LoadLeveler User's Guide
Release 2.1

SH26-7226-03

IBM

LoadLeveler User's Guide
Release 2.1

SH26-7226-03

Note

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

| **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).

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

Chapter 1. Getting Started with LoadLeveler 1
What Tasks Can I Accomplish Using LoadLeveler? 2
How Do I Use LoadLeveler's Graphical User Interface? 15

Chapter 2. Building and Submitting Jobs Using the Graphical User Interface 19
Starting Up the LoadLeveler Graphical User Interface 19
Task Scenario Using the Graphical User Interface 20

Chapter 3. Building and Submitting Jobs Using LoadLeveler Commands 35
Task Scenario Using the Commands 35

Chapter 4. LoadLeveler Commands 39

Appendix A. LoadLeveler Job Command File Keywords and Examples . 61
Building a Job Command File 61
Job Command File Syntax 61
LoadLeveler Keywords 62
Job Command File Macros 75
Job Command File Examples 76

Appendix B. Graphical User Interface Information 83
Customizing the Graphical User Interface 85
Customizing the skel.cmd File 100

Glossary 103

© Copyright IBM Corp. 1993, 1995

Index 107

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 IBM products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

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

IBM 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:

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

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

How to Use This Book

Overview of Contents

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

Chapter 2, “Building and Submitting Jobs Using the Graphical User Interface” on page 19 provides a task scenario that shows how to use the interface.

Chapter 3, “Building and Submitting Jobs Using LoadLeveler Commands” on page 35 provides a task scenario that shows how to use the commands.

Chapter 4, “LoadLeveler Commands” on page 39 provides an alphabetical listing of the commands and their syntax.

Appendix A, “LoadLeveler Job Command File Keywords and Examples” on page 61 provides the keywords you can include in job command files and examples of job command files.

Appendix B, “Graphical User Interface Information” on page 83 provides general information on graphical user interfaces such as windows, menus, and dialog boxes.

A glossary and index are also included.

Terms and Document Conventions

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

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

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

Related Publications

IBM LoadLeveler: Administration Guide, SH26-7220

IBM LoadLeveler Technical Presentation, ZZ81-0348-00

IBM LoadLeveler Memo To Users, included on the product tape.

Ordering Additional Copies of This Book

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

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 SP2 and SP1 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.

Program service for Version 1 Release 2 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 in the *LoadLeveler Administration 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.

New schedd Keywords

- **MAX_JOB_REJECT**

  Sepcifies the number of times a job is rejected before it is cancelled or put in User Hold staus.

- **ACTION_ON_MAX_REJECT**

  Determines whether a job is cancelled or put in User Hold status when the job exceeds the **MAX_JOB_REJECT** value.

Error Logging Enhancements

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

Chapter 1. Getting Started with LoadLeveler

This chapter introduces you to the LoadLeveler program product, otherwise referred to as LoadLeveler throughout this manual. It provides an overview of LoadLeveler and describes the tasks you can perform with LoadLeveler, using either the command line or the LoadLeveler the Graphical User Interface (GUI).

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.

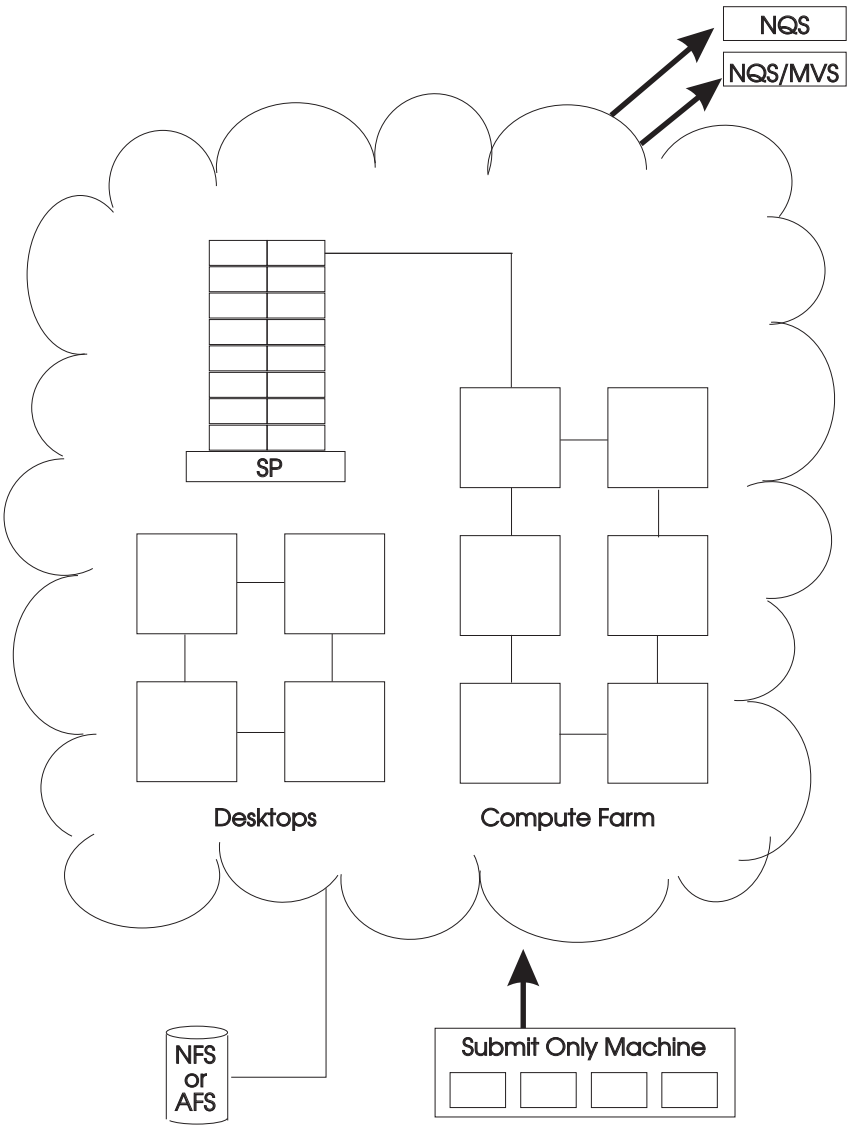


Figure 1. High-Level LoadLeveler Configuration

The items within the cloud figure represent the different environments to which LoadLeveler can schedule jobs. The environments are heterogeneous pools, dedicated nodes, and the RISC System/6000 Scalable POWERparallel (SP)

© Copyright IBM Corp. 1993, 1995

system. In addition, LoadLeveler can schedule jobs written for NQS to machines outside of the LoadLeveler pool 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. The users of these machines are allowed to submit LoadLeveler jobs, as well as cancel and query them.

Refer to the LoadLeveler Administration Guide

Refer to Chapter 1 of the LoadLeveler Administration Guide for detailed explanations that describe what LoadLeveler is and how it works.

What Tasks Can I Accomplish Using LoadLeveler?

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

Working with Jobs and Job Steps

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 each job step contains an *executable*. You can use LoadLeveler to sequentially run several jobs steps, each made up of one or more job steps and each one dependent upon the completion status of the previous job step. For example, the following diagram illustrates a stream of job steps:

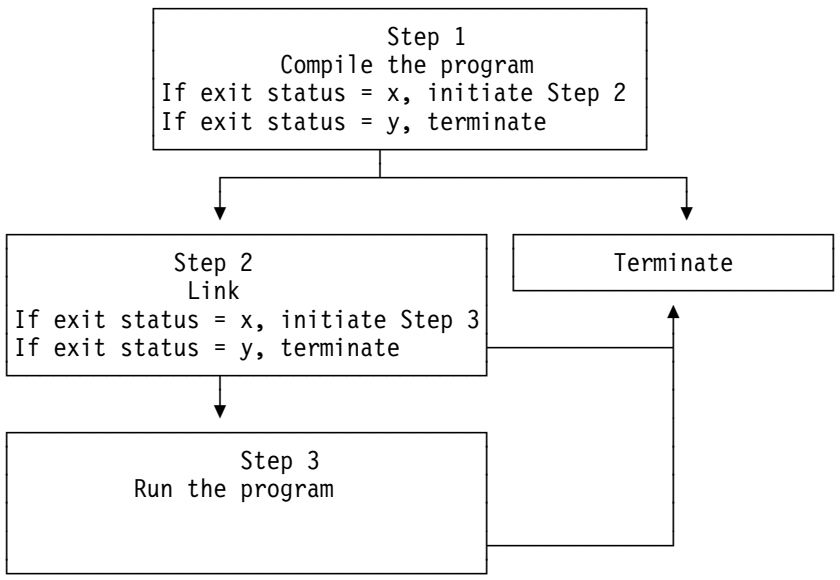


Figure 2. Illustration of Job Steps

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.

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.

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 itself 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 LoadLeveler keyword or keyword statement is preceded by # @. These lines are treated as comments by the UNIX shell. For information on these keywords, refer to Appendix A, "LoadLeveler Job Command File Keywords and Examples" on page 61

- Comment statements

  You can use comments to document your job command files, making them 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 LoadLeveler keyword statement is regarded as a comment.

You can build a job command file either by using the GUI or by using a text editor such as **vi**.

For jobs with multiple job steps, LoadLeveler determines whether a job step should run based upon the exit status of the previously run job step.

To specify a stream of job steps, you need to list each job step in the job command file. For example, the following job command file lists two separate job steps. Notice that `step1` is the first job step to run and that `step2` is a job step that runs only if `step1` exits with the correct exit status.

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

In this example, `step1` is called the *sustaining* job step. `step2` is called the *dependent* job step because whether or not it begins to run is dependent upon the exit status of `step1`. A single sustaining job step can have more than one dependent job step and a dependent job step can also have job steps dependent upon it.

