refer to Chapter 8, “LoadLeveler Commands” on page 127 for information on these commands.

© Copyright IBM Corp. 1993, 1995

Why Won't My Parallel Job Run?

If you submitted your parallel job and it is in the LoadLeveler queue but has not run, issue **llq -s** first to help diagnose the problem. If issuing this command does not help, refer to the previous table and to the following table for more information:

Your job may not be running because:	To help determine the problem, check that the:
The number of parallel machines requested in the job command file is not available.	The number of parallel machines requested must be less than or equal to the number of parallel machines in at least one of the batch parallel sub-pools associated with the SP system **startd** machines.
The minimum number of processors requested by your job is not available.	Sufficient resources must be available. Specifying a smaller number of processors with the **min_processors** keyword may help if your job can run with fewer resources.
The pool in your **requirements** statement specifies a pool which is invalid or not available.	The specified pool must be valid and available.
The adapter specified in your **requirements** statement specifies an adapter which is invalid or not available.	The specified adapter must be valid and available.
PVM3 is not installed	PVM3 must be installed on any machine you wish to use for pvm. The PVM3 system itself is not supplied with LoadLeveler.
You are already running a PVM3 job on one of the LoadLeveler machines.	PVM3 restrictions prevent a user from running more than one pvm daemon per user per machine. If you want to run pvm3 jobs on LoadLeveler, you must not run any pvm3 jobs outside of LoadLeveler control on any machine being managed by LoadLeveler.
The **parallel_path** keyword in your job command file is incorrect.	Use **parallel_path** to inform LoadLeveler where binaries that run your pvm tasks are for the pvm_spawn() command. If this is incorrect, the job may not run.
The **pvm_root** keyword in the administration file is incorrect.	This keyword corresponds to the pvm **ep** keyword and is required to tell LoadLeveler where the pvm system is installed.
The file /**tmp**/**pvmd.**<**userid**> exists on some LoadLeveler machine but no PVM jobs are running.	If PVM3 exits unexpectedly, it will not properly clean up after itself. Although LoadLeveler attempts to clean up after pvm, some situations are ambiguous and you may have to remove this file yourself. Check that all the systems specified as capable of running PVM3 and remove this file if it exists.

You can use the **llq** command to query the status of your job. Refer to "llq - Query Job Status" on page 145 for information on this command.

Why Won't My Submit-Only Job Run?

If a job you submitted from a *submit-only* machine does not run, verify that you have defined the following statements in the administration file of the submit-only machine:

```
schedd_host = true
submit_only = true
```

See "Sample 2" on page 209 for an example of an administration file.

What Happens to Running Jobs When the Machine Goes Down?

Both the **startd** daemon and the **schedd** daemon maintain persistent states of all jobs. Both daemons use a specific protocol to ensure that the state of all jobs is consistent across LoadLeveler. In the event of a failure, the state can be recovered. Neither the **schedd** nor the **startd** daemon discard the job state information until it is passed onto and accepted by another daemon in the process.

If:	Then:
The network goes down but the machines are still running	If the network goes down but the machines are still running, when LoadLeveler is restarted, it looks for all jobs that were marked running when it went down. On the machine where the job is running, the **startd** daemon searches for the job and if it can verify that the job is still running, it continues to manage the job through completion. On the machine where **schedd** is running, **schedd** queues a transaction to the **startd** to re-establish the state of the job. This transaction stays queued until the state is established. Until that time, LoadLeveler assumes the state is the same as when the system went down.
The network partitions or goes down.	All transactions are left queued until the recipient has acknowledged them. Critical transactions such as those between the **schedd** and **startd** are recorded on disk. This ensures complete delivery of messages and prevents incorrect decisions based on incomplete state information.
The machine with **startd** goes down.	Because job state is maintained on disk in **startd**, when LoadLeveler is restarted it can forward correct status to the rest of LoadLeveler. In the case of total machine failure, this is usually "JOB VACATED", which causes the job to be restarted elsewhere. In the case that only LoadLeveler failed, it is often possible to "find" the job if it is still running and resume management of it. In this case LoadLeveler sends "JOB RUNNING" to the **schedd** and central manager, thereby permitting the job to run to completion.
The central manager machine goes down.	All machines in the pool send current status to the central manager on a regular basis. When the central manager restarts, it queries each machine that checks in, requesting the entire queue from each machine. Over the period of a few minutes the central manager restores itself to the state it was in before the failure. Each **schedd** is responsible for maintaining the correct state of each job as it progressed while the central manager is down. Therefore, it is guaranteed that the central manager will correctly rebuild itself. All jobs started when the central manager was down will continue to run and complete normally with no loss of information. Users may continue to submit jobs. These new jobs will be forwarded correctly when the central manager is restarted.
The machine running the job goes down	The central manager thinks that the job is still running. When the machine comes back up, the job gets put back on the queue to be run again. This happens only if **restart** is set to yes.

If:	Then:
The **schedd** machine goes down	When **schedd** starts up again, it reads the queue of jobs and for every job which was in some sort of active state (i.e. PENDING, STARTING, RUNNING), it queries the machine where it is marked active. The running machine is required to return current status of the job. If the job completed while **schedd** was down, "JOB COMPLETE" is returned with exit status and accounting information. If the job is running, "JOB RUNNING" is returned. If the job was vacated, "JOB VACATED" is returned. Because these messages are left queued until delivery is confirmed, no job will be lost or incorrectly dispatched due to **schedd** failure. During the time the **schedd** is down, the central manager will not be able to start new jobs that were submitted to that **schedd**.
The **llsubmit** machine goes down	**schedd** gets its own copy of the executable so it does not matter if the **llsubmit** machine goes down.

Why Won't LoadLeveler Start?

If the **master** daemon will not run, go to the node where **LoadL_master** will not start and type from the command line:

```
LoadL_master -t
```

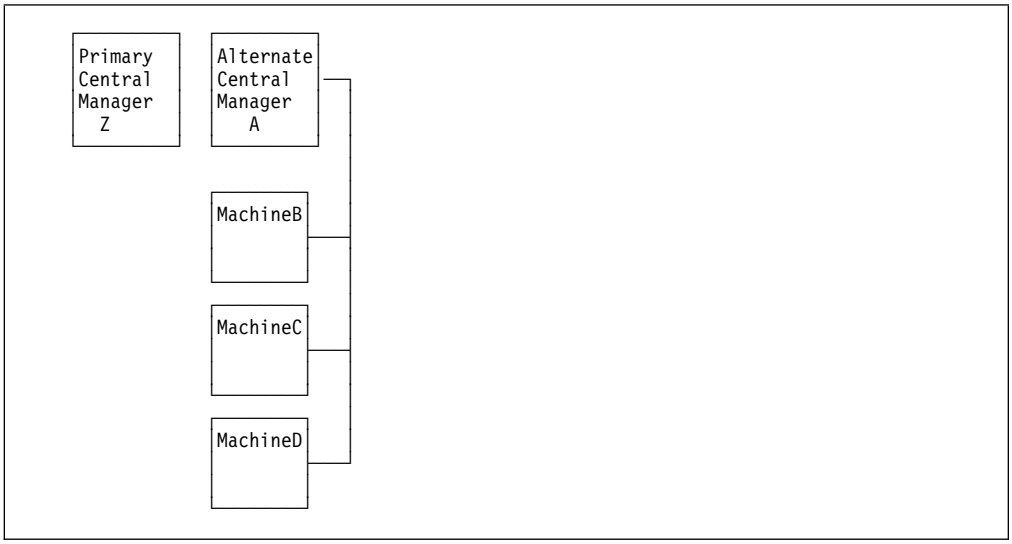
This generates messages that may help to diagnosis the problem. In addition, ensure that the:

1. Release and bin directory is properly specified in the configuration files.
2. Administration file exists and is properly defined in the configuration file.
3. Central manager is correctly defined in the administration file.
4. Log directories are correctly defined in the configuration file.
5. Log, spool, and execute directories exist and permissions are set as follows:
   - the **spool** subdirectory is set to 700
   - the **execute** subdirectory is set to 1777
   - the **log** subdirectory is set to 1770.
6. the **LoadL_master** binary is owned by root and has setuid bit set.
7. Daemons are not already running. To determine this, use the **ps** command.

What Happens if the Central Manager Isn't Operating?

In one of your machine stanzas specified in the administration file, you specified a machine to serve as the central manager. It is possible for some problem to cause this central manager to become unusable such as network communication or software or hardware failures. In such cases, the other machines in the LoadLeveler pool believe that the central manager machine is no longer operating. If you assigned one or more alternate central managers in the machine stanza, a new central manager will take control. The alternate central manager is chosen based upon the order in which its respective machine stanza appears in the administration file.

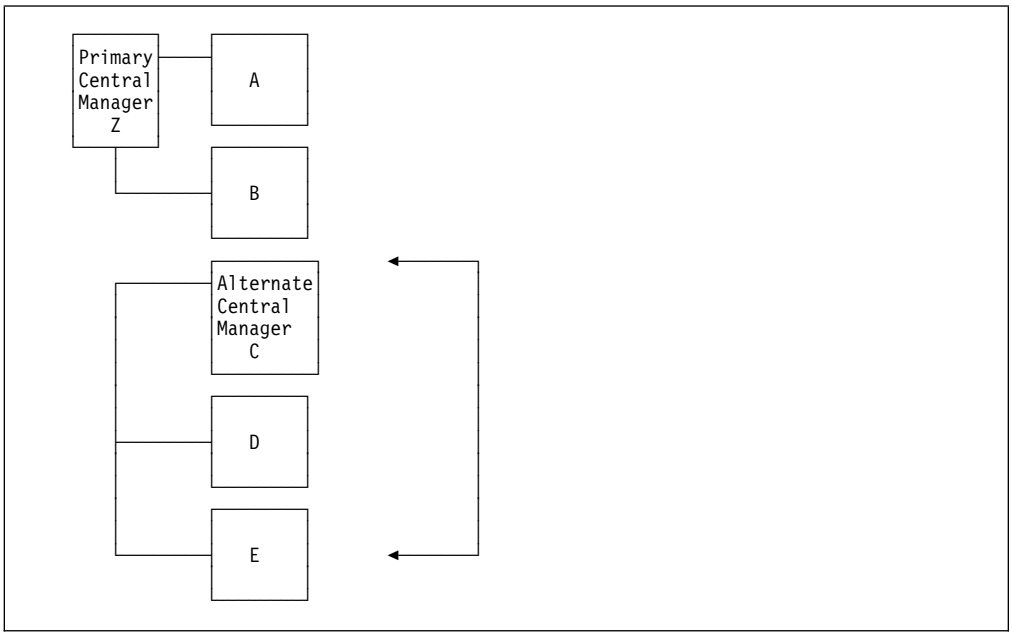
Once an alternate central manager takes control, it starts up its **negotiator** and **collector** daemons and notifies all of the other machines in the LoadLeveler pool that a new central manager has been selected. The following diagram illustrates how a machine can become the alternate central manager:



The diagram illustrates that Machine Z is the primary central manager but Machine A took control of the LoadLeveler pool by becoming the alternate central manager. Machine A remains in control as the alternate central manager until either:

- The primary central manager, Machine Z, resumes operation. In this case, Machine Z notifies Machine A that it is operating again and, therefore, Machine A terminates its **negotiator** and **collector** daemons.
- Machine A also loses contact with the remaining machines in the pool. In this case, another machine authorized to serve as an alternate central manager takes control. Note that Machine A may remain as its own central manager.

The following diagram illustrates how multiple central managers can function within the same LoadLeveler pool:



In this diagram, the primary central manager is serving machines A and B. Due to some network failure, machines C, D, and E have lost contact with the primary central manager machine and, therefore, machine C which is authorized to serve as an alternate central manager, assumes that role. Machine C remains as the alternate central manager until either:

- The primary central manager is able to contact machines C, D, and E. In this case, the primary central manager notifies the alternate central managers that it is operating again and, therefore, Machine C terminates its **negotiator** and **collector** daemons. The **negotiator** daemon running on the primary central manager machine is refreshed to discard any old job status information and to pick up the new job status information from the newly re-joined machines.
- Machine C loses contact with machines D and E. In this case, if machine D or E is authorized to act as an alternate central manager, it assumes that role. Otherwise, there will be no central manager serving these machines. Note that Machine C remains as its own central manager.

While LoadLeveler can handle this situation of two concurrent central managers without any loss of integrity, some installations may find administering it somewhat confusing. To avoid any confusion, you should specify all primary and alternate central managers on the same LAN segment.

For information on selecting alternate central managers, refer to “Step 1: Specify Machine Stanzas” on page 20

How Do I Move the Central Manager Function to Another Machine?

To move the central manager from one machine to another, follow these steps:

1. Obtain a list of all of the machines in the pool.
2. Change the **LoadL_admin** file to identify the new central manager location. If you maintain multiple copies of administration files, ensure that they are all the same.
3. Issue the **llctl** command with the **reconfig** option to all machines.

Keep in mind that jobs will not be lost but it may take a few minutes for all of the machines in the pool to check in with the new central manager machine so job status enquiries may be incorrect for a while.

Why Does llstatus Indicate that a Machine is Down but llq Indicates a Job is Running on The Machine?