In this example, each job step has its own executable, input, output, and error statements. Your job steps can have their own separate statements as well or they can use the same ones defined for the previous job step. For example, in the following job command file, `step2` uses the executable statement defined for `step1`:

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

Serial Job Command File

The following is an example of a simple serial job command file. For information on the keywords used in job command files, refer to Appendix A, “LoadLeveler Job Command File Keywords and Examples” on page 61.

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

Parallel Job Command File

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

To construct a parallel job command file, specify the number of machines the parallel job requires to be processed. To do this, use the keywords **min_processors** and **max_processors** to specify a range of processors needed. In addition, you could use the **job_type** keyword to specify the type of parallel job you want to run. You can also specify, using the **requirements** keyword, the type of adapter and parallel pool you want to use. You can find more information on using the parallel job command file keywords as well as examples by referring to "Example 5: Building and Submitting a Parallel Job Command File" on page 79.

For example, assume you wanted to run a PVM parallel program called *my_parallel_program* that requires a minimum of five parallel nodes to run. Your job command file would look similar to:

```
#
# @ executable = my_parallel_program
# @ input = job.in
# @ output = job.out
# @ error = job.err
# @ min_processors = 5
# @ max_processors = 7
# @ job_type = pvm3
# @ checkpoint = no
# @ queue
```

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

Editing a Job Command File

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

Checkpointing a Job Step

Checkpointing is a method of periodically saving the state of a job so that, if for some reason, the job does not complete, it can be restarted from the saved state. Users can checkpoint their serial jobs. They cannot, however, with LoadLeveler Release 1.2.0, 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 has to wait for the machine to restart before scheduling the job to run on another machine because only by recognizing that the machine has restarted, can LoadLeveler know that the machine had ever failed.

The LoadLeveler administrator can manually force jobs enabled for checkpointing on a machine to be checkpointed and vacated using the **llctl** flush command. In this case, no new jobs are redispatched to this machine until the **resume** option is issued on the **llctl** command. For more information on this command, refer to the *LoadLeveler Administration Guide*.

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

C Example

```
/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 $LL_LIBDIR/libloadl.a
/usr/lib/librpcsvc.a /usr/lib/libsun.a -lm -lc /usr/lib/nonshared/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

Guidelines to Follow When Checkpointing Jobs

Review the following guidelines before submitting a checkpointing job:

Reserve Adequate Disk Space in the Execute Directory: The execute directory is the directory from which LoadLeveler executes the job executable. 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 executable is created from the original executable and the core file. The original executable may be small in size but the new executable can be much larger. This is because the new executable is made up from the original one and from sections of the core file.

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

```
# @ core_limit = unlimited, unlimited
                     ↑          ↑
                     │          └──────── 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

Another limitation of checkpointing jobs is file I/O. The requirement is 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.

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. 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. For information on the **ckpt()** function call, refer to the Application Programming Interfaces appendix in the *LoadLeveler Administration Guide*.
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. Administrators should refer to the *LoadLeveler Administration Guide* for this information. **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 specify **checkpoint=system_initiated** and code the **ckpt()** function call.
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.

Setting and Changing the Priority of a Job

LoadLeveler 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 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

Submitting a Job Command File

After building a job command file, you can submit it for processing either to a machine in the LoadLeveler pool or one outside of the pool.

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

- Machine name

  The machine name is the name of the machine that scheduled the job, not necessarily the name of the machine that ran the job.

- Job id

  The job id was previously referred to in LoadLeveler documentation as the *cluster id*. A *job id*, is an identifier that is given to a group of jobs that were initiated from the same job command file. For example, if you created a job command file that submitted the same program five times (using 5 queue statements) possibly with different input and output, each program would have the same job id.

- Step Id

  This id is unique for every job step in the job you submit. If a job command file contains multiple job steps, every job step will have a unique step id but the same job id.

Submitting a Job Command File to be Routed to NQS Machines: When submitting a job command file to be routed to an NQS machine for processing, the job command file must also contain the shell script to be submitted to the NQS node. NQS accepts only shell scripts, binaries are not allowed. All options in the command file pertaining to scheduling are used by LoadLeveler to schedule the job. When the job is dispatched to the node running the specified NQS class, the LoadLeveler options pertaining to the runtime environment are converted to NQS options and the job is submitted to the specified NQS queue.

LoadLeveler command file options are used as follows:

arguments error message generated and job not submitted

checkpoint error message generated and job not submitted

class used only for LoadLeveler scheduling

core_limit converted to **-lc** option

cpu_limit converted to **-lt** option

data_limit converted to **-ld** option

environment	if COPY_ALL is specified, the option is converted to **-x**, otherwise error message generated and job not submitted
error	converted to **-e**
executable	error message generated and job not submitted
file_limit	converted to **-lf** option
hold	used only for LoadLeveler scheduling
image_size	error message generated and job not submitted
initialdir	error message generated and job not submitted
input	error message generated and job not submitted
notification	If the option specified is
	always converted to **-mb** and **-me** options
	error converted to **-me** option
	start converted to **-mb** option
	never ignored
	complete converted to **-me** option
notify_user	converted to **-mu** option
output	converted to **-o** option
preferences	used only for LoadLeveler scheduling
queue	places one copy of job in the LoadLeveler queue
requirements	used only for LoadLeveler scheduling
restart	If the option specified is
	yes ignored
	no converted to **-nr** option
rss_limit	converted to **-lw** option
shell	converted to **-s** option
stack_limit	converted to **-ls** option
start_date	used only for LoadLeveler scheduling
user_priority	used only for LoadLeveler scheduling

You can also submit an NQS script. In this case, any NQS options in the script are used to schedule the job and once dispatched by LoadLeveler, the file is sent to NQS unmodified.

Submitting a Job Command File Using a Submit-Only Machine: You can take advantage of the submit-only machine feature that allows machines that are not part of the LoadLeveler pool to submit jobs to the pool for processing. You can submit a job using either the submit-only version of the GUI or the **llsubmit** command.

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

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

For more information on the **llsubmit** command, see “llsubmit - Submit a Job” on page 59.

Querying the Status of a Job

Once you submit a job, you can query the status of the job to determine, for example, if it is still in the queue or if it is running. You also receive other job status related information such as the job id and job owner.

Querying the Status of a LoadLeveler Job: You can query the status of a LoadLeveler job either by using the GUI or the **llq** command.

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

Querying the Status of a Job Using a Submit-Only Machine: The submit-only machine, in addition to allowing you to submit and cancel jobs, allows you to query the status of jobs. You can query a job using either the submit-only version of the GUI or by using the **llq** command. For information on **llq**, see “llq - Query Job Status” on page 48.

Placing and Releasing a Hold on a Job

You may place a hold on a job which means that the job remains in the queue until you release it.

There are two types of holds: a user hold and a system hold. Both you and your system administrator can place and release a user hold on a job. Only a system administrator, however, can place and release a system hold on a job. For more information on system holds, see the *IBM LoadLeveler: Administration Guide*, SH26-7220.

You can place a hold on a job or release the hold either by using the GUI or the **llhold** command.

As a user or system administrator, you can also use the **startdate** keyword described in Appendix A, “LoadLeveler Job Command File Keywords and Examples” on page 61 to place a hold on a job. This keyword allows you to specify when you want to run the job. If you don't specify this keyword in your job command file, the job will run at the earliest possible time. The syntax is:

```
startdate = date  time
```

where the format of the date is `MM/DD/YY` and the format of the time is `HH:mm(:ss)`. For example, to specify that a job run on November 23, 1994 at 1:30 pm, you would enter:

```
startdate = 11/23/94 13:30
```

Cancelling a Job

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

Cancelling a LoadLeveler Job: You can cancel one of your own jobs either by using the GUI or the **llcancel** command.

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

Cancelling a Job Using a Submit-Only Machine: The submit-only machine, in addition to allowing you to submit and query jobs, allows you to cancel jobs. You can cancel a job using either the submit-only version of the GUI or by using the **llcancel** command.

Working with the Machines

Throughout this book, the terms *client*, *workstation*, *machine*, and *node* refer to the machines in your pool. For information on the roles these machines can play, refer to Chapter 1 of the *LoadLeveler Administration Guide*.

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

- Display Machine Status

  When you submit a job to a machine, the status of the machine automatically appears in the Machines window on the GUI. This window displays machine related information such as the names of the machines running jobs, as well as the machine's architecture and operating system. For detailed information on one or more machines in the pool, you can use the Details option on the Actions pull-down menu. This will provide you with a detailed report that includes information such as the machine's state and amount of installed memory.

- Display Central Manager

  The LoadLeveler administrator designates one of the machines in the LoadLeveler pool as the central manager. When jobs are submitted at any machine, the central manager is notified and decides where to schedule the jobs. In addition, it keeps track of the status of machines in the pool and jobs in the system by communicating with each machine. LoadLeveler uses this information to make the scheduling decisions and to respond to queries.

  Usually, the system administrator is more concerned about the location of the central manager than the typical end user but you may also want to determine its location. One reason why you might want to locate the central manager is if you want to browse some configuration files that are stored on the same machine as the central manager.

- Display Public Scheduling Machines

  Public scheduling machines are machines that participate in the scheduling of LoadLeveler jobs on behalf of users at submit-only machines and users at other workstations that are not running the **schedd** daemon. For more information on LoadLeveler daemons, refer to the *LoadLeveler Administration Guide*. You can find out the names of all these machines in the pool.

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

How Do I Use LoadLeveler's Graphical User Interface?

This section briefly describes LoadLeveler's Graphical User Interface (GUI). For detailed information on a GUI and how to use one, refer to Appendix B, “Graphical User Interface Information” on page 83, or to the appropriate Motif documentation.

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

LoadLeveler's Windows

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

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

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

Messages window
Use to view messages generated by LoadLeveler.

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

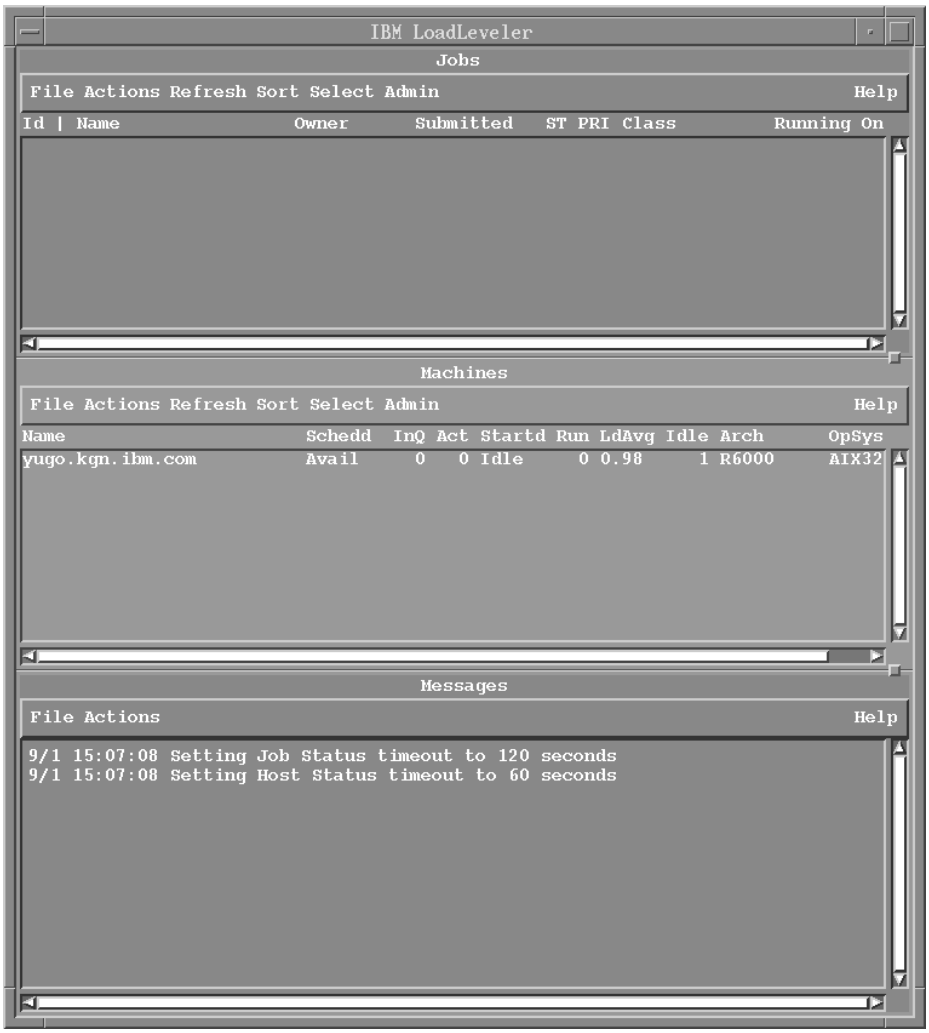


Figure 3. Main Window of the LoadLeveler GUI

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

Another example of a secondary window is a *dialog box*. A dialog box is a movable window that contains controls such as push buttons. You use these controls to provide information to LoadLeveler. The push buttons are arranged across the bottom of the dialog box. There are a number of common dialog box actions. For more information on the push buttons available in LoadLeveler dialog boxes, refer to Appendix B, “Graphical User Interface Information” on page 83.

LoadLeveler's Menu Bars and Pull-down Menus

LoadLeveler's primary window has three sub-windows. Each of these sub-windows has its own menu bar. For a definition of a menu bar, refer to Appendix B, “Graphical User Interface Information” on page 83.

The menu bar on the Jobs window relates to actions you can perform on jobs. The menu bar on the Machines window relates to actions you can perform on machines. Similarly, the menu bar on the Messages window displays actions you can perform related to LoadLeveler generated messages.

When you select an item from a menu bar, a pull-down menu appears. You can select an item from the pull-down menu to carry out an action or to bring up another pull-down menu originating from the first one.

Getting Help Using the Graphical User Interface

You can get help when using the GUI by pressing the HELP key. This key is function key 1 (F1) on most keyboards. To receive help on parts of LoadLeveler, place the cursor over the area or field on which you want help and press HELP. A help screen will appear that describes that area. You can also get help by using the Help pull-down menu and the Help push buttons available in pop-up windows.

Before you use the GUI, make sure your PATH statement includes the directory containing the LoadLeveler executable. Otherwise, some GUI functions may not work correctly.

Differences Between LoadLeveler's Graphical User Interface and Other Graphical User Interfaces

LoadLeveler's GUI contains many items common to other GUIs. There are, however, some differences that you should be aware of. These differences are:

- Accelerators or mnemonics do not appear on the menu bars.
- Submerged windows do not necessarily rise to the top when refreshed.

Customizing the Graphical User Interface

You can customize the GUI to suit your needs by overriding default settings. For example, you can set the color, initial size, and location of the main window. For more information, refer to either the Xloadl file or the Xloadl_so (for submit-only machines) that exists under **/usr/lib/X11/app-defaults**. See “Customizing the Graphical User Interface” on page 85.

Chapter 2. Building and Submitting Jobs Using the Graphical User Interface

This chapter explains how to build and submit a job to LoadLeveler using the GUI. In addition, you will learn how to perform other job related tasks. You can accomplish these same tasks by using the LoadLeveler commands. For information on these commands, refer to Chapter 3, “Building and Submitting Jobs Using LoadLeveler Commands” on page 35.

This manual presents step-by-step instructions for performing tasks. For each step in a task, a user action and a system response to the action are included. User actions appear in uppercase boldface type, for example **PRESS**. The system response to an action follows a ▲. For example:

▲ The main window appears.

An action is sometimes represented by itself. For example:

PRESS OK

Other actions can require a selection or decision. Selection and decision actions are presented in tables.

Selection tables list all possible selections in the left column of the table. The following is an example of a selection table:

To:	Do this:
Submit a job	Refer to “Step 3: Submit a Job Command File” on page 27.
Cancel a job	Refer to “Step 9: Cancel a Job” on page 30.

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

Did the job you submitted complete processing?	
Yes	Submit another job.
No	Check the status of the job.

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

SELECT Actions → Cancel

Starting Up the LoadLeveler Graphical User Interface

To start up the GUI, check your path variable to ensure that it is pointing to the LoadLeveler binaries. Then, check to see that your DISPLAY variable is set to your display to ensure that the GUI appears on your display. Finally, type one of the following to bring up the GUI and to run it as a background process:

© Copyright IBM Corp. 1993, 1995

xloadl_so & for users of submit-only machines

xloadl & for all other users.

Task Scenario Using the Graphical User Interface

The tasks described in this chapter are those that you, as a user might be interested in accomplishing and are presented in a typical step-by-step scenario. You do not have to follow the steps shown here and may perform certain tasks before others without any difficulty. Some tasks must be performed prior to others in order for succeeding tasks to work. For example, you cannot submit a job if you do not have a job command file that you built using either the GUI or the editor.

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

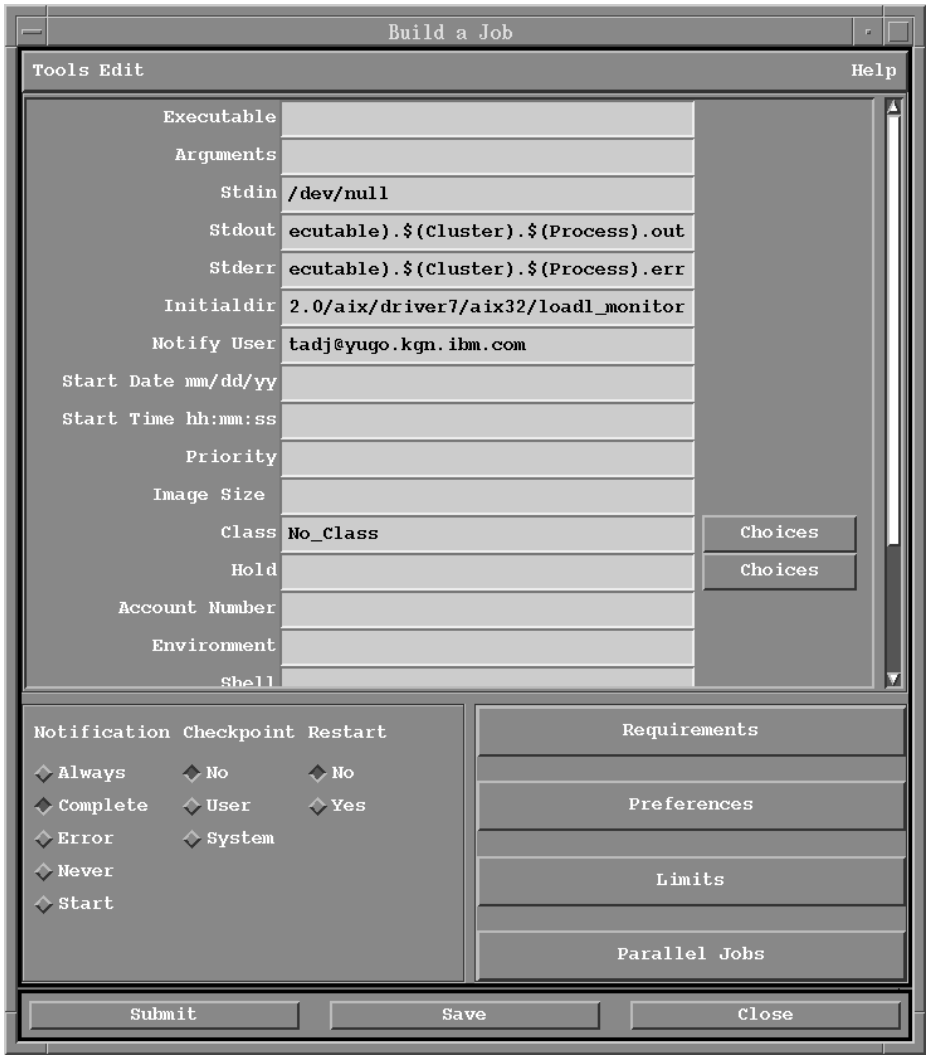
To:	Go to:
Build a job	Step 1
Cancel a job	Step 9
Change priorities of jobs in a queue	Step 6
Display, refresh, and obtain job status	Step 4
Display and refresh machine status	Step 10
Edit the job command file	Step 2
Find the location of the central manager	Step 12
Find the location of the public scheduling machines	Step 13
Hold a job	Step 7
Release a hold on a job	Step 8
Save LoadLeveler messages in a file	Step 16
Sort the Jobs window	Step 5
Sort the Machines window	Step 11
Specify which jobs appear in Jobs window	Step 14
Specify which machines appear in Machines window	Step 15
Submit a job command file	Step 3