If a machine fails while a job is running on the machine, the central manager does not change the status of any job on the machine. When the machine comes back up the central manager will be updated.

Why do I have to setuid = 0?

The **master** daemon starts the **startd** daemon and the **startd** daemon starts the **starter** process. The **starter** process runs the job. The job needs to be run by the userid of the submitter. You either have to have a separate **master** daemon running for every id on the system or the **master** daemon has to be able to su to every userid and the only userid that can su to any other userid is **root**.

How Do I Determine When My Job Started and Stopped?

You can determine the "wall clock" time for a job; that is, the time the job started and the time the job stopped. Suppose you submit a job and receive the following notification mail when the job finishes:

```
 Submitted at: Sun Apr 30 11:40:41 1995
 Exited    at: Sun Apr 30 12:49:10 1995

                Real Time:   0 01:08:29
      Job Step User Time:   0 00:30:15
    Job Step System Time:   0 00:12:55
     Total Job Step Time:   0 00:43:10

        Starter User Time:   0 00:00:00
      Starter System Time:   0 00:00:00
       Total Starter Time:   0 00:00:00
```

This mail tells you the following:

Submitted at The time you issued the **llsubmit** command or the time you submitted the job with the graphical user interface.

Exited at The actual time your job completed.

Real Time The wall clock time from submit to completion.

Job Step User Time The CPU time the job consumed executing in user space.

Job Step System Time The CPU time the system (AIX) consumed on behalf of the job.

Total Job Step Time The sum of the two fields above.

Starter User Time The CPU time consumed by the LoadLeveler starter process for this job, executing in user space. Time consumed by the starter process is the only LoadLeveler overhead which can be directly attributed to a user's job.

Starter System Time The CPU time the system (AIX) consumed on behalf of the LoadLeveler starter process running for this job.

Total Starter Time The sum of the two fields above.

You can also get the starting time by issing `llsummary -l -x` and then issuing `awk '/Date|Event/'` against the resulting file. For this to work, you must have `ACCT = A_ON A_DETAIL` set in the **LoadL_config** file.

Why Doesn't LoadLeveler Execute my .profile or .login Script?

When you submit a non-checkpointing batch job to LoadLeveler, the operating system will execute your **.profile** script before executing the batch job if your login shell is the Korn shell. On the other hand, if your login shell is the Bourne shell, on most operating systems (including AIX), the **.profile** script is not executed. Similarly, if your login shell is the C shell then AIX will execute your **.login** script before executing your LoadLeveler batch job but some other variants of UNIX may not invoke this script.

The reason for this discrepancy is due to the interactions of the shells and the operating system. To understand the nature of the problem, examine the following C program that attempts to open a login Korn shell and execute the "ls" command:

```
#include <stdio.h>
main()
{
  execl("/bin/ksh","-","-c","ls",NULL);
}
```

UNIX documentations in general (SunOS, HP-UX, AIX, IRIX) give the impression that if the second argument is "-" then you get a login shell regardless of whether the first argument is /bin/ksh or /bin/csh or /bin/sh. In practice, this is not the case. Whether you get a login shell or not is implementation dependent and varies depending upon the UNIX version you are using. On AIX you get a login shell for /bin/ksh and /bin/csh but not the Bourne shell.

If your login shell is the Bourne shell and you would like the operating system to execute your **.profile** script before starting your batch job, add the following statement to your job command file:

```
# @ shell = /bin/ksh
```

LoadLeveler will open a login Korn shell to start your batch job which may be a shell script of any type (Bourne shell, C shell, or Korn shell) or just a simple executable.

Can I Have Multiple Administration and Global Configuration Files?

In order to easily manage LoadLeveler, it is strongly recommended that you have only one administration file and one global configuration file, centrally located on a machine in the LoadLeveler pool. The machines in the pool must have read access to these files. LoadLeveler does not prevent you from having multiple copies of administration files but you need to be sure to update all the copies whenever you make a change to one. Having only one administration file prevents any confusion.

Similarly, you should have only one global configuration file. You can, however, have multiple local configuration files that specify information specific to individual machines. For more information on the global and local configuration files, refer to “Configuring LoadLeveler” on page 34.

Messages

This section lists only a few of the most common messages that you may receive.

errno = 2: No such file or directory

You may get this message when submitting a job from a directory that does not exist on all of the systems in your cluster. As a result, the job submission fails.

Warning: Error Receiving Job Data. The Query May be Incomplete.

If you get this message, you are probably trying to use submit-only hosts. Ensure that the userid submitting the command exists on the machine running the daemon. It may be that different userids are assigned on different machines for the same user or the user does not have a userid on the machine running the daemons. In addition, check the following:

1. Ensure that you have a common name space, so that each userid name and number effectively match in /**etc**/**passwd** on all participating LoadLeveler machines.
2. There may be an incompatibility between LoadLeveler installation levels.

Getting Help from IBM

Should you require help from IBM in resolving a LoadLeveler problem, you can get assistance by calling IBM Support. Before you call, be sure you have the following information:

1. Your access code (customer number).
2. The LoadLeveler product number for the:
   - RISC System/6000 and SP is 5765-145
   - Sun is 5765-227
   - SGI is 5765-228
   - HP is 5765-287
3. The name and version of the operating system you are using.
4. A telephone number where you can be reached.

In addition, issue the following command:

```
llctl version
```

This command will provide you with code level information. Provide this information to the IBM representative.

The number for IBM support in the United States is 1-800-CALL-AIX (800-225-5249) or 817-962-6902 and select option 8.

The Facsimile number is 817-962-6723.

Office hours are from 8:00 a.m. - 6:00 p.m. Central time Monday through Friday, excluding holidays. A voice mail message may be left if calling other hours. Note that customers can receive service 24 hours a day/every day under special agreement with IBM.

For specific information regarding AIX and AIX product support, ask your IBM representative to provide you with support information from the following document:

Title AIX SUPPORT FAMILY ENHANCEMENTS AND NEW AIX TECHNICAL SUPPORT STRUCTURE

Number 694-018

Date July 26, 1994

Category GI00, GI10, GD00, GD70

Appendix B. LoadLeveler Job Command File Keywords

This appendix instructs users how to build a job command file and provides an alphabetical list of the keywords they can use. For additional information on job command files, consult the *LoadLeveler User's Guide*.

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, as well as the job steps that you want to submit and can contain other LoadLeveler statements. To specify a binary to be executed, the **executable** keyword (described later) must be used. To specify a shell script to be executed, the **executable** keyword can be used. If it is not used, LoadLeveler assumes that the job command file iteself is the executable.

The job command file consists 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 step you want to submit is specified with the **executable** keyword.

  See "Job Command File Macros" on page 196 for information on invoking a macro from a keyword statement.

- 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 Build a Job window on the GUI or by using a text editor.

Note that LoadLeveler uses the syntax *host.jobid.stepid* to refer to a job name.

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
```

© Copyright IBM Corp. 1993, 1995

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

Specify **system_initiated** if you want LoadLeveler to automatically checkpoint your program at preset intervals.

Specify **system_initiated** if you want both system initiated and user initiated checkpointing. Specify **no** if you do not want your program to be checkpointed. The default is `no`.

Note: If you have existing job command files that specify **checkpoint=yes**, this will default to **system-initiated**.

Your program must be linked with the LoadLeveler libraries or LoadLeveler will reject the program at submit time.

The syntax is:

```
checkpoint = user_initiated | system_initiated | no
```

class

Specifies the name of a job class defined locally in your pool. 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 a class called *largejobs*, your class keyword would be similar to:

```
class = largejobs
```

comment

Describes characteristics or distinguishing features of the job.

core_limit

Specifies the maximum size of a core file. Refer to “Setting Limits” on page 61 for more information on hardlimits and softlimits.

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
```

Some examples are:

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

cpu_limit

Specifies the maximum amount of CPU time that a submitted job step can use. See “Setting Limits” on page 61 for an explanation of hardlimit and softlimit. Express the size as:

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

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

The syntax is:

```
data_limit = hardlimit,softlimit
```

For example:

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

dependency

Specifies the dependencies between job steps.

The syntax is:

```
dependency = expression
```

where the syntax for the `expression` is as follows:

```
step_name operator value
```

where `step_name` must be a previously defined job step (see “step_name” on page 196) and `operator` can be one of the following:

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

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

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

CC_REMOVED is the return code set by LoadLeveler for a job step which is removed from the system (e.g. because of llcancel against the job step.)

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

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

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

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

In the following example, step1 will run with the executable called `myprogram1`. Step2 will run if the return code of step1 equals zero. If the return code of step1 does not equal zero, step2 does not get executed. In this case, the dependency statement in step3 gets evaluated and it is determined that step2 did not run. Therefore, `myprogram3` gets executed.

```
# Beginning of step1
# @ step_name = step1
# @ executable = myprogram1
# @ ...
# @ queue
# Beginning of step2
# @ step_name = step2
# @ dependency = step1 == 0
# @ executable = myprogram2
# @ ...
# @ queue
# Beginning of step3
# @ step_name = step3
# @ dependency = step2 == CC_NOTRUN
# @ executable = myprogram3
# @ ...
# @ queue
```

environment

Specifies your initial environment variables when your job step starts. If you only specify this keyword in the first job step, its value is inherited to the other job steps. Separate environment specifications with semicolons. An environment specification may be one of:

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 step runs. If you only specify this keyword in the first job step, its value is inherited to the other job steps. If not specified, the file /**dev**/**null** is used.

The syntax is:

```
error = filename
```

For example:

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

executable

For serial jobs, **executable** identifies the name of the program to run. For parallel jobs, **executable** is used as follows:

- For Parallel Operating Environment (POE) jobs – specifies the POE partition manager.
- For Parallel Virtual Machine (PVM) jobs – specifies the name of your parallel job.
- For Parallel Virtual Machine extended (PVMe) jobs – specifies the PVMe daemon.

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

The syntax is:

```
executable = name
```

Examples:

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

```
# @ executable = /usr/lpp/pvmd3e  (For PVMe jobs)
# @ executable = my_parallel_job  (For PVM jobs)
```

Note that the **@ executable** statement automatically sets the **$base_executable** variable, which is the file name of the executable without the directory component.

file_limit

Specifies the maximum size of the file.

The syntax is:

```
file_limit = hardlimit,softlimit
```

For example:

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

group

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

```
group = group name
```

For example:

```
group = my_group_name
```

hold

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

user specifies user hold

system specifies system hold

usersys specifies user and system hold

The syntax is:

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

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

```
hold = user
```

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

initialdir

The pathname of the directory to use as the initial working directory during execution of the job step. If none is specified, the initial directory is the current working directory at the time you submitted the job. Filenames mentioned in the command file which do not begin with a / are relative to the initial directory. If you only specify this keyword in the first job step, its value is inherited to the other job steps.

The syntax is:

```
initialdir = pathname
```

For example:

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

job_cpu_limit

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

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

If you specify a fraction, it is rounded to seconds.

job_name

Specifies the name of the job. This keyword must be specified in the first job step. If it is specified in other job steps in the job command file, it is ignored. You can name the job using any combination of letters and/or numbers.

The syntax is:

```
job_name = job_name
```

For example:

```
job_name = my_first_job
```

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

job_type

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

pvm3 PVM jobs. This is not for PVMe jobs.

parallel For other parallel jobs.

serial For serial jobs.

If not specified, the job will be serial.

The syntax is:

```
job_type = string
```

For example:

```
job_type = pvm3
```

max_processors

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

The syntax is:

```
max_processors = number
```

For example:

```
max_processors = 6
```

min_processors

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

The syntax is:

```
min_processors = number
```

For example:

```
min_processors = 4
```

notification

Specifies when the user specified in **notify_user** is sent mail. If you only specify this keyword in the first job step, its value is inherited to the other job steps. 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 step completes, your notification keyword would be:

```
notification = complete
```

notify_user

Specifies the user to whom mail is sent based on the **notification** keyword. If you only specify this keyword in the first job step, its value is inherited to the other job steps. The default is submitting user and the submitting machine.

The syntax is:

```
notify_user = userid
```

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

```
notify_user = bob
```

output

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

The syntax is:

```
output = filename
```

For example:

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

parallel_path

Used for PVM. Tells the PVM process where to look for the executables when the parallel job spawns tasks. The syntax is:

```
parallel_path = directory
```

For example:

```
parallel_path = /u/tadj/cmds/pvm/$pvm_arch
```

preferences

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

The syntax is:

```
queue
```

requirements

List of requirements which a remote machine must meet to execute any job steps in the pool. If you only specify this keyword in the first job step, its value is inherited to the other job steps. 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 pool. When specifying a feature, be sure to specify the feature in the same way in which the feature is specified in the machine configuration file on the machine that supports the feature.

- Machine

  Specifies the name(s) of machines on which to run the jobs. When specifying a machine, be sure to specify the machine in the same way in which the machine is specified on the machine of interest. To see how it is defined on any machine, use the **llstatus** command or the details function available on the graphical user interface.

  You can also set the machine equal to a job step name. This means that you want the job step to run on the same machine on which the previous job step ran. This option and the previous option are mutually exclusive. The syntax for this is:

  `requirements = (Machine == machine.`*`step_name`*`)`

  where *step_name* is a step name previously defined in the job command file. The use of `Machine == machine.`*`step_name`* is limited to serial jobs.