Step 1: Build a Job

From the Jobs window:

SELECT File → Build a Job

▲ The Build a Job dialog box appears:



Complete those fields for which you want to override what is currently specified in your **skel.cmd** defaults file. For more information on this file, refer to “Customizing the skel.cmd File” on page 100.

Field	Input
Executable	Name of the program to run. It must be an executable file. Optional. If omitted, the command file is executed as if it were a shell script.
Arguments	Parameters to pass to the program. Required only if the executable requires them.
Stdin	Filename to use as standard input (stdin) by the program. Optional. The default is /**dev**/**null**.
Stdout	Filename to use as standard output (stdout) by the program. Optional. The default is /**dev**/**null**.
Stderr	Filename to use as standard error (stderr) by the program. Optional. The default is /**dev**/**null**.

Field	Input
Initialdir	Initial directory. LoadLeveler changes to this directory before running the job. Optional. The default is your current working directory.
Notify User	User id of person to notify regarding status of submitted job. Optional. The default is your userid.
StartDate*	Month, day, and year in the format mm/dd/yy. The job will not start before this date. Optional. The default is to run the job as soon as possible.
StartTime*	Hour, minute, second in the format hh:mm:ss. The job will not start before this time. Optional. The default is to run the job as soon as possible.
Priority	Number between 0 and 100, inclusive. Optional. The default is 50. This is the user priority. For more information on this priority, refer to “Setting and Changing the Priority of a Job” on page 10.
Image size	Number in kilobytes that reflects the maximum size you expect your program to grow to as it runs. Optional.
Class	Class type. The job will only run on machines that support the specified class type. Your system administrator defines the class types. Optional. You can press the Choices button to get a list of available classes. Press the Details button under the class list to verify your permissions.
Hold	Hold status of the submitted job. Permitted values are: **user** user hold **system** system hold** **usersys** user and system hold** Optional. The default is a no-hold state. You can press the **choice** button to get a list of available hold types.
Account Number	Number identifying the job. For use with the llacctmrg and llsummary commands for acquiring job accounting data. Optional. Required only if the **ACCT** keyword is set to **A_VALIDATE** in the configuration file.
Environment	Specifies your initial environment variables when your job starts. Separate environment specifications with semicolons. Optional.
Shell	The name of the shell to use for the job. Optional. If not specified, the shell used in the owner's password file entry is used. If none is specified, /**bin**/**sh** is used.
Group	The LoadLeveler group name to which the user belongs. Optional.
Step Name	The name of this job step. Optional.
Dependency	A Boolean expression defining the relationship between the job steps. Optional.

Field	Input
Comments	Comments associated with the job. These comments help to distinguish one job from another job. They are not incorporated into the job command file. Optional.

Note:

* If you specify *StartTime* but not *StartDate*, the default *StartDate* is the current day. If you specify *StartDate* but not *StartTime*, the default *StartTime* is 00:00:00. This means that the job will start as soon as possible on the specified date.

** Only the system administrator can specify this type of hold.

Note:

The fields that appear in this table are what you see when viewing the Build a Job window. The text in these fields do not necessarily correspond with the keywords listed in Appendix A, "LoadLeveler Job Command File Keywords and Examples" on page 61.

SELECT a Notification option

Your choices are:

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

SELECT a Checkpoint option

Your choices are:

No	Do not checkpoint the job. This is the default.
User	Yes, checkpoint the job. This type of checkpointing requires that you code the LoadLeveler **ckpt()** function in your program.
System	Yes, checkpoint the job. LoadLeveler automatically checkpoints the program.

Not checkpointing a job is the default so it does not have to be in the job command file.

Refer to "Checkpointing a Job Step" on page 6 for information on checkpointing a job.

SELECT a Restart option

Your choices are:

No	Do not restart the job.
Yes	Yes, restart the job. This is the default.

PRESS Requirements

▲ The Requirements dialog box appears.

Complete those fields for which you want to specify requirements. Defaults are used for those fields that you leave blank. For any requirements that you specify, LoadLeveler dispatches your job only to one of those machines with resources that match your requirements.

Field	Input
Architecture	Machine type. The job will not run on any other machine type. Optional. The default is the architecture of your current machine.
Operating System	Operating system. The job will not run on any other operating system. Optional. The default is the operating system of your current machine.
Disk	Quantity of disk space in the execute directory. The job will only run on a machine with at least this much disk space. Optional. The default is defined in your local configuration file.
Memory	Quantity of memory. The job will only run on a machine with at least this much memory. Optional. The default is defined in your local configuration file.
Machine(s)	Machine name(s). The job will only run on the specified machines. Optional
Feature(s)	Features. The job will only run on machines with specified features. Optional
Adapter	Adapter. Specifies the adapter on the SP system to use when processing a parallel job step. If you specify a value for this keyword, your job must run on the SP system. The default is Ethernet.
Pool	Pool. Specifies the number associated with the SP system parallel pool to use. If you specify a value for this keyword, your job must run on the SP system. The default is to select nodes from any pool.
Requirement	Requirements. The job will only run if the requirements are met.
Hint: If you type in a resource that is not available, you will NOT receive a message. LoadLeveler holds your job in the queue until the resource becomes available. Therefore, ensure the spelling of your entry is correct.	
Note: If you do not specify an architecture or operating system, LoadLeveler assumes that your job can run only on your machine's architecture and operating system. If your job is not a shell script that can be run successfully on any platform, you should specify a required architecture and operating system.	

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

PRESS Preferences

▲ The Preferences dialog box appears.

This dialog box, except for the last field, is the same as the Requirements dialog box. Complete the fields for those parameters that you want to specify. These parameters are not binding. For any preferences that you specify, LoadLeveler attempts to find a machine that matches these preferences along with your requirements. If it cannot find the machine, LoadLeveler chooses the first machine that matches the requirements.

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

PRESS Limits

▲ The Limits dialog box appears.

Complete the fields for those limits that you want to impose upon your job. If you type *copy* in any field, the limits in effect on the submit machine are used. If you leave any field blank, the default limits in effect for your userid on the machine that runs the job are used.

Field	Input
CPU Limit	Maximum amount of CPU time that the submitted job can use. Express the amount as: `[[hours:]minutes:]seconds[fraction]` For example, 12:56:21 is 12 hours, 56 minutes, and 21 seconds. Optional
Data Limit	Maximum amount of the data segment that the submitted job can use. Express the amount as: `integer{.fraction}{units}` where integer and fraction represent strings of up to eight digits. Optional
Core Limit	Maximum size of a core file. Optional
RSS Limit	Maximum size of the resident set size. It is the largest amount of physical memory a user's process can allocate. Optional
File Limit	Maximum size of a file that is created. Optional
Stack Limit	Maximum size of the stack. Optional
Job CPU Limit	Maximum amount of CPU a single job step can use per processor. Optional

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

PRESS Parallel Jobs

▲ The Parallel dialog box appears.

Complete those fields for which you want to specify requirements. Defaults are used for those fields that you leave blank.

Field	Input
Min # of Processors	Minimum number of processors required for running the parallel job. Optional. The default is one.
Max # of Processors	Maximum number of processors required for running the parallel job. Optional. The default is one.
Job Type	The type of job that you want to process. Press the **Choices** button to get the following list of job types from which to choose: • Serial • PVM3 • Parallel Optional. The default is Serial.
Parallel Path	The directory that defines where the PVM3 executables are located.

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

Step 2: Edit the Job Command File

There are several ways that you can edit the job command file that you just built:

1. Using the Jobs window:

   SELECT **File → Submit a Job**

   ▲ The Submit a Job dialog box appears.

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

   PRESS **Edit**

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

   If you have an icon manager, an icon may appear. An icon manager is a program that creates a graphic symbol, displayed on a screen, that you can point to with a device such as a mouse in order to select a particular function or application. Select this icon to view your job command file.

2. Using the **Tools Edit** pulldown menus on the Build a Job window:

   Using the Edit pulldown menu, you can modify the job command file. Your choices appear in the following table:

To:	Select
Add a step to the job command file	Add a Step
Delete a step from the job command file	Delete a Step
Clear the fields in the Build a Job window	Clear Fields
Select defaults to use in the fields	Set Field Defaults
Note: Other options include Go to Next Step, Go to Previous Step, and Go to Last Step that allow you to edit various steps in the job command file.	

   Using the **Tools** pulldown menu, you can modify the job command file. Your choices appear in the following table:

To:	Select
Name the job	Set Job Name
Open a window where you can enter a script file	Append Script
Fill in the fields using another file	Restore from File
View the job command file in a window	View Entire Job
Determine which step you are viewing	What is step #
Start a new job command file	Start a new job

To:	Do this:	
Save the information you entered into a file which you can submit later	**PRESS**	**Save**
		▲ A window appears prompting you to enter a job filename.
	TYPE IN	a job filename in the text entry field.
	PRESS	**OK**
		▲ The window closes and the information you entered is saved in the file you specified.
Run the program immediately and discard the information you entered	**PRESS**	**Submit**
	GO TO	Step 4

If you already submitted your job, go to "Step 4: Display, Refresh and Obtain Job Status." Otherwise, go to "Step 3: Submit a Job Command File."

Step 3: Submit a Job Command File

After building a job command file, you can submit it to one or more machines for processing. In addition to scripts with LoadLeveler keywords, you can also submit scripts that contain NQS options. You cannot, however, in this release of LoadLeveler, combine NQS and LoadLeveler options.

To submit a job, from the Jobs window:

SELECT **File → Submit a Job**

▲ The Submit a Job dialog box appears.

SELECT the job file that you want to submit from the file column.

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

PRESS **Submit**

▲ The job is submitted for processing.

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

Go to the next step.

Step 4: Display, Refresh and Obtain Job Status

When you submit a job, the status of the job is automatically displayed in the Jobs window. You can update or refresh this status using the Jobs window and selecting one of the following:

- **Refresh → Refresh Jobs**
- **Refresh → Refresh All**.

To change how often the amount of time should pass before the jobs window is automatically refreshed, use the Jobs window.

SELECT **Refresh → Set Auto Refresh**

▲ A window appears.

TYPE IN a value for the number of seconds to pass before the Jobs window is updated.

Automatic refresh can be expensive in terms of network usage and CPU cycles. You should specify a refresh interval of 120 seconds or more for normal use.

PRESS OK

▲ The window closes and the value you specified takes effect.

To receive detailed information on a job:

SELECT **Actions → Extended Status** to receive additional information on the job. Selecting this option is the same as typing **llq -x** command.

You can also get information in the following way:

SELECT **Actions → Extended Details**

Selecting this option is the same as typing **llq -x -l** command. You can also double click on the job in the Jobs window to get details on the job.

Note: Obtaining extended status or details on multiple jobs can be expensive in terms of network usage and CPU cycles.

SELECT **Actions → Job Status**

You can also use the **llq -s** command to determine why a submitted job remains in Idle state.

The following list defines the job status indicators and their definitions:

Indicator	Definition
I	the job step is in idle state.
R	the job step is running.
C	the job step has completed.
RM	the job step has been removed.
P	the job step is pending.
ST	the job step is starting.
NR	the job step is not running because dependencies have not been met.
H	the job step is in user hold state.
S	the job step is in system hold state.
HS	the job step is in user and system hold state.
D	the job step has been deferred.

For more information on these states, refer to the long listing generated by the **llq** command.

Go to the next step.

Step 5: Sort the Jobs Window

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

From the Jobs window:

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

Go to the next step.

Step 6: Change Priorities of Jobs in a Queue

If your job has not yet begun to run and is still in the queue, you can change the priority of the job in relation to your other jobs in the queue that belong to the same class. This only affects the user priority of the job. For more information on this priority, refer to “Setting and Changing the Priority of a Job” on page 10. Only the owner of a job or the LoadLeveler administrator can change the priority of a job.

From the Jobs window:

SELECT a job by clicking on it with the mouse

SELECT **Actions → Priority**

▲ A window appears.

TYPE IN a number between 0 and 100, inclusive, to indicate a new priority.

PRESS **OK**

▲ The window closes and the priority of your job changes.

Go to the next step.

Step 7: Hold a Job

Only the owner of a job or the LoadLeveler administrator can place a hold on a job.

From the Jobs window:

SELECT the job you want to hold by clicking on it with the mouse

SELECT **Actions → Hold**

▲ The job is put on hold and its status changes in the Jobs window.

Go to the next step.

Step 8: Release a Hold on a Job

Only the owner of a job or the LoadLeveler administrator can release a hold on a job.

From the Jobs window:

SELECT the job you want to release by clicking on it with the mouse

SELECT **Actions → Release from Hold**

▲ The job is released from hold and its status is updated in the Jobs window.

Go to the next step.

Step 9: Cancel a Job

Only the owner of a job or the LoadLeveler administrator can cancel a job.

From the Jobs window:

SELECT the job you want to cancel by clicking on it with the mouse

SELECT **Actions → Cancel**

▲ A warning dialog box appears prompting you to confirm your cancellation request. Once you confirm your request, LoadLeveler cancels the job and the job information disappears from the Jobs window.

Go to the next step.

Step 10: Display and Refresh Machine Status

The status of the machines is automatically displayed in the Machines window. You can update or refresh this status using the Machines window and selecting one of the following:

- **Refresh → Refresh Machines**
- **Refresh → Refresh All**.

To specify an amount of time to pass before the Machines window is automatically refreshed, from the Machines window:

SELECT **Refresh → Set Auto Refresh**

▲ A window appears.

TYPE IN a value for the number of seconds to pass before the Machines window is updated.

Automatic refresh can be expensive in terms of network usage and CPU cycles. You should specify a refresh interval of 120 seconds or more for normal use.

PRESS **OK**

▲ The window closes and the value you specified takes effect.

Go to the next step.

Step 11: Sort the Machines Window

You can specify a sorting option for the Machines window. The option you specify determines the order in which machines appear in the window.

From the Machines window:

To:	Select Sort →
Sort by machine name	**Sort by Name**
Sort by the machine which scheduled the job	**Sort by Schedd**
Sort by queue	**Sort by InQ**
Sort by active	**Sort by Act**
Sort by the machine which started the job	**Sort by Startd**
Sort by the number of jobs running	**Sort by Run**
Sort by load average	**Sort by LdAvg**
Sort by jobs that are idle	**Sort by Idle**
Sort by hardware architecture	**Sort by Arch**
Sort by operating system type	**Sort by OpSys**
Not specify a sort	**No Sort**

Go to the next step.

Step 12: Find the Location of the Central Manager

The LoadLeveler administrator designates one of the nodes in the LoadLeveler cluster as the central manager. When jobs are submitted at any node, the central manager is notified and decides where to schedule the jobs. In addition, it keeps track of the status of machines in the cluster and the jobs in the system by communicating with each node. LoadLeveler uses this information to make the scheduling decisions and to respond to queries.

To find the location of the central manager, from the Machines window:

SELECT Actions → Find Central Manager

▲ A message appears in the message window declaring on which machine the central manager is located.

Go to the next step.

Step 13: Find the Location of the Public Scheduling Machines

Public scheduling machines are those machines that participate in the scheduling of LoadLeveler jobs on behalf of the submit-only machines.

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

SELECT Actions → Find Public Scheduler

▲ A message appears displaying the address of these machines.

Go to the next step.

Step 14: Specify Which Jobs Appear in the Jobs Window

Normally, only your jobs appear in the Jobs window. You can, however, specify which jobs you want to appear by using the Select pull-down menu on the Jobs window.

To specify that all jobs:	Select Select →
In the queue appear	**All***
Belonging to a specific user or group of users appear	**By User***
Submitted on a specific machine or group of machines appear	**By Machine***
Having a particular id appear	**By Job Id** A dialog box prompts you to enter the id of the job you want to appear. This id appears in the left column of the Jobs window. Type in the id and press OK.
Note: * Following this choice, you can further qualify which jobs appear in the Jobs window by specifying only jobs running, jobs idle, or all jobs.	

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

Go to the next step.

Step 15: Specify Which Machines Appear in Machines Window

You can specify which machines will appear in the Machines window. The default is to view all of the machines in the LoadLeveler pool.

From the Machines window:

To:	Select Select →
View all of the machines	**All**
View machines by operating system	**by OpSys** ▲ A window appears prompting you to enter the operating system of those machines you want to view.
View machines by hardware architecture	**by Arch** ▲ A window appears prompting you to enter the hardware architecture of those machines you want to view.
View machines by state	**by State** ▲ A cascading pulldown menu appears prompting you to select the state of the machines that you want to view.

Go to the next step.

Step 16: Save LoadLeveler Messages in a File

Normally, all the messages that LoadLeveler generates appear in the Messages window. If you would also like to have these messages written to a file, use the Messages window.

SELECT **Actions → Start logging to a file**

▲ A window appears prompting you to enter a filename in which to log the messages.

TYPE IN the filename in the text entry field.

PRESS **OK**

▲ The window closes.

Chapter 3. Building and Submitting Jobs Using LoadLeveler Commands

This chapter explains how to build and submit a job to LoadLeveler using the LoadLeveler commands. For detailed descriptions of the commands, refer to Chapter 4, "LoadLeveler Commands" on page 39. In addition, you will learn how to perform other job related tasks. You can accomplish these same tasks by using the GUI. For information on using the GUI, refer to Chapter 2, "Building and Submitting Jobs Using the Graphical User Interface" on page 19.

Task Scenario Using the Commands

The tasks described in this chapter are presented in a typical step by step scenario that you, as a user, might be interested in accomplishing. You do not have to follow the steps shown here and may perform certain tasks before others without any difficulty. Some tasks must be performed prior to others in order for succeeding tasks to work. For example, you cannot submit a job before building a job.