  If *step_name* equals *stepn*, then *stepn* has to be the sustaining job in the **dependency** statement.

  For example:

  ```
  # @ step_name    = step1
  # @ executable   = c1
  # @ output       = $(executable).$(jobid).$(step_name).out
  # @ queue
  # @ step_name    = step2
  # @ dependency   = ( step1 == 0 )
  # @ requirements = ( Machine == machine.step1)
  # @ executable   = c2
  # @ output       = $(executable).$(jobid).$(step_name).out
  # @ queue
  ```

- Disk

  Specifies the amount of disk space in kilobytes you believe is required in the execute directory on the remote workstation. The location of the execute directory is defined using the EXECUTE keyword.

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

- Adapter

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

The syntax is:

```
Adapter == adapter_name
```

The valid adapter names are:

hps_ip	Refers to the high-performance switch in IP mode.
hps_user	Refers to the high-performance switch in user mode.
ethernet	Refers to Ethernet. This is the default.
fddi	Refers to Fiber Distributed Data Interface (FDDI).
tokenring	Refers to Token Ring.
fcs	Refers to Fiber Channel Standards.

For example:

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

- Pool

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

  The syntax is:

  ```
  Pool == number
  ```

  For example:

  ```
  Pool == 4
  ```

Specify multiple requirements in a single requirements statement.

The syntax is:

```
requirements = Boolean expression
```

For example,

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

restart

Restarts job steps 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.

The syntax is:

```
rss_limit = hardlimit,softlimit
```

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

```
rss_limit=120,100
```

Because no units are specified in the above example, bytes are assumed. See "core_limit" on page 185 for an explanation of the units and values you can specify.

shell

Specifies the name of the shell to use for the job step. If not specified, the shell used in the owner's password file entry is used. If none is specified, the /bin/sh is used. If you only specify this keyword in the first job step, its value is inherited to the other job steps.

The syntax is:

```
shell = name
```

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

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

stack_limit

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

The syntax is:

```
stack_limit = hardlimit,softlimit
```

For example:

```
stack_limit = 120, 100
```

Because no units are specified in the above example, bytes are assumed. See "core_limit" on page 185 for an explanation of the units and values you can specify.

startdate

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

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
```

step_name

Specifies the name of the job step. You can name the job step using any combination of letters, numbers, underscores (_) and periods (.). You cannot, however, name it T or F, or use a number in the first position of the step name. If you don't specify step names, by default, the first job step is named the character string "0", the second is named the character string "1", and so on.

The syntax is:

```
step_name = step_name
```

For example:

```
step_name = step_3
```

user_priority

Sets the initial priority of your job step. Priority only affects your job steps. It orders job steps you submitted with respect to other job steps submitted by you, not with respect to job steps submitted by other users. If you only specify this keyword in the first job step, its value is inherited to the other job steps.

The syntax is:

```
user_priority = number
```

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

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

Job Command File Macros

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

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

$(*macro_name*)

The following macros are available to you:

$(host) $(hostname)
: The hostname of the machine from which the job was submitted.

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

$(jobid) $(cluster)
: The sequential number assigned to this job by the submitting machine.

$(stepid) $(process)
: The sequential number assigned to this job step when multiple queue statements are used in the job command file.

Example 1: The following job command file creates an output file called **stance.78.out**, where `stance` is the host and `78` is the jobid.

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

In addition, the following keywords are also available as macros. However, you must define them in the job command file. These keywords are described in detail in "LoadLeveler Keywords" on page 183.

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

Note that for the **$(comment)** macro, the keyword definition must be a single string with no blanks.

Example 2: The following job command file creates an output file called **computel.step1.March05**.

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

Appendix C. Sample Files

This appendix contains sample configuration and administration files. These sample files are located in the samples directory called **/usr/lpp/LoadL/nfs/samples**.

Sample Configuration Files

Sample 1

You can find the following configuration file in the samples directory.

```
#       Machine Description
#
ARCH = R6000
#
#  Specify LoadLeveler Administrators here:
#
LOADL_ADMIN = loadl
#
#       Default to starting LoadLeveler daemons when requested
#
START_DAEMONS = TRUE
#
# Machine authentication
#
# If TRUE, only connections from machines in the ADMIN_LIST are accepted.
# If FALSE, connections from any machine are accepted.  Default if not
# specified is FALSE.
#
MACHINE_AUTHENTICATE = FALSE
#
#       Specify which daemons run on each node
#
SCHEDD_RUNS_HERE        =       True
STARTD_RUNS_HERE        =       True
#
# Specify information for backup central manager
#
# CENTRAL_MANAGER_HEARTBEAT_INTERVAL = 300
# CENTRAL_MANAGER_TIMEOUT = 6
#
# Specify pathnames
#
RELEASEDIR      = $(tilde)
LOCAL_CONFIG    = $(tilde)/LoadL_config.local
ADMIN_FILE      = $(tilde)/LoadL_admin
LOG             = $(tilde)/log
SPOOL           = $(tilde)/spool
EXECUTE         = $(tilde)/execute
HISTORY         = $(SPOOL)/history
BIN             = $(RELEASEDIR)/bin
LIB             = $(RELEASEDIR)/lib
```

© Copyright IBM Corp. 1993, 1995

```
# Specify port numbers
#
COLLECTOR_STREAM_PORT  = 9612
MASTER_STREAM_PORT     = 9616
NEGOTIATOR_STREAM_PORT = 9614
SCHEDD_STREAM_PORT     = 9605
STARTD_STREAM_PORT     = 9611
COLLECTOR_DGRAM_PORT   = 9613
STARTD_DGRAM_PORT      = 9615
MASTER_DGRAM_PORT      = 9617
#
# Specify accounting controls
#
ACCT                    = A_OFF
ACCT_VALIDATION         = $(BIN)/llacctval
GLOBAL_HISTORY          = $(SPOOL)
#
# Specify prolog and epilog path names
#
# JOB_PROLOG =
# JOB_EPILOG =
#
# Specify the default job class
#
DEFAULT_JOB_CLASS = No_Class
#
# Specify checkpointing intervals
#
MIN_CKPT_INTERVAL    = 900
MAX_CKPT_INTERVAL    = 7200
#       LoadL_KeyboardD Macros
#
KBDD                = $(BIN)/LoadL_kbdd
KBDD_LOG            = $(LOG)/KbdLog
MAX_KBDD_LOG        = 64000
KBDD_DEBUG          =
#
# Specify whether to start the keyboard daemon
#
X_RUNS_HERE          = True
```

```
# Specify whether to use X server XGetIdleTime() protocol extension
#
USE_X_IDLE_EXTENSION = False
#
#               LoadL_StartD Macros
#
STARTD                  = $(BIN)/LoadL_startd
STARTD_LOG              = $(LOG)/StartLog
MAX_STARTD_LOG          = 64000
STARTD_DEBUG            =
POLLING_FREQUENCY       = 5
POLLS_PER_UPDATE        = 24
#
#               LoadL_SchedD Macros
#
SCHEDD                  = $(BIN)/LoadL_schedd
SCHEDD_LOG              = $(LOG)/SchedLog
MAX_SCHEDD_LOG          = 64000
SCHEDD_DEBUG            =
SCHEDD_INTERVAL         = 120
MAX_JOBS_RUNNING        = 15
#
# Collector Macros
#
COLLECTOR               = $(BIN)/LoadL_collector
COLLECTOR_DEBUG         =
COLLECTOR_LOG           = $(LOG)/CollectorLog
MAX_COLLECTOR_LOG       = 64000
CLIENT_TIMEOUT          = 30
#
# Negotiator Macros
#
NEGOTIATOR              = $(BIN)/LoadL_negotiator
NEGOTIATOR_DEBUG        =
NEGOTIATOR_LOG          = $(LOG)/NegotiatorLog
MAX_NEGOTIATOR_LOG      = 64000
NEGOTIATOR_INTERVAL     = 60
MACHINE_UPDATE_INTERVAL = 300
#
# Starter Macros
#
STARTER                 = $(BIN)/LoadL_starter
STARTER_DEBUG           =
STARTER_LOG             = $(LOG)/StarterLog
MAX_STARTER_LOG         = 64000
```

```
# LoadL_Master Macros
#
MASTER                  = $(BIN)/LoadL_master
MASTER_LOG              = $(LOG)/MasterLog
MASTER_DEBUG            =
MAX_MASTER_LOG          = 64000
RESTARTS_PER_HOUR       = 12
PUBLISH_OBITUARIES      = TRUE
OBITUARY_LOG_LENGTH     = 25
#
# Specify whether log files are truncated
# when opened
TRUNC_MASTER_LOG_ON_OPEN     = False
TRUNC_STARTD_LOG_ON_OPEN     = False
TRUNC_SCHEDD_LOG_ON_OPEN     = False
TRUNC_KBDD_LOG_ON_OPEN       = False
TRUNC_STARTER_LOG_ON_OPEN    = False
TRUNC_COLLECTOR_LOG_ON_OPEN  = False
TRUNC_NEGOTIATOR_LOG_ON_OPEN = False
#       NQS Directory
#
# For users of NQS resources:
# Specify the directory containing qsub, qstat, qdel
#
# NQS_DIR               = /usr/bin
#
# Specify ISS metric keywords
#
# CUSTOM_METRIC =
# CUSTOM_METRIC_COMMAND =
#
#       Machine control expressions and macros
#
OpSys   : "$(OPSYS)"
Arch    : "$(ARCH)"
Machine : "$(HOST).$(DOMAIN)"
#
#       Expressions used to control starting and stopping of foreign jobs
#
MINUTE          = 60
HOUR            = (60 * $(MINUTE))
StateTimer      = (CurrentTime - EnteredCurrentState)
BackgroundLoad          = 0.7
HighLoad                = 1.5
StartIdleTime           = 15 * $(MINUTE)
ContinueIdleTime        = 5 * $(MINUTE)
MaxSuspendTime          = 10 * $(MINUTE)
MaxVacateTime           = 10 * $(MINUTE)
```

```
KeyboardBusy    = KeyboardIdle < $(POLLING_FREQUENCY)
CPU_Idle        = LoadAvg <= $(BackgroundLoad)
CPU_Busy        = LoadAvg >= $(HighLoad)
#
#  Refer to LoadL_config man page for an explanation of these control
#  expressions
#
# START         : $(CPU_Idle) && KeyboardIdle > $(StartIdleTime)
# SUSPEND       : $(CPU_Busy) || $(KeyboardBusy)
# CONTINUE      : $(CPU_Idle) && KeyboardIdle > $(ContinueIdleTime)
# VACATE        : $(StateTimer) > $(MaxSuspendTime)
# KILL          : $(StateTimer) > $(MaxVacateTime)

START           : T
SUSPEND         : F
CONTINUE        : T
VACATE          : F
KILL            : F
#
# Expressions used to prioritize job queue
#
# Values which can be part of the SYSPRIO expression are:
#
#    QDate           Job submission time
#    UserPrio        User priority
#    UserSysprio     System priority value based on userid (from the user
#                    list file with default of 0)
#    ClassSysprio    System priority value based on job class (from the
#                    class list file with default of 0)
#
#       The following expression is an example.
#
#SYSPRIO  : (ClassSysprio * 100) + (UserSysprio * 10) - (QDate)
#
# The following (default) expression for SYSPRIO creates a FIFO
# job queue.
#
SYSPRIO   : 0 - (QDate)
```

Sample 2

The following sample illustrates another variation of a configuration file.

```
#     Specify the Machine Description
ARCH = R6000     # The machine is a RISC System/6000
#     Specify LoadLeveler Administrators
LOADL_ADMIN = loadl root
#     Default to starting LoadLeveler daemons when requested
START_DAEMONS = TRUE
#     Machine Authentication
# If TRUE, only connections from machines in the ADMIN_LIST are
# accepted.  If FALSE, connections from any machine are accepted.
# Default if not specified is FALSE.
MACHINE_AUTHENTICATE = FALSE
#     Specify which daemons run on each node
SCHEDD_RUNS_HERE = True
STARTD_RUNS_HERE = True
#     Pathnames
RELEASEDIR    = /u/loadl
NODEDIR       = /LL
LOCAL_CONFIG  = $(tilde)/$(HOST)/LoadL_config.local
ADMIN_FILE    = $(tilde)/LoadL_admin
LOG           = $(NODEDIR)/log
SPOOL         = $(NODEDIR)/spool
EXECUTE       = $(NODEDIR)/execute
HISTORY       = $(SPOOL)/history
BIN           = $(RELEASEDIR)/bin
LIB           = $(RELEASEDIR)/lib
DEFAULT_JOB_CLASS = No_Class
#     LoadL_KeyboardD Macros
KBDD           = $(BIN)/LoadL_kbdd
KBDD_LOG       = $(LOG)/KbdLog
MAX_KBDD_LOG   = 64000
KBDD_DEBUG     =
# Specify whether to start the keyboard daemon
X_RUNS_HERE    = False
# Specify whether to se X server XGetIdleTime protocol extension
USE_X_IDLE_EXTENSION = False
```

```
# LoadL_StartD Macros
STARTD
STARTD_LOG         = $(BIN)/LoadL_startd
MAX_STARTD_LOG     = $(LOG)/StartLog
STARTD_DEBUG       = D_FULLDEBUG D_LOAD D_JOB D_MACHINE
POLLING_FREQUENCY  = 5
POLLS_PER_UPDATE   = 24
# LoadL_SchedD Macros
SCHEDD           = $(BIN)/LoadL_schedd
SCHEDD_LOG       = $(LOG)/SchedLog
MAX_SCHEDD_LOG   = 100000
SCHEDD_DEBUG     = D_FULLDEBUG D_SCHEDD
SCHEDD_INTERVAL  = 120
MAX_JOBS_RUNNING = 200
# Collector Macros
COLLECTOR         = $(BIN)/LoadL_collector
COLLECTOR_DEBUG   =
COLLECTOR_LOG     = $(LOG)/CollectorLog
MAX_COLLECTOR_LOG = 64000
CLIENT_TIMEOUT    = 30
# Negotiator Macros
NEGOTIATOR              = $(BIN)/LoadL_negotiator
NEGOTIATOR_DEBUG
NEGOTIATOR_LOG          = $(LOG)/NegotiatorLog
MAX_NEGOTIATOR_LOG      = 64000
NEGOTIATOR_INTERVAL     = 60
MACHINE_UPDATE_INTERVAL = 300
# Starter Macros
STARTER                 = $(BIN)/LoadL_starter
STARTER_DEBUG           = D_NODATE D_FULLDEBUG
MIN_CKPT_INTERVAL       = 900
MAX_CKPT_INTERVAL       = 7200
```

```
# LoadL_Master Macros
MASTER                   = $(BIN)/LoadL_master
MASTER_LOG               = $(LOG)/MasterLog
TRUNC_MASTER_LOG_ON_OPEN = False
MASTER_DEBUG             =
MAX_MASTER_LOG           = 64000
RESTARTS_PER_HOUR        = 60
PUBLISH_OBITUARIES       = True
OBITUARY_LOG_LENGTH      = 25
# Machine Control Expressions and Macros
OpSys:"$(OPSYS)"
ARCH:"$(ARCH)"
# Expressions used to control starting and stopping of foreign jobs
MINUTE           = 60
HOUR             = (60 * $(MINUTE))
StateTimer       = (CurrentTime - EnteredCurrentState)
BackgroundLoad   = 0.7
HighLoad         = 1.5
StartIdleTime    = 15 * $(MINUTE)
ContinueIdleTime = 5 * $(MINUTE)
MaxSuspendTime   = 10 * $(MINUTE)
MaxVacateTime    = 10 * $(MINUTE)
KeyboardBusy     = KeybardIdle < $(POLLING_FREQUENCY)
CPU_Idle         = LoadAvg <= $(BackgroundLoad)
CPU_Busy         = LoadAvg >= $(HighLoad)
#         Control Expressions
START     : $(CPU_Idle) && KeyboardIdle > $(StartIdleTime)
SUSPEND   : $(CPU_Busy) || $(KeyboardBusy)
CONTINUE  : $(CPU_Idle) && KeyboardIdle > $(ContinueIdleTime)
VACATE    : $(StateTimer) > $(MaxSuspendTime)
KILL      : $(StateTimer) > $(MaxVacateTime)
# Expressions used to prioritize job queues
# Values which can be part of the SYSPRIO expression are:
# QDate           - job submission time
# UserPrio        - User priority
# UserSysprio     - System priority value based on userid
# ClassSysprio    - System priority value based on job class
# GroupSysprio    - System priority value based on job group
# The following expression on SYSPRIO creates a FIFO job queue
SYSPRIO: 0 -(QDate)
```

Administration Files

Sample 1

You can find the following administration file in the samples directory.

```
# ******************************************************************
# LoadL_admin file: Remove comments and edit this file to suit your
# site.  This file consists of machine, class, and user stanzas.
# Each stanza has defaults, as specified in a "defaults:" section.
# Default stanzas are used to set specifications for fields which
# are not filled in.  Class, user, and group stanzas are
# optional.  Refer to LoadL_admin man page and the LoadLeveler
# Administration Guide for detailed information about keywords
# and associated values.  Also see LoadL_admin.1 in the
#  loadl/samples directory for sample stanzas.
####################################################################
# DEFAULTS FOR MACHINE, CLASS, USER, AND GROUP STANZAS:
# Remove initial # (comment), and edit to suit.
#
default:   type = machine
#          central_manager = false  # default not central manager
#          schedd_host = false      # default not a public scheduler
#          submit_only = false      # default not a submit-only mach.
#          speed = 1                # default machine speed

default:   type = class             # default class stanza
#          priority = 0             # default ClassSysprio
#          max_processors = -1      # default max processors for
                                    # class (no limit).

default:   type = user              # default user stanza
#          priority = 0             # default UserSysprio
           default_class = No_Class # default class = No_Class (not
                                    # optional)
           default_group = No_Group # default group = No_Group (not
                                    # optional)
#          maxjobs = -1             # default maximum jobs user is
                                    # allowed to run simultaneously
                                    # (no limit)
#         maxqueued = -1        # default maximum jobs user is
                                # (no limit).  Does not limit
                                # jobs submitted.

default:  type = group          # default group stanza
#         priority = 0          # default GroupSysprio
#         maxjobs = -1          # default maximum jobs group
                                # is allowed to run simultaneously
                                # (no limit)
#         maxqueued = -1        # default maximum jobs group is
                                # allowed on system queue
                                # (no limit).  does not
                                # limit jobs submitted.
```

```
########################################################################
# MACHINE STANZAS:
# These are the machine stanzas; the first machine is defined as
# the central manager.  mach1:, mach2:, etc. are machine name labels -
# revise these placeholder labels with the names of the machines in the
# pool, and specify any schedd_host and submit_only keywords and values
# (true or false), if required.
########################################################################
serv1:    type = machine
               central_manager = true
#
# mach1:        type = machine
#               schedd_host = true
#
# mach2:        type = machine
#               schedd_host = true
#
# mach3:        type = machine
#               schedd_host = true
#
# mach4:        type = machine
#
# mach5:        type = machine
#
# mach6:        type = machine
#
# mach7:        type = machine
#               submit_only = true
#
# mach8:        type = machine
#               submit_only = true
#
# etc:          type = machine
########################################################################
# CLASS STANZAS: (optional)
# These are sample class stanzas; small, medium, large, and nqs.
# labels for job classes - revise these labels and specify attributes
# to each class.
########################################################################
#small:    type = class             # class for small jobs
#          priority = 100           # ClassSysprio
#          include_users=<userlist> # only these users can submit
#                                        # jobs of this class
#
#          exclude_users=<userlist> # only these cannot submit
#          cpu_limit = 00:15:00           # 15 minute run time limit
#          data_limit = <size>
#          core_limit = <size>
#          file_limit = <size>
#          stack_limit = <size>
#          rss_limit = <size>
#
#medium:   type = class             # class for medium jobs
#          priority = 60            # ClassSysprio
#          cpu_limit= 02:00:00      # 2 hour run time limit
#
```

```
#large:    type = class             # class for large jobs
#          priority = 20            # ClassSysprio
#          cpu_limit = 24:00:00     # 24 hour run time limit
#nqs:           type = class             # class for nqs jobs
#              NQS_class = true         # will be routed to NQS
#              NQS_submit = nqs_pipe_q_name    # name of pipe queue
#              NQS_query = q_name1@host1 q_name2@host2... #queue names
#########################################################################
# GROUP STANZAS: (optional)
# These are sample group stanzas; group1, group2 are sample labels
# for groups - revise these labels and specify attributes to each group.
#########################################################################
#group1:        type = group
#               priority = 80
#               maxjobs = 40
#               maxqueued = 80
#
#group2:        type = group
#               priority = 50
#               maxjobs = 20
#               maxqueued = 40
#########################################################################
# USER STANZAS: (optional, default user stanza not optional)
# These are sample user stanzas; user1, user2, user3 are sample labels
# for users - revise these labels and specify attributes to each user.
#########################################################################
# user1:        type = user
#               priority = 80
#               default_class = small
#               default_group = group1
#               maxjobs = 20
#               maxqueued = 40
#
# user2:        type = user
#               priority = 50
#               default_class = medium
#               default_group = group2
#               maxjobs = 10
#               maxqueued = 20
#
# user3:        type = user
#               priority = 10
#               maxjobs = 5
#               maxqueued = 10
#########################################################################
```

Sample 2

The following is another variation of an administration file:

```
# Machine Stanzas
yugo:     type            = machine
          central_manager = true   # this machine is the
                                   # central manager
earth:    type            = machine
          central_manager = false  # this machine is not the
                                   # central manager
          schedd_host     = false  # not a scheduling machine
          submit_only     = true   # a submit-only machine

mars:     type            = machine
          central_manager = false
          schedd_host     = false
          alias           = machinea # the alias is machinea
# User Stanzas
fred:   type = user
        priority = 50
        default_class = No_class
        maxjobs = 20
#
wilma:  type = user
        priority = 50
        default_class = No_class
        maxjobs = 10
        account = dept10 user3 user4
# Class Stanzas
small:  type = class               # class for small jobs
        priority = 50
        include_users=susann holly # authorized users
        cpu_limit=00:15:00         # 15 minute run time limit
        data_limit=30mb
        core_limit=10mb
        file_limit=50mb
        stack_limit=10mb
        rss_limit=35mb

nqs:    type          = class      # This class is for nqs jobs
        NQS_class     = true
        NQS_submit    = pipe_queue_name
        NQS_query     = one two three # query names

# Group Stanzas
dept_a:   type          = group
          priority      = 80
          include_users = susann holly
dept_b:   type          = group
          priority      = 30
          include_users = bob dave
```

Sample Illustrating How the Configuration File and the Administration File Work Together

```
# This configuration file defines classes.
# Cjour (day class): cpu_limit=30,
#       data_limit=unlimited.
# Dnuit (night class): cpu_limit=16,
#       data_limit=64mb.
# Fnuit (night class): cpu_limit=16,
#       data_limit=unlimited.
# Fnuit suspended during the day.
# Dnuit is suspended if LoadAverage > 3
# and the job is continued if the LoadAverage <= 1.5
START_DAEMONS = TRUE
X_RUNS_HERE = FALSE
# Definition of classes
Class = { "Cjour" "Dnuit" "Fnuit" }
# Define the maximum number of LoadLeveler
# jobs that will run on this machine.
MAX_STARTERS = 3
HighLoadStD = 1.5  # HighLoad Start for Fnuit
HighLoadSuD = 3
MIN_CKPT_INTERVAL = 900     # Specify how often
MAX_CKPT_INTERVAL = 7200    # checkpointing occurs
# Define the macros Jour and Nuit
Nuit = ((tm_hour >= 17) || (tm_hour <=9) || \
       (tm_wday == 6) || (tm_wday == 0))
Jour = ((tm_wday > 0) && (tm_wday < 6) \
       && (tm_hour >=9) && \
       (tm_hour <=17))
# A Cjour job will start during
# the day.  A Dnuit job will start during
# the night.  A Fnuit job will start
# during the night if the load average
# is less than or equal to 1.5.
#
START: (((Class == "Cjour") && $(Jour) || \
       ((Class == "Dnuit") && $(Nuit) || \
       ((Class == "Fnuit") && $(Nuit) && \
       (LoadAvg <=$(HighLoadStD))))
# The Dnuit job will get suspended
# during the day if the LoadAvg is
# greater than HighLoadSuD.  The Fnuit job
# will be suspended during the day.
SUSPEND: (((Class == "Dnuit") && $(Jour) && \
       ((LoadAvg >= $(HighLoadSuD))) || \
       ((Class == "Fnuit") && $(Jour)))
# The Dnuit job may continue during the day if
# the LoadAvg is less than HighLoadStD.
CONTINUE: (((Class == "Dnuit") && $(Jour) && \
       ((LoadAvg <= $(HighLoadStD))) || \
       ((Class == "Fnuit") && $(Nuit)))
VACATE: F
KILL: F
# The priorities of the classes
SYSPRIO: 0-(Qdate) + (ClassSysprio) * 10)
```