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

To:	**Go to:**
Build a job	Step 1
Cancel a job	Step 9
Change the priorities of jobs in a queue	Step 5
Display the status of a job	Step 4
Display the status of a machine	Step 8
Edit a job	Step 2
Find the location of the central manager	Step 10
Find the location of the public scheduling machines	Step 11
Hold a job	Step 6
Release a hold on a job	Step 7
Submit a job command file	Step 3

For this task scenario, assume the following:

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

© Copyright IBM Corp. 1993, 1995

Step 1: Build a Job

Since you are not using the GUI, you have to build your job command file by using a text editor to create a script file. Into the file enter the name of the executable, other keywords designating such things as output locations for messages, and the necessary LoadLeveler statements. For example:

longjob.cmd

```
# This job command file is called longjob.cmd.  The
# executable is called longjob, the input file is longjob.in,
# the output file is longjob.out, and the error file is
# longjob.err.
#
# @ executable = longjob
# @ input      = longjob.in
# @ output     = longjob.out
# @ error      = longjob.err
# @ queue
```

Go to the next step.

Step 2: Edit a Job

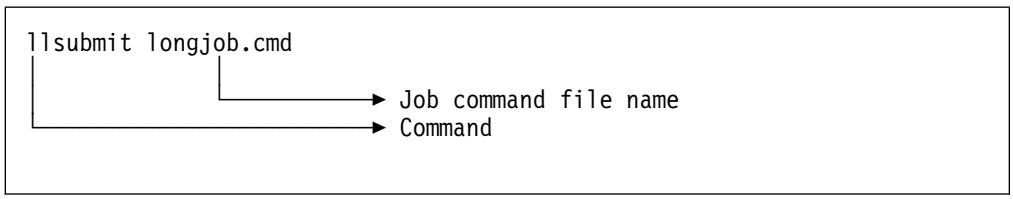
You can optionally edit the job command file you created in step 1.

Go to the next step.

Step 3: Submit a Job

To submit the job command file that you created in step 1, use the **llsubmit** command. See “llsubmit - Submit a Job” on page 59 for the command syntax.

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



LoadLeveler responds by issuing a message similar to:

```
submit: The job "wizard.22" has been submitted.
```

where *wizard* is the name of the machine to which the job was submitted and 22 is the job ID.

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

Step 4: Display the Status of a Job

To display the status of the job you just submitted, use the **llq** command. This command returns information about all jobs in the LoadLeveler queue. See “llq - Query Job Status” on page 48 for the command syntax.

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

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

Step 5: Change the Priorities of Jobs in the Queue

You can change the user priority of a job that is in the queue or one that is running. This only affects jobs belonging to the same user and that belong to the same class. If you change the priority of a job in the queue, the job's priority increases or decreases in relation to your other jobs in the queue. If you change the priority of a job that is running, it does not effect the job while it is running. It only affects the job if the job re-enters the queue to be dispatched again.

To change the priority of a job, use the **llprio** command. Refer to “llprio - Changes the Priority of Submitted Job Steps” on page 46 for the command syntax.

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

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

Step 6: Hold a Job

To place a temporary hold on a job in a queue, use the **llhold** command. See “llhold - Hold or Release a Submitted Job” on page 44 for the command syntax. This command only takes affect if jobs are in the queue or if they get suspended once they begin to run. You can only place a hold on jobs that you submitted.

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

```
llhold wizard.22.0
```

Step 7: Release a Hold on a Job

To release the hold you placed in step 6, use the **llhold** command. See “llhold - Hold or Release a Submitted Job” on page 44 for the command syntax.

To release the hold, type:

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

Step 8: Display the Status of a Machine

To display the status of the machine to which you submitted a job, use the **llstatus** command. See “llstatus - Query Machine Status” on page 54 for the command syntax.

To display the status, type:

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

For information on the long listing, refer to “llstatus - Query Machine Status” on page 54.

Step 9: Cancel a Job

To cancel *wizard.22.0*, use the **llcancel** command. See “ llcancel - Cancel a Submitted Job” on page 40 for the command syntax.

To cancel the job, type:

```
llcancel wizard.22.0
```

Step 10: Find the Location of the Central Manager

The **llstatus** command can also be used to display the machine on which the central manager is running.

Step 11: Find the Location of the Public Scheduling Machines

Public scheduling machines are those machines that participate in the scheduling of LoadLeveler jobs. The **llstatus** command can also be used to display the public scheduling machines.

Chapter 4. LoadLeveler Commands

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

Command	Purpose
llcancel	Cancels a submitted job.
llhold	Places a user or system hold on one or more jobs in the queue. Also, releases a hold.
llprio	Changes the priority of a submitted job step.
llq	Returns information about one or more LoadLeveler jobs.
llstatus	Returns status information about one or more machines in the pool.
llsubmit	Submits a job.

© Copyright IBM Corp. 1993, 1995

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.

Usage 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 adminstrator 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.

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.

Usage 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
```

Errors

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.

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.

Usage 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
```

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 preempt a job step that is running, even if its priority is lower than other jobs steps.

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.

Usage Notes

Priority changes resulting in a value less than 0 become 0.

Priority changes resulting in a value greater than 100 become 100.

Any change to a job step's priority applied by a user is relative only to *that user's other job steps*. 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 -x -s -l -v joblist -u userlist -h hostlist -c classlist
```

where

-H Provides a short usage message.

-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.

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. Only the owner of a job and the administrator can use this option.

If **-l** is *not* specified, then the standard listing is generated.

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

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. 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.

Usage Notes

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

The **-u** and **-h** options override the *jobname* 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
```

Output

Standard Listing

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

Id job identifier presented as *host.jobname.jobstep* number.

Owner userid of the job submitter.

Submitted date and time of job submission.

ST current job status, where:

R indicates job is running.

I indicates job idle. Job has been submitted, dispatched, vacated, or never run.

P indicates job pending. Job has been dispatched to a server but is not yet running.

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. This only occurs when multiple jobs have been submitted from a job command file or when the **llcancel** command is issued.

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 44.

H indicates job has been put in user hold. See "llhold - Hold or Release a Submitted Job" on page 44

SH indicates job has been put in system hold and user 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 46.

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 44
User Hold	indicates job has been put in user hold. See "llhold - Hold or Release a Submitted Job" on page 44.

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 of the jobs status categories.
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 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 is generated.
hostlist	Is a blank-delimited list of machines for which status is requested.

Usage Notes

If no *hostlist* is specified, all machines are queried.

If you have more than a few machines configured for LoadLeveler, consider redirecting the output to a file when using the **-l** flag.

Each machine periodically updates the central manager with a snapshot of its situation. Since the information returned by using **llstatus** is a collection of such snapshots, all taken at varying times, the total picture may not be completely consistent.

Examples

This example requests a long status listing for 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.

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
```

llsubmit

Appendix A. LoadLeveler Job Command File Keywords and Examples

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

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 LoadLeveler keyword or keyword statement is preceded by # @. These lines are treated as comments by the UNIX shell. The name of the executable code you want to submit is specified with the **executable** keyword.

  See “Job Command File Macros” on page 75 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. You can add comment lines to the file as you would in a shell script. Any line whose first non-blank character is a pound sign (#) and is not a LoadLeveler keyword statement is regarded as a comment.

You can build a job command file either by using the Build a Job window on the GUI or by using a text editor.

Job Command File Syntax

The following general rules apply to job command files.

- Comments begin with a #.
- Statements begin with a # @. Note that there can be any number of blanks between the # and the @.
- Statement components are separated by blanks. You can use blanks before or after other delimiters to improve readability but they are not required if another delimiter is used.
- Keywords are case insensitive. This means you can enter them in lower case, upper case, or mixed case.

© Copyright IBM Corp. 1993, 1995

LoadLeveler Keywords

account_no

Supports centralized accounting. Allows you to specify an account number to associate with a job. This account number is stored with job resource information in local and global history files. It may also be validated before LoadLeveler allows a job to be submitted.

The syntax is:

```
account_no = string
```

where `string` consists of a combination of numbers and letters. For example, if the job accounting group charges for job time based upon the department to which you belong, your account number would be similar to:

```
account_no = dept34ca
```

arguments

Specifies the list of arguments to pass to your program when your job runs.

The syntax is:

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

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

```
arguments = 5 8 9
```

checkpoint

Specifies whether you want to checkpoint your program. Specify **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. 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 “core_limit” on page 63 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. See “core_limit” on page 63 for an explanation of these values.

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` (as described in “step_name” on page 75) must be a previously defined job step and `operator` can be one of the following:

== equal to
!= not equal to

<=	less than or equal to
>=	greater than or equal to
<	less than
>	greater than
&&	and
||	or

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

CC_NOTRUN 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/pvm3/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. See "core_limit" on page 63 for an explanation of these values.

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.

image_size

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

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

The syntax is:

```
image_size = number
```

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

```
image_size = 11
```

initialdir

The pathname of the directory to use as the initial working directory during execution of the job step. If none is specified, the initial directory is the current working directory at the time you submitted the job. Filenames mentioned in the command file which do not begin with a / are relative to the initial directory. If you 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
```

input

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

The syntax is:

```
input = filename
```

For example:

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

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 an SP system to use when processing a parallel job step. If you specify a value for this keyword, your job must be run on an SP system. If you only specify this keyword in the first job step, its value is inherited 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 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 63 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 63 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 with 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 62.

$(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
```

Job Command File Examples

The following are examples of job command files.

Example 1: Submitting a Simple Job File

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

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

After you create the script file, you can submit the file to LoadLeveler using the **llsubmit** command. You must issue this command from your home directory.