```
   Administration File
   default: type=machine
            central_manager=false
            schedd_host=true
   default: type=class
            priority=50
            cpu_limit=unlimited
            data_limit=unlimited
default: type=user
         priority=50
            default_class=A_jour
            max_jobs=2
   # machine stanzas
   fturbo1: type=machine
            central_manager=true
   fturbo2: type=machine
   # class stanzas
   Cjour: type=class
          priority=50
          cpu_limit=00:30:00
   #
   Dnuit: type=class
    priority=50
          cpu_limit=16:00:00
    data_limit=64mb
Fnuit: type=class
       priority=50
         cpu_limit=16:00:00
   # user stanzas
   user1: type=user
          priority=50
          default_class=No_class
          max_jobs=20
   user2: type=user
          priority=50
          default_class=No_class
      max_jobs=10
```

Appendix D. LoadLeveler Daemons and Processes

"How LoadLeveler Works" on page 2 briefly describes the LoadLeveler daemons and processes. "The LoadLeveler Job Cycle" on page 9 describes how these daemons and processes work together in step-by-step graphical format. This appendix provides a more detailed explanation of the daemons and processes.

The Master Daemon

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

This daemon starts and if necessary, restarts all the LoadLeveler daemons that the machine it resides on is configured to run. As part of its startup procedure, this daemon executes the **.llrc** file (a dummy file is provided under the release /**bin** directory). You can use this script to customize your local configuration file, specifying what particular data is stored locally. The **master** also runs the **schedd** and **startd** daemons on all machines in the pool. For all machines that don't already monitor X activity, this daemon also runs the **kbdd** daemon when **X_RUNS_HERE** is set to **true**.

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

On the machine designated as the central manager, the **master** runs the **collector** and **negotiator** daemons. The **master** determines whether to run the **collector** and **negotiator** by checking its own machine name against a variable in the administration file. In addition, this daemon controls the central manager backup function. The master daemons, running on primary or alternate central managers keep in touch with each other. If a central manager failure is detected, one of the alternate central managers becomes the primary central manager by starting the **negotiator** and **collector** daemons.

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

When a daemon must be restarted, the **master** sends mail to the administrator(s) identified by the **LOADL_ADMIN** keyword in the configuration file. (The volume of mail sent to the administrator is controlled through the debug flag statement in the configuration file. See "Controlling Debugging Output" on page 54 for more information.) The mail contains the name of the failing daemon, its termination status, and the daemon's most recent log file. If **master** aborts after exceeding **RESTARTS_PER_HOUR**, it will also send that mail before exiting.

The **master** daemon may perform the following actions:

- Kill all daemons and exit

© Copyright IBM Corp. 1993, 1995

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

The Schedd Daemon

The **schedd** daemon receives jobs sent by the **llsubmit** command and schedules those jobs to machines selected by the **negotiator** daemon. The **schedd** daemon is started, restarted, signalled, and stopped by the **master** daemon. For submit-only machines, the **schedd** gets the executable from the submit-only host.

The **schedd** daemon performs the following functions:

- Assigns new job ids when requested by the **llsubmit** command.
- Receives new jobs from the **llsubmit** command. A new job is received as a series of descriptors (*procs*) for each job step in the cluster. A null proc indicates the end of the series. The **schedd** forwards procs to the **negotiator** daemon as soon as they are received from the submit command.
- Maintains on disk copies of jobs submitted locally (on this machine) that are either waiting or running on a remote (different) machine. The central manager can use this information to reconstruct the job information.
- Responds to directives sent by the administrator through the **negotiator** daemon. The directives include:
  - Run a job. The **schedd** may indicate to the **negotiator** daemon that the job is marked as running, or it may reject the job because the job was previously rejected by the **startd**.
  - Change the priority of a job.
  - Change job flags.
  - Remove a job.
  - Hold a job.
  - Send information about all jobs.
  - Kill a running job.
- Sends job events to the **negotiator** daemon when:
  - **schedd** is restarting.
  - A new job cluster is arriving. A job cluster is sent as a series of procs terminated by a null proc.
  - A job is started.
  - A job has been rejected or has completed or vacated. **schedd** determines the status by examining the exit status returned by the **startd**.
- Requests that a remote **LoadL_startd** kill a job.
- Sends statistics to the **LoadL_collector**. These statistics may include:
  - Running jobs submitted from this machine.
  - Idle or waiting jobs submitted from this machine.
  - Users who have submitted jobs to this machine.

- Handles the checkpoint file associated with the job.

The Startd Daemon

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

The **startd** performs these functions:

- Runs a timeout procedure that includes building a snapshot of the state of the machine that includes static and dynamic data. This timeout procedure is run at the following times:
  - After a job completes.
  - After an X event occurs.
  - According to the definition of the **POLLING_FREQUENCY** keyword in the configuration file.
- Determines the following information and sends it to the **collector**:

  State State of the **startd** daemon. The state can be one of the following:
  - Busy
  - Idle
  - Running
  - Suspended
  - Vacating
  - Killed.

  EnteredCurrentState
  UNIX date when current state was entered.

  Memory Physical memory installed on the machine in megabytes.

  Disk Free disk in megabytes on the file system where remote checkpoints are stored.

  KeyboardIdle Time in seconds since last activity on any remote or local keyboard or mouse.

  Cpus Number of CPU's installed (1 unless on a multi-processor).

  LoadAvg A measure of the CPU load on the system. The load average is the average of the number of processes ready to run or waiting for disk I/O to complete. LoadLeveler determines this average every 60 seconds. The load average does not map to CPU time.

  Machine Name of the current machine.
- Receives job requests from the **schedd** daemon to:
  - Start a job
  - Suspend a job
  - Continue a job
  - Kill a job.

When the **schedd** daemon tells the **startd** to start a job, the **startd** does the following:

Determines whether its own state permits a new job to run	
If:	**Then this happens:**
Yes, it can start a new job	The **startd** forks a **starter** process.
No, it cannot start a new job	The **startd** rejects the request because either: • Jobs have been suspended, flushed, or drained • The job limit set for the **MAX_STARTERS** keyword has been reached • There are not enough slots available for the designated job class • In the case of a parallel job, if it cannot allocate the number of parallel machines needed to run the parallel job.

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

  For each request, **startd** marks its own new state, forwards its new state to the **collector** daemon, and then performs the appropriate action for any jobs that are active.
- Receive notification of X events (from the **kbdd** daemon) on all machines except Sun machines.
- Periodically examines the process table for LoadLeveler jobs and accumulates resources consumed by those jobs. This resource data is used to determine if a job has exceeded its job limit and for recording in the history file.

The Starter Process

The **startd** daemon spawns a **starter** process after the **schedd** daemon tells the **startd** to start a job. The **starter** process is responsible for running the job and reporting status back to **startd**.

The **starter** process performs these functions:

- Proceses the prolog and epilog programs as defined by the **JOB_PROLOG** and **JOB_EPILOG** keywords in the configuration file. The job will not run if the prolog program exits with a return code other than zero.
- Handles authentication. This includes:
  - Authenticates AFS, if necessary
  - Verifies that the submitting user is *not* root
  - Verifies that the submitting user has access to the appropriate directories in the local file system.

- Runs the job by forking a child process that runs with the user id and all groups of the submitting user. The **starter** child creates a new process group of which it is the process group leader, and executes the user's program or a shell. The **starter** parent is responsible for detecting the termination of the **starter** child. This means that if the parent process forks a child and once the parent process ends, LoadLeveler assumes the job has ended. LoadLeveler does not monitor the children of the parent.
- Responds to vacate and suspend orders from the **startd**.
- Periodically generates a new checkpoint file.

The Collector Daemon

The **collector** daemon maintains status of each machine in the cluster and responds to queries from the **llstatus** command and from the **negotiator** daemon. The **collector** and the **negotiator** both run on a single machine in the cluster and comprise the central manager. The **collector** is started, restarted, signalled, and stopped by the **master** daemon.

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

1. Which **schedd** daemons are running and compiles statistics on the jobs they submit
2. Which **startd** daemons are running and the status of each **startd** machine.

If the **collector** does not receive an update from any machine within the time period defined by the **MACHINE_UPDATE_INTERVAL** keyword, it assumes that machine is down.

The Kbdd Daemon

The **kbdd** daemon monitors X activity on all machines in the LoadLeveler pool except for those machines that have other ways of monitoring X activity. The **kbdd** daemon reads and is dependent on the structure of /**dev**/**kmem**, the kernal's virtual memory and I/O address space, to determine if there has been X activity. The **kbdd** is spawned by the **master** daemon if the **X_RUNS_HERE** keyword in the configuration file is set to **true**.

The **kbdd** daemon notifies the **startd** daemon when it detects X events, however, the **kbdd** is *not* interrupt driven. It sleeps for the number of seconds defined by the **POLLING_FREQUENCY** keyword and then determines if X events, in the form of mouse or keyboard activity, have occurred.

The Negotiator Daemon

The **negotiator** tracks job status in the cluster and decides where jobs run. It responds to queries about job status from the **llq** command, as well as **llprio**, **llhold**, and **llcancel** requests. The **collector** and **negotiator** daemons both run on a single machine in the cluster and comprise the central manager. The **negotiator** is started, restarted, signalled, and stopped by the **master** daemon.

The following table lists and describes the tables that the **negotiator** maintains in its memory:

Table	Description
Machine status	All machines on which **startd** is running, ordered by load average.
User status	all users who submitted jobs.
Job queues	all jobs by current state: waiting, running, pending, completed, and holding.

The **negotiator** performs the following functions:

- Receives and records job status changes from the **schedd** daemon.
- Schedules jobs based on a variety of scheduling criteria and policy options. Once a job is selected, the **negotiator** contacts the **schedd** that originally created the job.
- Handles requests to:
  - Set priorities
  - Query about jobs
  - Remove a job
  - Hold or release a job
  - Favor or unfavor a user or a job.
- Receives notification of **schedd** resets indicating that a **schedd** has restarted.
- Terminates if an inconsistency appears between job status as reported from individual nodes and the **negotiator** daemon's resident status.

Appendix E. LoadLeveler's Application Programming Interfaces

LoadLeveler provides several Application Programming Interfaces (API) that you can use. LoadLeveler's APIs are interfaces that allows application programs written by customers to use specific data or functions that are a part of LoadLeveler. This appendix lists and describes these APIs.

Warning: These APIs are not *thread safe*; that is, they should not be linked to a threaded application.

Accounting APIs

LoadLeveler's job accounting function has two APIs. One is for account validation and the other is for extracting accounting data.

Account Validation API

LoadLeveler provides a program called **llacctval** which performs account validation. Your installation may choose to replace this routine with your own accounting routine by specifying the name of the program in the **ACCT_VALIDATION** keyword in the configuration file. The program is invoked from the **llsubmit** command.

Invoked as:

```
program user_name user_group user_acct# acct1 acct2 ...
```

where:

program
is the name of the program to perform account validation which is the value specified in the configuration file in the **ACCT_VALIDATION** keyword.

user_name
is the name of the user to validate.

user_group
is the login group name of the user to validate.

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

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

If the validation succeeds, the exit status must be zero. If it does not succeed, the exit status must be a non-zero number.

© Copyright IBM Corp. 1993, 1995

Report Generation API

Programmers may write C programs to process records in either a local or the global history file. These C programs can call the function **GetHistory** which will obtain an accounting record from the history file and invoke a caller supplied routine, passing the routine the address of an **LL_job** structure. This structure defines a history file record. The **GetHistory** function reads every record of the file, specified by the argument called **file**. Each record is read into an **LL_job** structure and passed to the function passed as the second argument.

An include file called **llapi.h** and a LoadLeveler library called **libllapi.a** are used to compile programs which call **GetHistory** and link them with **GetHistory**. **libllapi.a** can be found in the **lib** subdirectory and **llapi.h** can be found in the **include** subdirectory of the release directory.

Syntax for GetHistory

```
#include <stdio.h>
#include <sys/resource.h>
#include <sys/types.h>
#include <sys/time.h>
#include "llapi.h"

void
GetHistory(file, function, version)
char *file;
int function();
int version;
```

where:

file
Name of the history file to process.

function
Points to the function to call to process each history record. The syntax to specify this function is:

```
#include <stdio.h>
#include "llapi.h"

void
function(job)
LL_job *job;
```

where

job
is a pointer to the **LL_job** structure representing a record in the history file.

version
is an integer indicating the version of the history file being used. This argument should be set to **LL_JOB_VERSION** which is defined in the **llapi.h** include file.

Sample Accounting API

A sample program called **llpjob** is provided in the **samples**/**llphist** subdirectory of the release directory. **llpjob** invokes **GetHistory** to print every record in the history file. In order to compile **llpjob**, you need to update the sample **Makefile** to reflect your release directory if not /**usr**/**lpp**/**LoadL**/**nfs**. The syntax for running **llpjob** is:

```
llpjob file
```

where `file` is a local or global history file.

Backward Compatibility

If you had written a program to interface with the LoadLeveler 1.1.3 **GetHistory** function, you may continue to use that program with minor modifications.

To allow your program to process LoadLeveler Release 1.2.1 history files, you need to modify the calls to **GetHistory** to include the new version argument and recompile specifying the 1.2.1 libraries.

The **GetHistory** function may be called to retrieve history records in the same format that is used in LoadLeveler release 1.1.3. To use **GetHistory** in this way, you need to specify a version value of 113 instead of LL_JOB_VERSION when invoking it. When invoked in this way, **GetHistory** obtains an accounting record from the history file and invokes the caller supplied routine, passing the routine the address of the **llacct** structure. This structure defines the old format of a history file record.

The same C libraries are required to compile and link programs using **GetHistory** in this way. They are located in the **lib** subdirectory of the release directory.

The include header **llacct.h** defines the old history file format and is located in the include subdirectory of the release directory.

Syntax for GetHistory for Release 1.1.3

```
#include <stdio.h>
#include <sys/resource.h>
#include <sys/types.h>
#include <sys/time.h>
#include "llacct.h"
void
GetHistory (file, function, version)
char *file;
int function();
int version;
```

where:

file
Name of the history file to process.

function
Points to the function to call to process each history record. The syntax to specify this function is:

```
#include <stdio.h>
#include "llacct.h"

void
function(job)
LL_ACCT *job;
```

where

job
is a pointer to the **LL_ACCT** structure representing a record in the history file.

version
is an integer indicating the version of the history file being used. This argument should be set to 113.

Prolog and Epilog APIs

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

Prolog and epilog programs fall into two categories: those that run as the LoadLeveler user ID, and those that run in a user's environment.

To specify prolog and epilog programs, specify the following keywords in the configuration file:

JOB_PROLOG = *pathname*
where *pathname* is the full path name of the prolog program. This program runs under the LoadLeveler user ID.

JOB_EPILOG = *pathname*
where *pathname* is the full path name of the epilog program. This program runs under the LoadLeveler user ID.

JOB_USER_PROLOG = *pathname*
where *pathname* is the full path name of the user prolog program. This program runs under the user's environment.

JOB_USER_EPILOG = *pathname*
where *pathname* is the full path name of the user epilog program. This program runs under the user's environment.

A user environment prolog or epilog runs with AFS authentification (if AFS is installed). You must configure these programs on the machines where the job runs *and* on the machine that schedules the job. If you do not define a value for these keywords, the user enviroment prolog and epilog settings on the executing machine are ignored.

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

env *id* **=** *value*
where

id Is the name of the environment variable

value Is the value (setting) of the environment variable

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

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

Prolog Programs

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

Invoked as:

```
prolog_name
```

where:

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

No arguments are passed to the program but the following environment variables are set:

LOADL_ACTIVE	LoadLeveler version
LOADL_STEP_ID	Job step id
LOADL_STEP_OWNER	Job step owner
LOADL_STEP_COMMAND	Command
LOADL_STEP_ARGS	Arguments
LOADL_STEP_IN	File for stdin
LOADL_STEP_OUT	File for stdout
LOADL_STEP_ERR	File for stderr
LOADL_STEP_INITDIR	Initial working directory
LOADL_STEP_CLASS	Class
LOADL_STEP_ACCT	Account number
LOADL_STEP_GROUP	Group name
LOADL_STEP_TYPE	Type (SERIAL, PARALLEL, PVM3, NQS)
LOADL_PROCESSOR_LIST	Blank-delimited list of hostnames allocated for the step.

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

Return Code Values

0 the job will begin.

If the prolog program is killed, the job will not begin and a message is written to the starter log.

Sample Prolog Programs

Sample of a Prolog Program for Korn Shell:

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

Sample of a Prolog Program for C Shell:

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

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

Epilog Programs

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

Invoked as:

```
epilog_name
```

The following environment variables are set:

LOADL_ACTIVE LoadLeveler version

LOADL_STEP_ID	Job step id
LOADL_STEP_OWNER	Job step owner
LOADL_STEP_COMMAND	Command
LOADL_STEP_ARGS	Arguments
LOADL_STEP_IN	File for stdin
LOADL_STEP_OUT	File for stdout
LOADL_STEP_ERR	File for stderr
LOADL_STEP_INITDIR	Initial working directory
LOADL_STEP_CLASS	Class
LOADL_STEP_ACCT	Account number
LOADL_STEP_GROUP	Group name
LOADL_STEP_TYPE	Type (SERIAL, PARALLEL, PVM3, NQS)
LOADL_PROCESSOR_LIST	Blank-delimited list of hostnames allocated for the step.

Notes

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

Sample Epilog Programs

Sample of an Epilog Program for Korn Shell:

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

Sample of an Epilog Program for C Shell:

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

Checkpointing API

Checkpointing is a method of periodically saving the state of a job so that, if for some reason, the job does not complete, it can be restarted from the saved state. “Step 13: Enable Checkpointing” on page 56 describes how to checkpoint your jobs in various ways including system-initiated and user-initiated. This section describes the API used for user-initiated checkpointing.

To activate user-initiated checkpointing, code the LoadLeveler **ckpt()** function call in the FORTRAN or C program. Whenever this call is invoked, a checkpoint of the program will be taken.

An include file called **llapi.h** and a LoadLeveler library called **libllapi.a** are used to compile programs and link them. **libllapi.a** can be found in the **lib** subdirectory and **llapi.h** can be found in the **include** subdirectory of the release directory.

C Syntax

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

FORTRAN Syntax

```
call ckpt()
```

The Submit API

Normally, the **llsubmit** command is used to submit jobs to LoadLeveler for scheduling. However, you may also write a program using these APIs to submit jobs for scheduling.

Note: **llsubmit** is the name of the LoadLeveler command used to submit jobs as well as the function described here.

The llsubmit API

The function to submit the job is called **llsubmit**.

Syntax for llsubmit

```
int llsubmit(char *job_cmd_file, char *monitor_program,
             char *monitor_arg, LL_job *job_info, int job_version)
```

where:

job_cmd_file
is a pointer to a string containing the name of the job command file.

monitor_program
is a pointer to a string containing the name of the monitor program to be invoked when the state of the job is changed. It is set to NULL if a monitoring program is not provided.

monitor_arg
is a pointer to a string which is stored in the job proc and is passed to the monitor program. The maximum length of the string is 256 bytes. It is set to NULL if an argument is not provided.

job_info
is a pointer to a structure defined in the **llapi.h** header file. No fields are required to be filled in. Upon return, the structure will contain the number of job steps in the job command file and a pointer to an array of pointers to information about each job step. Space for the array and the job step information is allocated by **llsubmit**. The caller should free this space using the function **llfree_job_info**.

job_version
is an integer indicating the version of **llsubmit** being used. This argument should be set to **LL_JOB_VERSION** which is defined in the **llapi.h** include file.

Return Code Values

0 Job has been submitted.

-1 Job has not been submitted. Error messages are written to stderr.

Notes

LoadLeveler must be installed and configured correctly on the machine on which the submit application is run.

The uid and gid in effect when **llsubmit** is invoked is the uid and gid used when the job is run.

Syntax for llfree_job_info

```
void llfree_job_info(LL_job *job_info, int job_version);
```

where:

job_info
is a pointer to a **LL_job** structure. Upon return, the space pointed to by the `step_list` variable and the space associated with the **LL_job** step structures pointed to by the `step_list` array are freed. All fields in the **LL_job** structure are set to zero.

job_version
is an integer indicating the version of **llfree_job_info** being used. This argument should be set to **LL_JOB_VERSION** which is defined in the **llapi.h** header file.

The Monitor API

You can create a monitor program that will monitor jobs that were submitted using the **llsubmit** API. The **schedd** daemon invokes this monitor program if the **monitor_program** argument to **llsubmit** is not null. The monitor program is invoked each time a job step changes state. This means that the monitor program will be informed when the job step is started, completed, vacated, removed, or rejected.

Invoked as:

```
monitor_program job_id user_arg state exit_status
```

where:

monitor_program
is the name of the program supplied in the `monitor_program` argument passed to the **llsubmit** function.

job_id
the full id for the job step.

user_arg
the string supplied to the `monitor_arg` argument that is passed to the **llsubmit** function.

state
the current state of the job step. Possible values for the state are:

JOB_STARTED
The job step has started.

JOB_COMPLETED
The job step has completed.

JOB_VACATED
The job step has been vacated. The job step will be rescheduled if the job step is restartable or if it is checkpointable.

JOB_REJECTED
A **startd** daemon has rejected the job. The job will be rescheduled to another machine if possible.

JOB_REMOVED
The job step was cancelled or could not be started.

JOB_NOTRUN
The job step cannot be run because a dependency cannot be met.

exit_status
is the exit status from the job step. The argument is meaningful only if the state is JOB_COMPLETED.

Parallel Job API

If you are using any of the parallel operating environments already supported by LoadLeveler, you do not have to use the parallel APIs. The supported parallel operating environments have already been coded to interface with the APIs described in this section. If you have another application environment that you want to use, you need to use the APIs described here to interface with LoadLeveler.

There are two APIs you can use to allow users to submit parallel jobs, one to acquire the list of LoadLeveler selected parallel nodes and the other is to start the parallel task under the LoadLeveler **starter**. An include file called **llapi.h** and a LoadLeveler library called **libllapi.a** are used to compile and link programs. **libllapi.a** can be found in the **lib** subdirectory of the release directory and **llapi.h** can be found in the include subdirectory of the release directory.

Before reviewing the parallel APIs, review the following section that describes how parallel job submission works. Understanding this will help you to better understand the APIs.

Interaction Between LoadLeveler and the Parallel APIs

Program applications which use the parallel APIs to interface with LoadLeveler Release 1.2.1 are supported under a job type called **parallel**. When a user submits a job specifying the keyword **job_type** equal to parallel, the LoadLeveler API job control flow is as follows:

The **negotiator** will select the nodes based on the user's requested **min_processors**, **max_processors**, and **requirements** keywords. Once the nodes have been obtained, the **negotiator** will contact the **schedd** to start the job.

The **schedd** will mark the job pending and contact the affected **startd**s to start their starter processes.

One machine becomes the **Master Starter**. This machine will be a resource manager configured machine if one exists. The **Master Starter** machine is one of the selected parallel nodes. The **Master Starter** will try to allocate the machines through the resource manager when configured to do so. If allocation fails, the job will be rejected and returned to the queue. All **Starter Slaves** will be terminated.

The **Master Starter** will start the **executable** specified in the job command file. The executable is your API application and is referred to from now on as the **Parallel Master**. A LOADLBATCH environment variable will be set to yes so that the **Parallel Master** can distinguish between callers.

The **Parallel Master** must:

- Obtain the machine list through the **ll_get_hostlist** API.
- Start a task on all selected machines through the **ll_start_host** API. It is mandatory that one and only one task be started on each machine. Each task is considered a **Parallel Slave**. Acquiring the task name, path and arguments is the responsibility of the **Parallel Master**. The user may pass this information through the **arguments** or **environment** keywords in the job command file.

Once all **Parallel Slaves** have been started, the job is marked Running. Once the **Parallel Master** has exited, the job is marked Complete.

Termination Paths: The **Parallel Master** is expected to cleanup and exit when:

- All of the **Parallel Slaves** have exited.
- A negative value is returned by either the **ll_get_hostlist** or **ll_start_host** API.
- A SIGCONT, followed by a SIGTERM, is received. A SIGTERM is issued to all slaves. Possible sources include:
  - LoadLeveler receives a job cancel
  - LoadLeveler receives a stop LoadLeveler daemons command
- A SIGCONT, followed by a SIGUSR1, is received. SIGINT is issued to all slaves and the job is retruned to the queue to be redispatched. Possible sources include:
  - **Master Starter** has encountered a bad return code while monitoring the resource manager socket.
  - One of the **Parallel Slaves** terminates abnormally.

When the **Parallel Master** exits before the **Parallel Slaves**, LoadLeveler issues a SIGINT to terminate all the **Parallel Slaves**. A SIGKILL is issued to any process (Master or Slave) which does not exit within five minutes of receiving a termination signal.

ll_get_hostlist API

This API is used to obtain a list of machines from the **Master Starter** machine so that the **Parallel Master** can start the **Parallel Slaves**. The **Parallel Master** is the LoadLeveler executable specified in the job command file and the **Parallel Slaves** are the processes started by the **Parallel Master** through the **ll_start_host** API.

Syntax

```
ll_get_hostlist(struct JM_JOB_INFO* jobinfo)
```

Input

jobinfo

This is a pointer to a resource manager compatible structure available in **llapi.h**. No fields are required to be filled in. **ll_get_hostlist** allocates storage for an array of JM_NODE_INFO and returns the pointer in the jm_min_node_info pointer. It is the caller's responsibility to free this storage. **ll_get_hostlist** will update the fields marked with an arrow. The rest of the fields are used by LoadLeveler when communicating with the resource manager and will not be updated.

```
struct JM_JOB_INFO {
        int  jm_request_type;
        char  jm_job_description[50];
        enum  JM_ADAPTER_TYPE jm_adapter_type;
        int  jm_css_authentication;
  →     int  jm_min_num_nodes;
  →     struct JM_NODE_INFO *jm_min_node_info;
};
struct JM_NODE_INFO {
  →     char  jm_node_name[MAXHOSTNAMELEN];
  →     char  jm_node_address[50];
  →     int  jm_switch_node_number;
        int  jm_pool_id;
        int  jm_cpu_usage;
        int  jm_adapter_usage;
        int  jm_num_virtual_tasks;
        int  *jm_virtual_task_ids;
        enum  JM_RETURN_CODE jm_return_code;
};
```

where:

jm_min_num_nodes
is the number of elements in the array of JM_NODE_INFO structures. It is the number of hosts allocated to a job.

jm_min_node_info
It is the pointer to the array of JM_NODE_INFO structures. The first entry in this array describes the node which is mapped to task 0 in the high-performance switch partition. The second entry is mapped to task 1, and so on. For more information, refer to the *IBM RS/6000 Scalable POWERparallel Systems: Administration Guide*.

jm_node_name
is the name of the node.

jm_node_address
is the address corresponding to the adapter requested.

jm_switch_node_number
is the relative node number set only for the High Performance Switch jobs. Otherwise, it is set to -1.

Return Code Values

0 No errors

-2 Cannot get LoadLeveler jobid from environment

-5 Cannot make socket

This means that the UNIX stream socket could not be created. This socket is needed to establish communications with the **Starter** for both of the API's functions.

-6 Cannot connect to host

-8 Cannot get hostlist

The **Parallel Master** must:

1. Issue error messages as appropriate.
2. Exit when the **ll_get_hostlist** API returns with a negative return value. The **Parallel Master** exit status is included in the job mail returned to the user.

ll_start_host API

This API starts a task on a selected machine. This function must be invoked for all the machines returned from the **ll_get_hostlist** API once and only once by the **Parallel Master**. Acquiring the **start_cmd** is the responsibility of the **Parallel Master**. The user may pass this information through the **arguments** or **environment** keywords in the job command file.

Syntax

```
ll_start_host(char *node_name, char *start_cmd)
```

Input

node_name
is the name of the node.

start_cmd
is the actual command to execute on the node including flags and args.

Return Code Values

fd The socket connected to the **Parallel Slave's** standard I/O (stdio)

-1 Remote host cannot fork new process

-2 Cannot get LoadLeveler jobid from environment

-3 Cannot get my hostname

-4 Nameserver cannot resolve host

-5 Cannot make socket

This means that the UNIX stream socket could not be created. This socket is needed to establish communications with the **Starter** for both of the API's functions.

-6 Cannot connect to host

-7 Cannot send PASS_OPEN_SOCKET command to remote **startd**

The **Parallel Master** must:

1. Issue error messages as appropriate.
2. Exit when the **ll_start_host** API returns with a negative return value. The **Parallel Master** exit status is included in the job mail returned to the user.

Sample Parallel API

A sample program called **llpara_api.c** is provided in the **samples**/**llpara** subdirectory of the release directory, usually /**usr**/**lpp**/**LoadL**/**nfs**.

In order to run this example, you need to do the following:

1. Copy the sample **Makefile** and the sample program called **para_api.c** to your home directory.
2. Update the **StartCmd** variable in **para_api.c** to reflect your home directory versus /**usr**/**lpp**/**LoadL**/**nfs**/**samples**/**llpara**. For example:

   ```
   char *startCmd = "/home/user/para_api -s";
   ```

3. Issue **make** to create the executable **para_api**.
4. Update your job command file as follows:

```
#!/bin/ksh
# @ initialdir     = /home/user
# @ executable     = para_api
# @ output         = para_api.$(cluster).$(process).out
# @ error          = para_api.$(cluster).$(process).err
# @ job_type       = parallel
# @ min_processors = 2
# @ max_processors = 2
# @ queue
```

5. Submit the job command file to LoadLeveler.

   The syntax to invoke the **Parallel Master** is:

   ```
   para_api
   ```

   The syntax to invoke the **Parallel Slave** is:

```
para_api -s
```

Parallel Master: The **Parallel Master** does the following:

- Acquires the hostlist through the **ll_get_hostlist** API and prints out the returned fields.
- Starts a **Parallel Slave** task by executing the command specified in the **StartCmd** variable on all hosts returned in the hostlist.
- Acquires the socket connected to the **Parallel Slave's** standard I/O (stdio).
- Writes a command over the socket to verify stdin.
- Reads acknowledgments over the socket to verify stderr and stdout.
- Prints out host names and acknowledgments.

Example output follows:

```
num_nodes=2

name=host1.kgn.ibm.com address=9.115.8.162 switch_number=-1

name=host2.kgn.ibm.com address=9.115.8.164 switch_number=-1

Connected to host1.kgn.ibm.com at sock 3
Received acko "8000" and acke "10000" from host 0

Connected to host2.kgn.ibm.com at sock 4
Received acko "8001" and acke "10001" from host 1