```
llsubmit longjob.cmd
```

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

Example 2: Submitting Multiple Job Steps From a Single Script

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

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

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

```
llsubmit longjob.cmd
```

LoadLeveler responds by issuing the following:

```
submit: The job "ll6.4" with 2 job steps has been submitted.
```

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

Example 3: Generating Multiple Jobs With Varying Outputs

To run a program several times, varying the initial conditions each time, you could create multiple LoadLeveler scripts, each specifying a different input and output file as described previously in Example 2. It would probably be more convenient to prepare different input files and submit the job only once, letting LoadLeveler generate the output files and do the multiple submissions for you.

This example illustrates the following:

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

Assume that you created five input files and each input file has different initial conditions for the program. Note that the job step id starts at 0 and increases by one for each job step listed in the job command file.

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

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

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

Now, type the command:

```
llsubmit longjob.cmd
```

LoadLeveler responds by issuing the following:

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

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

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

Example 4: Using the Job Command File as the Executable

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

This example illustrates the following:

- The name of the script is *long.sh*. You can name your LoadLeveler script anything.

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

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

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

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

Type in the command:

```
llsubmit long.sh
```

LoadLeveler responds by issuing the following:

```
submit: The job "wizard.4" has been submitted.
```

Example 5: Building and Submitting a Parallel Job Command File

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

To construct a parallel job command file, specify the number of machines the parallel job requires to be processed. Use the keywords **min_processors** and **max_processors** to specify a range of processors needed. In addition, you could use the **job_type** keyword to specify the type of parallel job you want to run. You can also specify, using the **requirements** keyword, the type of adapter and parallel pool you want to use. You should only specify the adapter and parallel pool if you

are submitting your parallel job to the SP system. For more information, refer to chapter 4 of the LoadLeveler Administration Guide.

Example 6: Parallel Job Command Files

The following job command files walk you through the process of submitting a parallel job. These job command files assume the following:

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

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 4. 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 4, 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 4, 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 4 on page 80, what your job command file might look like when using PVM 3.3:

```
# @ executable    = my_parallel_job
# @ job_type      = pvm3
# @ parallel_path = /home/userid/cmds/pvm3/$PVM_ARCH:$PVM_ROOT/lib/$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.
- The job must have a *job_type* of **pvm3**.

Appendix B. Graphical User Interface Information

This appendix provides some introductory information on graphical user interfaces. This chapter provides neither complete nor detailed instructions on using either the LoadLeveler user interface or any other graphical user interface.

This appendix also discusses how to customize your graphical user interface by modifying the **Xloadl** and **Xloadl_so** files and provides a discussion of the **skel.cmd** file.

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

What is a Window?

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

What are the Types of Windows?

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

Primary Windows

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

Secondary Windows

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

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

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

© Copyright IBM Corp. 1993, 1995

What is the Layout of a Window?

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

What is a Dialog Box?

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

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

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

How Do I Enter Data in the Interface?

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

Text entry fields

A field you can select and type directly into.

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

What is a Menu Bar?

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

What is a Pull-down Menu?

A pull-down menu is a menu that emerges in a downward direction from a point or line at or near the top of the screen. When you select an item from a menu bar, a pull-down menu appears. You can select an item from the pull-down menu to carry out an action or to bring up another pull-down menu originating from the first one.

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

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

What is a Scroll Bar?

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

Customizing the Graphical User Interface

LoadLeveler provides an installation with two types of graphical user interfaces. One interface is for use by LoadLeveler users whose machines are completely interacting with LoadLeveler. The second interface is available to users whose machines are only participating on a limited basis. This second type of machine is called a submit-only machine. A complete discussion of the different machine types appears in chapter 1 of the LoadLeveler Administration and Installation Guide.

You can customize the GUI by modifying one of the appropriate files:

Xloadl for fully participating machines

Xloadl_so for submit-only machines

Both of these files are located in the **/usr/lib/Xll/app-defaults** directory. These files contain the default values for the graphical user interface. Following is a brief description of the syntax of these files. Most of the resource names in the files are common to all AIX or UNIX Xloadl files. To customize for your individual needs, copy the resources to be changed from the file into your **.Xdefaults** file and edit it.

Syntax of an Xloadl File

- Comments begin with !
- Resource variables may begin with *
- Colons follow resource variables
- Resource variable values follow colons.

Modifying Windows and Buttons

All of the windows and buttons that are part of the GUI have certain characteristics in common. For example, they all have a foreground and background color, as well as a size and a location. Each one of these characteristics is represented by a resource variable. For example, the foreground characteristic is represented by the resource variable **foreground**. In addition, every resource variable has a value associated with it. The values of the resource variable **foreground** are a range of colors.

Before customizing a window, you need to locate the resource variables associated with the desired window. To do this, search for the window identifier in your Xloadl file. The following table lists the windows and their respective identifiers:

Window	Identifier
Jobs	job_status
Machines	machine_status
Messages	message_area
Build a Job	builder
Submit a Job	submit
Requirements	requirements
Preferences	preferences
Limits	limits
Account Report Data	reporter
Parallel	parallel
Script	script

The following table lists the resource variables for all the windows and the buttons along with a description of each resource variable. Use the information in this table to modify your graphical user interface by changing the values of desired resource variables. The values of these resource variables depend upon Motif requirements.

Resource Variable	Description
geometry	The location of the object.
foreground	The foreground color of the object.
background	The background color of the object.
width	The width of the object.
height	The height of the object.
label.string	The text associated with the object.

Modifying Help Panels

Help panels have the same characteristics as all of the windows plus a few unique ones:

Resource Variable	Values	Description
help*work_area.width	Any integer*	The width of the help panel.
help*work_area.height	Any integer*	The height of the help panel.
help*scrollHorizontal	[true|false] The default is False.	Sets the scrolling option on or off.
help*wordWrap	[true|false] The default is True.	Sets word wrapping on or off.
Note: * The work area and height depend upon your screen limitations.		

Modifying the Standard X Resources

Resource Variable	Values	Description
iconic	true | false	Specifies whether LoadLeveler comes up as an icon. The initial setting is false.
highlightOnEnter	true | false	Determines whether to draw the widget's highlighting rectangle whenever the cursor moves into a widget. Applies only when the shell has a focus policy of pointer. It true, highlighting is drawn. If false, highlighting is not drawn. The initial setting is false.
keyboardFocusPolicy	pointer | explicit	Defines how the mouse and windows interact. If set to **pointer**, the window the mouse is currently in will be the window that is selected for a keyboard or mouse button action. If set to **explicit**, click mouse button 1 in the window to move input focus to that window.

Resource Variable	Values	Description
fontlist	adobe courier bold r normal 14 140 75 75 m 90 iso8859 1	Fonts used for displaying text.
highlightThickness	An integer	Thickness of the highlighting rectangle. Initial value is 0.

Modifying Application Defined Resources

Resource Variable	Values	Description
editCommand	Available editors	Command to use when you select “edit” from the Submit window.
job filter	Current filters are: running_by_user, idle_by_user, all_by_user, running_by_host, idle_by_host, all_by_host, all_running, all_idle, and all_jobs.	If you select any of the “by_user” filters, you may also specify which user in the “job_select_name” resource. If you select any of the “by_host” filters, you may also specify which host in the “job_select_machine” resource.
job_refresh	Any integer.	Specifies the length of time between automatic updates of the Jobs Window. A recommended value is 120.
job_select_machine	Any valid hostname.	Host name to use for job filtering. When job filter is set as “allmachine”, only jobs from this machine are shown.
job_select_name	Any valid userid. The default is your userid.	User name to use for job filtering. When job filter is set as “alluser”, only jobs from this user are shown. If job_select_user is not specified, the default is the userid from which xloadl was started.
job_sort	Current sort is: name time usage priority machine	Specifies how the jobs are sorted in the Jobs window.
machine_refresh	Integer. The default is 60 seconds.	Specifies the length of time between automatic updates of the Machines window.
machine sort	Current sort is: name running total state ldavg idle arch opsys.	Specifies how the machines are sorted in the Machines window.
prompt	[true|false] The default is True.	Specifies whether you are prompted for confirmation of certain actions.

Resource Variable	Values	Description
logging	[true|false] The default is false.	Specifies whether to automatically log messages to a file.
logfile	Characters and/or Integers. The default file is `xloadl.log`	Specifies the file name to which messages are logged if logging is set to true.
skelfile	Characters and/or Integers	Name of the file to use as a template for the Job Builder window.
snapfile	Characters and/or Integers. The default file is `xloadl.snapshot`	Name of the file to use when requesting a snapshot of the log.