<Master Exiting>
```

Parallel Slave: The **Parallel Slave** does the following:

- Reads command from stdin.
- Writes acknowledgment to stdout and stderr.

User Exits

This section discusses a user exit for handling an AFS token and a user exit for filtering a job script.

Handling an AFS Token

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

AFS_GETNEWTOKEN = *myprog*
where *myprog* is is a filter that receives the AFS authentication information on standard input and writes the new information to standard output. The filter is run when the job is started and can be used to revive a token which expired while the job was being queued.

The program uses the following structures:

ktc_principal-service
ktc_principal-client
ktc_token-token

The ktc_principal and ktc_token structures are defined in the AFS header files **afs/auth.h** and **rx/rxkad.h**.

The following environment variables are set when the you invoke this program:

LOADL_STEP_OWNER	The owner of the job
LOADL_STEP_LIMIT	The number of seconds the job is scheduled to run.

User Exit: Filtering a Job Script

You can write a program to filter a job script when the job is submitted. This program can, for example, modify defaults or perform site specific verification of parameters. To invoke the program, specifying the following keyword in your configuration file:

SUBMIT_FILTER = *myprog*

where *myprog* is called with the job file as the standard input. The standard output is submitted to LoadLeveler. If the program returns with a non-zero exit code, the job submission is cancelled.

The following environment variables are set when the you invoke this program:

LOADL_ACTIVE	LoadLeveler version
LOADL_CPU_LIMIT	CPU limit
LOADL_STEP_COMMAND	Job command file name
LOADL_STEP_OWNER	Job step owner
LOADL_STEP_CLASS	Class
LOADL_STEP_ID	Job step ID

Glossary

This section contains some of the terms that are commonly used in the LoadLeveler books and in this book in particular.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the American National Standard *Vocabulary for Information Processing* (Copyright 1970 by American National Standards Institute, Incorporated), which was prepared by Subcommittee X3K5 on Terminology and Glossary of the American National Standards Committee X3. ANSI definitions are preceded by an asterisk (*).

Other definitions in this glossary are taken from *IBM Vocabulary for Data Processing, Telecommunications, and Office Systems* (GC20-1699), *IBM DATABASE 2 Application Programming Guide for TSO Users* (SC26-4081), and *Internetworking With TCP/IP, Principles, Protocols, and Architecture*, by Douglas Comer, Copyright 1988 by Prentice Hall, Incorporated

© Copyright IBM Corp. 1993, 1995

A

AFS. Andrew File System.

AIX. Abbreviation for Advanced Interactive Executive, IBM's licensed version of the UNIX operating system. AIX is particularly suited to support technical computing applications, including high function graphics and floating point computations.

B

Berkeley Load Average. The average number of processes on the operating system's ready to run queue.

C

C. A general purpose programming language. It was formalized by ANSI standards committee for the C language (X3J11) in 1984 and by Uniforum in 1983.

client. *(1) A function that requests services from a server, and makes them available to the user. *(2) An address space in MVS that is using TCP/IP services. *(3) A term used in an environment to identify a machine that uses the resources of the network.

cluster. (1) A group of processors interconnected through a high speed network that can be used for high performance computing. (2) A group of jobs submitted from the same job command file. (3)A set of machines with something in common between them. This commonality could be that they are all backed up by one machine or they are all in the LoadLeveler administration file.

D

daemon. A process, not associated with a particular user, that performs system-wide functions such as administration and control of networks, execution of time-dependent activities, line printer spooling, and so on.

datagram. A protocal known as the User Datagram Protocol (UDP). It is an internet standard protocol that allows an application program on one machine to send a datagram to an application program on another machine. UDP uses the Internet Protocol to deliver datagrams. Conceptually, the important difference between UDP and IP is that UDP messages include a protocol port number, allowing the sender to distinguish among multiple destinations (application programs) on the remote machines. In practice, UDP also includes a checksum over the data being sent.

default. An alternative value, attribute, or option that is assumed when none has been specified.

H

host. A computer connected to a network, and providing an access method to that network. A host provides end-user services.

M

menu. A display of a list of available functions for selection by the user.

Motif. The UNIX industry's standard user interface, originally developed by the Open Systems Foundation. Motif is based on the X-Window system and is a Presentation Manager look-alike. Motif is available for all IBM AIX workstations.

N

network. An interconnected group of nodes, lines, and terminals. A network provides the ability to transmit data to and receive data from other systems and users.

NFS. Network File System.

node. In a network, the point where one or more functional units interconnect transmission lines. A computer location defined in a network.

NQS. Network Queueing System.

P

parameter. *(1) A variable that is given a constant value for a specified application and that may denote the application. *(2) An item in a menu for which the operator specifies a value or for which the system provides a value when the menu is interpreted. *(3) A name in a procedure that is used to refer to an argument that is passed to the procedure. *(4) A particular piece of information that a system or application program needs to process a request.

process. *(1) A unique, finite course of events defined by its purpose or by its effect, achieved under defined conditions. *(2) Any operation or combination of operations on data. *(3) A function being performed or waiting to be performed. *(4) A program in operation. For example, a daemon is a system process that is always running on the system.

S

server. (1) A function that provides services for users. A machine may run client and server processes at the same time. (2) A machine that provides resources to the network. It provides a network service, such as disk storage and file transfer, or a program that uses such a service.

shell. The shell is the primary user interface for the UNIX operating system. It serves as command language interpreter, programming language, and allows foreground and background processing. There are three different implementations of the shell concept: Bourne, C and Korn.

stream. An internet standard transport level protocol that provides the reliable, full duplex, stream service on which many application protocols depend. TCP allows a process on one machine to send a stream of data to a process on another. It is connection-oriented in the sense that before transmitting data, participants must establish a connection. Software implementing TCP usually resides in the operating system and uses the IP protocol to transmit information across the Internet. It is possible to terminate (shut down) one direction of flow across a TCP connection, leaving a one-way (simplex) connection. The Internet protocol suite is often referred to as TCP/IP because TCP is one of the two most fundamental protocols.

System Administrator. The user who is responsible for setting up, modifying, and maintaining LoadLeveler.

U

user. Anyone who is using LoadLeveler.

W

working directory. All files without a fully qualified path name are relative to this directory.

workstation. *(1) A configuration of input/output equipment at which an operator works. *(2) A terminal or microcomputer, usually one that is connected to a mainframe or to a network, at which a user can perform applications.

Index

A

account 25, 26
account_no 183
accounting 43, 107, 109, 128, 183
  llacctmrg command 128
  reports 109
  shifts 43
ACCT 50, 73
ACCT_VALIDATION 50, 73
Adapter 83, 193
admin 27, 32
admin keyword 32
ADMIN_FILE 52
administering LoadLeveler 15
administraion file keywords 69
administration file 206
  structure and syntax 15
administrators 41
AFS authentication 65
AFS authentication user exit 236
AFS token handling 236
AFS_GETNEWTOKEN 65, 73
AIX Error Log 113
  sample formatted output 113
AIX Error Notification Facility 114
alias 20, 21
application programming interfaces 219
ARCH 44, 73, 75
arguments 176, 184

B

BackgroundLoad 74
BIN 52

C

central manager 9, 20, 50, 176
central_manager keyword 22
CENTRAL_MANAGER_HEARTBEAT_INTERVAL 51, 73
CENTRAL_MANAGER_TIMEOUT 51, 73
changing job step priority
  using llprio 143
checkpoint keyword
  system_initiated checkpointing 184
  user_initiated checkpointing 184
checkpointing 184
  ckpt function 229
  system_initiated checkpointing 56
  user_initiated checkpointing 56
ckpt 229
class 73, 93, 184
  defining 45
class information 132
Class keyword 44
class stanza keywords
  admin 27
  class_comment 29
  core_limit 30
  cpu_limit 29
  data_limit 30
  exclude_groups 28
  exclude_users 28
  file_limit 30
  include_groups 28
  include_users 28
  master_node_requirement 28
  max_processors 29
  nice value keyword 27
  NQS_class 28
  NQS_query 28
  NQS_submit 28
  priority 27
  rss_limit 31
  stack_limit 30
class stanzas 27, 31
  examples 31
  format 27
class_comment 27, 29
ClassSysprio 47
ClassSysprio macro 75
CLIENT_TIMEOUT 66, 73
COLLECTOR 66, 73
collector daemon 217
collector daemon keywords 65
COLLECTOR_DEBUG 66
COLLECTOR_DGRAM_PORT 55, 73
COLLECTOR_LOG 66
COLLECTOR_STREAM_PORT 55, 73
commands 127
comment 184
comment statements 183
common name space 15
configuration 35
configuration file keywords 36, 37, 70, 73, 74
configuring
  configuration file 199
  global configuration file 34
  LoadLeveler user ID 34
  local configuration file 34
constants 37

© Copyright IBM Corp. 1993, 1995

continue expression 43, 73
control functions 42
copy 29
core_limit 27, 30, 185
CPU_Busy 74
CPU_Idle 74
cpu_limit 27, 29, 185
cpu_speed_scale 20, 23
Cpus 38, 48
CurrentTime 38
CUSTOM 96
CUSTOM_METRIC 44, 73
CUSTOM_METRIC_COMMAND 44, 73
customizing 35

D

daemons
  collector 217
  kbdd 217
  master 213
  negotiator 217
  schedd 214
  startd 215
data_limit 27, 30, 186
debugging 53
debugging information 113
  error logging 113
default_class 25
default_group 25, 26
dependency 186
diagnosing 173
Disk 38, 49, 75
domain (domain name) 38, 75
drain 39

E

EnteredCurrentState 38, 75
environment 187
epilog programs 55, 222
error 188
error log events
  classification 113
error logging
  definition 113
  DETECTING MODULE 113
  getting notified 114
  overview 113
  switch error log reports 114
etc/services 54
exclude_groups 27, 28
exclude_users 27, 28, 32, 33
executable 188
EXECUTE 52, 73
expressions 42
  continue
  kill
  start
  suspend
  vacate
extended report, accounting 110

F

favor jobs 168
  llfavorjob command 137
favor users 167
  llfavoruser command 138
feature 45, 73
file_limit 27, 30, 189
filtering a job script 237
filtering job scripts 237
flush 39

G

GetHistory 110, 220
GLOBAL_HISTORY 50, 73, 109
graphical user interface (GUI) 166
group
  default 26
  in job command file 189
  LoadLeveler 33
  UNIX 26, 33
group stanza keywords
  admin 32
  exclude_users 33
  include_users 33
  max_processors 33
  maxjobs 32
  maxqueued 33
  priority 32
group stanzas
  examples 33
  format 32
GroupRunningJobs macro 48, 75
GroupSysprio macro 48, 75

H

help 181
high performance switch error reports 114
HighLoad 74
HISTORY 52, 73
hold 189
host or hostname 38, 75
HOUR 74

I

include_groups 27, 28
include_users 27, 28, 32, 33
initialdir 189
Initiators 38
interactive login sessions 89
ISS
  CLASS 93
  configuring
  CUSTOM 96
  ending an interactive session 102
  establishing an interactive session 102
  installing 89
  ISS custom configuration 90
  ISS round-robin configuration 90
  Iss_config file
  ISS_TRACE 92, 95
  METRIC 95
  NAME 93
  NAMED_BOOTFILE 92
  POLICY 94
  POLL 94
  ROUNDROBIN 97
  SERVERS 95
  starting ISS 101
  stopping ISS 102
  troubleshooting 102
ISS_TRACE 92, 95

J

job accounting 107
job script filter 237
job status 145
JOB_ACCT_Q_POLICY 73, 107
job_cpu_limit 27, 190
JOB_EPILOG 55, 73, 222
JOB_LIMIT_POLICY 73, 107
job_name 190
JOB_PROLOG 55, 73, 222
job_type 190
JOB_USER_EPILOG 56, 73, 222
JOB_USER_PROLOG 55, 73, 222
JobLoad 74

K

KBDD 73
kbdd daemon 217
keyboard daemon keywords
  KBDD 64
  KBDD_DEBUG 64
  KBDD_DEVICE 64
  KBDD_LOG 64
  MAX_KBDD_LOG 64
  MAX_STARTD_LOG 64
keyboard daemon keywords *(continued)*
  MOUSE_DEVICE 64
  STARTD 64
  STARTD_LOG 64
  X_RUNS_HERE 64
KeyboardBusy 74
KeyboardIdle 39
keywords
  administration file 20—41, 69
  configuration file 35, 36, 37, 41—68, 70, 73, 74
    reserved macros 37, 74
    user-defined 36, 73
  job command file 183—197
  reserved 37—41
  user-defined 36—37
kill expression 43, 73

L

LIB 52, 73
limits 63
llacctmrg 128
llcancel 130
llclass 132
llctl 134
llfavorjob 137
llfavoruser 138
llhold 139
llinit 141
llprio 143
llq 145
llstatus 151
llsubmit 156
llsummary 157
LoadAvg 39, 48
loadl user ID 16, 34
LOADL_ADMIN 41
LoadL_admin file 206
LoadLeveler user ID 16
LOCAL_CONFIG 52, 73
LOG 52, 73
log files 52
  COLLECTOR_LOG 53
  KBDD_LOG 53
  MASTER_LOG 53
  MAX_KBDD_LOG 53
  MAX_NEGOTIATOR_LOG 53
  MAX_STARTER_LOG 53
  NEGOTIATOR_LOG 53
  SCHEDD_LOG 53
  STARTD_LOG 53
  STARTER_LOG 53
login 179
LowLoad 74

M

Machine 39, 75
machine feature 45
machine stanza keywords
  alias 21
  central_manager 22
  cpu_speed_scale 23
  master_node_exclusive 23
  max_jobs_scheduled 23
  name_server 24
  pvm_root 22
  rm_host 23
  schedd_host 22
  speed 22
  submit_only 22
machine stanzas
  examples 24
  format 20
machine status 151
MACHINE_AUTHENTICATE 44, 73
MACHINE_UPDATE_INTERVAL 66, 73
MACHPRIO 48, 73
macros 37
  ARCH 44
  Cpus 38
  CurrentTime 38
  Disk 38
  domain (domain name) 38
  EnteredCurrentState 38
  host or hostname 38
  Initiators 38
  KeyboardIdle 39
  LoadAvg 39
  Machine 39
  QDate 39
  state 39
  status 39
  tilde 39
  UserPrio 39
MAIL keyword 37
MASTER 68
master daemon 213
master daemon keywords
  MASTER 68
  MASTER_DEBUG 68
  MASTER_LOG 68
  MAX_MASTER_LOG 68
  OBITUARY_LOG_LENGTH 68
  PUBLISH_OBITUARIES 68
  RESTARTS_PER_HOUR 68
MASTER expression 73
MASTER_DEBUG 68
MASTER_DGRAM_PORT 55, 73
MASTER_LOG 68
master_node_exclusive 23
master_node_requirement 28
MASTER_STREAM_PORT 55, 73
MAX_CKPT_INTERVAL 58, 73
MAX_COLLECTOR_LOG 66
max_jobs_scheduled 20, 23
MAX_MASTER_LOG 68
max_processors 25, 26, 27, 32, 33, 190
MAX_SCHEDD_LOG 65
MAX_STARTERS 73
maxjobs 25, 32
maxqueued 25, 26, 32, 33
Memory 48, 75
messages 180
METRIC 95
MIN_CKPT_INTERVAL 58, 73
min_processors 191
MINUTE 74

N

NAME 93
name_server 20, 24
NAMED_BOOTFILE 92
named.boot 99
named.data 99
NEGOTIATOR 73
negotiator daemon 217
negotiator daemon keywords
  MAX_NEGOTIATOR_LOG 66
  NEGOTIATOR 66
  NEGOTIATOR_DEBUG 66
  NEGOTIATOR_INTERVAL 66
  NEGOTIATOR_LOG 66
  NEGOTIATOR_PARALLEL_DEFER 66
  NEGOTIATOR_PARALLEL_HOLD 66
  NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL 67
  NEGOTIATOR_REDRIVE_PENDING 66
  NEGOTIATOR_REMOVE_COMPLETED 66
  NEGOTIATOR_RESCAN_QUEUE 66
NEGOTIATOR_INTERVAL 73
NEGOTIATOR_PARALLEL_DEFER 73
NEGOTIATOR_PARALLEL_HOLD 73
NEGOTIATOR_RECALCULATE_SYSPRIO_INTERVAL 67, 73
NEGOTIATOR_REDRIVE_COMPLETED 73
NEGOTIATOR_REDRIVE_PENDING 73
NEGOTIATOR_RESCAN_QUEUE 73
NEGOTIATOR_STREAM_PORT 55, 73
nice value 27
notification 191
notify_user 191
NQS
  options 123
  routing jobs to NQS machines 121

NQS jobs
cancel 125
status 124
submit 123
NQS scripts 125
NQS_class 27, 122
NQS_DIR 52, 73, 122
NQS_query 27, 122
NQS_submit 27, 122

O

OBITUARY_LOG_LENGTH 68, 73
operators 37
OpSys 75
output 191

P

parallel jobs, submitting 82
parallel_path 192
performance 15
planning considerations 15
POLICY 94
POLL 94
POLLING_FREQUENCY 73
POLLS_PER_UPDATE 73
Pool 83, 194
port numbers 54
preferences 192
priority 25, 27, 32, 143
processes
starter 216
prolog programs 55, 222
PUBLISH_OBITUARIES 68, 73
pvm jobs 80
pvm_root 20, 22

Q

QDate 39, 47, 75
query a job
llq command 145
query class information
llclass command 132
querying multiple clusters 61
questions and answers 173
queue 192

R

release from hold 168
RELEASEDIR 52, 73
requirements 192
reserved macros 37, 74
restart 194
RESTARTS_PER_HOUR 68, 73
rlim_infinity 29
rm_host 20, 23
ROUNDROBIN 97
rss_limit 27, 31, 194

S

sample formatted output
AIX Error Log 113
SCHEDD 65, 73
schedd daemon 214
schedd daemon keywords
AFS_GETNEWTOKEN 65
MAX_SCHEDD_LOG 65
SCHEDD 65
SCHEDD_DEBUG 65
SCHEDD_INTERVAL 65
SCHEDD_LOG 65
SCHEDD_DEBUG 65
schedd_host 20, 22
SCHEDD_INTERVAL 65
SCHEDD_LOG 65
SCHEDD_RUNS_HERE 45, 73
SCHEDD_STREAM_PORT 55, 73
SERVERS 95
service numbers 54
shell 195
shifts 43
short report, accounting 110
signals 232
speed 20, 22, 48, 112
SPOOL 52, 73
log 52
stack_limit 27, 30, 195
stanzas
class 27
default 20
label 19
machine 20
type 19
user 20
start expression 43, 73
start LoadLeveler 168
START_DAEMONS 44, 73
startd daemon 215
startd daemon keywords
JOB_ACCT_Q_POLICY 64
JOB_LIMIT_POLICY 64
POLLING_FREQUENCY 64
POLLS_PER_UPDATE 64
STARTD expression 73
STARTD_DEBUG 64
STARTD_DGRAM_PORT 55, 73
STARTD_RUNS_HERE 45, 73

STARTD_STREAM_PORT 55, 73
startdate 195
STARTER expression 73
starter process 216
starter process keywords
  MAX_CKPT_INTERVAL 67
  MAX_STARTER_LOG 67
  MIN_CKPT_INTERVAL 67
  STARTER 67
  STARTER_DEBUG 67
  STARTER_LOG 67
state 39, 75
StateTimer 74
status 39, 75, 151
step_name 195
stop LoadLeveler 168
SUBMIT_FILTER 73, 237
submit_only 20, 22
submitting jobs 156
support services 181
suspend 39
suspend expression 43, 73
switch error log 114
syntax, administration file 15
syshold 168
SYSPRIO 73
system administrator vii
system_initiated checkpointing 56, 184

T

tbx adapters error reports 114
TCP/IP nameserver 105
TCP/IP service and port numbers 54
tilde 39, 75
tm_hour 40, 75
tm_isdst 40, 75
tm_mday 40, 75
tm_min 40, 75
tm_mon 40, 75
tm_sec 40, 75
tm_wday 40, 75
tm_yday 40, 75
tm_year 40, 75
TRUNC statements 53, 73
  TRUNC_COLLECTOR_LOG_ON_OPEN 53
  TRUNC_KBDD_LOG_ON_OPEN 53
  TRUNC_MASTER_LOG_ON_OPEN 53
  TRUNC_NEGOTIATOR_LOG_ON_OPEN 53
  TRUNC_SCHEDD_LOG_ON_OPEN 53
  TRUNC_STARTD_LOG_ON_OPEN 53
  TRUNC_STARTER_LOG_ON_OPEN 53

U

unfavor jobs 168
unfavor users 167
Unix_group 33
USE_X_IDLE_EXTENSION 73
user name 35
user stanza keywords
  account 26
  default_class 25
  default_group 26
  max_processors 26
  maxjobs 25
  maxqueued 26
  priority 25
user stanzas
  examples 26
  format 20
user-defined keywords 36
user_initiated checkpointing 56, 184
user_priority 196
UserPrio 39, 48, 75
UserRunningJobs 48
UserRunningJobs macro 75
UserSysprio 47, 75

V

vacate expression 43, 73
virtual memory 75
VirtualMemory 39, 49

X

X_RUNS_HERE 45, 73, 213, 217
xloadl 166, 167
Xloadl_so 167

Communicating Your Comments to IBM

IBM LoadLeveler Administration Guide
Release 2.1

Publication No. SH26-7220-03

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:

  FAX: (International Access Code)+1+914+432-9405
- If you prefer to send comments electronically, use this network ID:
  - IBMLink: KGNRCF at KGNVMC
  - IBM Mail Exchange: USIB27BQ at IBMMAIL
  - Internet: kgnrcf@vnet.ibm.com

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Readers' Comments — We'd Like to Hear from You

IBM LoadLeveler Administration Guide
Release 2.1

Publication No. SH26-7220-03

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Today's date: _____

What is your occupation?

Newsletter number of latest Technical Newsletter (if any) concerning this publication:

How did you use this publication?

[ ]	As an introduction	[ ]	As a text (student)
[ ]	As a reference manual	[ ]	As a text (instructor)
[ ]	For another purpose (explain)		

_____

_____

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number: Comment:

Name	Address
Company or Organization	
Phone No.	

Readers' Comments — We'd Like to Hear from You
SH26-7220-03

Cut or Fold
Along Line

IBM®

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 55JA Mail Station P384
522 South Road
Poughkeepsie, NY 12601-5400

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

IBM®

Printed in U.S.A.

SH26-7220-03