Modifying the Titles of the Main Windows

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

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

Modifying the Labels in the Build a Job Window

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

Resource Variable	Current Value
builder*Tools_Set_Name.labelString	Set Job Name
builder*Tools_Script.labelString	Append Script
builder*Tools_Restore.labelString	Restore from File
builder*Tools_View_Job.labelString	View Entire Job
builder*Tools_What_Num.labelString	What is Step #
builder*Edit_Add.labelString	Add a Step
builder*Edit_AddFirst.labelString	Add a First Step
builder*Edit_Delete.labelString	Delete a Step
builder*Edit_Clear.labelString	Clear Fields
builder*Edit_Defaults.labelString	Set Field Defaults
builder*Edit_Next.labelString	Go to Next Step
builder*Edit_Previous.labelString	Go to Previous Step
builder*Edit_Get.labelString	Go to Step ...
builder*Edit_Last.labelString	Go to Last Step
builder*work_area*AccountNo.labelString	Account Number

Resource Variable	Current Value
builder*work_area*Notify_Always.labelString	Always
builder*work_area*Arguments.labelString	Arguments
builder*work_area*Class.labelString	Class
builder*work_area*Checkpoint.labelString	Checkpoint
builder*work_area*Class_Choice.labelString	Choices
builder*work_area*Hold_Choice.labelString	Choices
builder*action_area*Clear.labelString	Clear
builder*action_area*Close.labelString	Close
builder*work_area*Comment.labelString	Comments
builder*work_area*Notify_Complete.labelString	Complete
builder*action_area*Defaults.labelString	Defaults
builder*work_area*Depend.labelString	Dependency
builder*work_area*Environment.labelString	Environment
builder*work_area*Notify_Error.labelString	Error
builder*work_area*Executable.labelString	Executable
builder*work_area*Group.labelString	Group
builder*action_area*Help.labelString	Help
builder*work_area*Hold.labelString	Hold
builder*work_area*Workdir.labelString	Initialdir
builder*work_area*ImageSize.labelString	Image Size
builder*work_area*Limits.labelString	Limits
builder*work_area*Syscalls_Local.labelString	Local
builder*work_area*Notify_Never.labelString	Never
builder*work_area*Checkpoint_No.labelString	No
builder*work_area*Restart_No.labelString	No
builder*work_area*Notification.labelString	Notification
builder*work_area*NotifyUser.labelString	Notify user
builder*work_area*Parallel.labelString	Parallel Jobs
builder*work_area*Preferences.labelString	Preferences
builder*work_area*Priority.labelString	Priority
builder*work_area*Syscalls_Remote.labelString	Remote
builder*work_area*Requirements.labelString	Requirements
builder*work_area*Restart.labelString	Restart
builder*action_area*Save.labelString	Save
builder*work_area*Shell.labelString	Shell
builder*work_area*Notify_Start.labelString	Start
builder*work_area*StartDate.labelString	Start Date mm/dd/yy
builder*work_area*StartTime.labelString	Start Time hh:mm:ss.
builder*work_area*Stdin.labelString	Stdin

Resource Variable	Current Value
builder*work_area*Stdout.labelString	Stdout
builder*work_area*Stderr.labelString	Stderr
builder*work_area*StepName.labelString	Step Name
builder*action_area*Submit.labelString	Submit
builder*work_area*Syscalls.labelString	Syscalls
builder*work_area*Checkpoint_system_initiated.labelString	System
builder*work_area*Checkpoint_user_initiated.labelString	User
builder*work_area*Checkpoint_Yes.labelString	Yes
builder*work_area*Restart_Yes.labelString	Yes

Modifying the Labels in the Requirements Window

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

Resource Variable	Current Value
requirements*work_area*Architecture.labelString	Architecture
requirements*work_area*OpSys.labelString	Operating Sys
requirements*work_area*Disk.labelString	Disk
requirements*work_area*Memory.labelString	Memory
requirements*work_area*Machines.labelString	Machines
requirements*work_area*Features.labelString	Features
requirements*work_area*Adapter.labelString	Adapter
requirements*work_area*Pool.labelString	Pool
builder*requirements*action_area*Help.labelString	Help
builder*requirements*action_area*Close.labelString	Close
builder*requirements*Req_Adapter_Choice.labelString	Choices
builder*requirements*Adapter*Selections.labelString	Adapter Type Choices

Modifying the Labels in the Preferences Window

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

Resource Variable	Current Value
preferences*work_area*Architecture.labelString	Architecture
preferences*work_area*OpSys.labelString	Operating Sys
preferences*work_area*Disk.labelString	Disk
preferences*work_area*Memory.labelString	Memory
preferences*work_area*Machines.labelString	Machines
preferences*work_area*Features.labelString	Features
preferences*work_area*Adapter.labelString	Adapter
preferences*work_area*Pool.labelString	Pool
builder*preferences*action_area*Help.labelString	Help
builder*preferences*action_area*Close.labelString	Close
builder*preferences*Prf_Adapter_Choice.labelString	Choices
builder*preferences*Adapter*Selections.labelString	Adapter Type Choices

Modifying the Labels in the Limits Window

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

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

Modifying the Labels in the Parallel Jobs Window

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

Resource Variable	Current Value
parallel*work_area*min_processors.labelString	Min # of Processors
parallel*work_area*max_processors.labelString	Max # of Processors
parallel*work_area*job_type.labelString	Job Type
parallel*work_area*parallel_path.labelString	Parallel Path
builder*parallel*action_area*Help.labelString	Help
builder*parallel*action_area*Close.labelString	Close
builder*parallel*Job_Type_Choice.labelString	Choices
builder*parallel*Selections.labelString	Job Type Choices

Modifying the Labels in the Create Account Report Window

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

Resource Variable	Current Value
reporter.geometry	745x460+103+150
reporter*background	SteelBlue
reporter*work_area*background	SkyBlue
reporter*work_area*XmText*background	lightsteelblue
reporter*work_area*XmText*foreground	black
reporter*action_area*background	SkyBlue
reporter*action_area*foreground	black
reporter*work_area*Format.labelString	Format
reporter*work_area*Start.labelString	Start date
reporter*work_area*StartTo.labelString	To
reporter*work_area*End.labelString	End date
reporter*work_area*EndTo.labelString	To
reporter*work_area*Short.labelString	Short
reporter*work_area*Long.labelString	Long
reporter*work_area*Extended.labelString	Extended
reporter*work_area*User.labelString	User ID
reporter*work_area*Class.labelString	Class
reporter*work_area*LoadlGroup.labelString	LoadL Group
reporter*work_area*UnixGroup.labelString	UNIX Group
reporter*work_area*Host.labelString	Allocated Host
reporter*work_area*Jobid.labelString	Job ID
reporter*work_area*Type.labelString	Report Type
reporter*work_area*Section.labelString	Section
reporter*work_area*Input.labelString	Input data file
reporter*work_area*Output.labelString	Output to file

Modifying the Labels in the List of Classes Window

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

Resource Variable	Current Value
class_select*applyLabelString	Details
class_select*okLabelString	OK
class_select*helpLabelString	Help
class_select*cancelLabelString	Cancel

Modifying the Labels in the Script Window

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

Resource Variable	Current Value
script*action_area*Help.labelString	Help
script*action_area*Close.labelString	Close
script*action_area*Ok.labelString	Save
script*action_area*Get.labelString	Get
script*action_area*label.labelString	Job Script Contents

Modifying the Labels in the Submit a Job Window

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

Resource Variable	Current Value
submit*Edit.labelString	Edit file
submit*Shell.labelString	Open a Shell
submit*cancelLabelString	Close
submit*applyLabelString	Filter
submit*okLabelString	Submit
submit*helpLabelString	Help

Modifying the Jobs Window Pull-Down Menu Bar Labels

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

Resource Variable	Current Value
The following resource variables are for the File pull-down menu	
job_status*menubar*File.labelString	File
job_status*File_Build_Job.labelString	Build a Job...
job_status*File_Submit_Job.labelString	Submit a Job...
job_status*File_sep.labelString	sep
job_status*File_Exit.labelString	Exit
The following resource variables are for the Actions pull-down menu	
job_status*menubar*Actions.labelString	Actions
job_status*Actions_Cancel.labelString	Cancel

Resource Variable	Current Value
job_status*Actions_Priority.labelString **This is not part of the submit-only GUI.**	Priority
job_status*Actions_Hold.labelString **This is not part of the submit-only GUI.**	Hold
job_status*Actions_Release.labelString **This is not part of the submit-only GUI.**	Release From Hold
job_status*Actions_Details.labelString	Details
job_status*Actions_ExDetails.labelString	Extended Details
job_status*Actions_ExStatus.labelString	Extended Status
job_status*Actions_JobStatus.labelString	Job Status Report
job_status*Actions_sep.labelString	sep
job_status*Actions_Select_All.labelString	Select All
job_status*Actions_Deselect_All.labelString	Deselect All
The following resource variables are for the Refresh pull-down menu	
job_status*menubar*Refresh.labelString	Refresh
job_status*Refresh_Jobs.labelString	Refresh Jobs
job_status*Refresh_Auto.labelString	Set Auto Refresh...
job_status*Refresh_sep.labelString	sep
job_status*Refresh_All.labelString	Refresh All
The following resource variables are for the Sort pull-down menu	
job_status*menubar*Sort.labelString	Sort
job_status*menubar*Sort_by_Machine.labelString	Sort by Submitting Machine
job_status*Sort_by_Name.labelString	Sort by Owner
job_status*Sort_by_Class.labelString	Sort by Class
job_status*Sort_by_Submission_Time.labelString	Sort by Submission Time
job_status*menubar*Sort_by_State.labelString	Sort by State
job_status*Sort_by_Priority.labelString	Sort by Priority
job_status*Sort_by_Class.labelString	Sort by Class
job_status*Sort_by_Run_Machine.labelString	Sort by Running Machine
job_status*Sort_by_Dispatch_Order.labelString	Sort by Dispatch Order
The following resource variables are for the Select pull-down menu	
job_status*menubar*Select.labelString	Select
job_status*Select_All.labelString	All
job_status*Select_All_Running.labelString	Jobs Running
job_status*Select_All_Idle.labelString	Jobs Idle
job_status*Select_All_Jobs.labelString	All Jobs
job_status*Select_User.labelString	By User
job_status*Select_User_Running.labelString	Jobs Running...
job_status*Select_User_Idle.labelString	Jobs Idle...

Resource Variable	Current Value
job_status*Select_User_Jobs.labelString	All Jobs...
job_status*Select_Machines.labelString	By Machine
job_status*Select_Machines_Running.labelString	Running Jobs Submitted To...
job_status*Select_Machines_Idle.labelString	Idle Jobs Submitted To...
job_status*Select_Machines_All.labelString	All Jobs Submitted To...
job_status*Select_Job_Id.labelString	By Job Id...
job_status*Select_Show.labelString	Show Selection
The following resource variables are for the Admin pull-down menu These are not part of the submit-only GUI.	
job_status*menubar*Admin.labelString	Admin
job_status*Admin_Favor_User.labelString	Favor User...
job_status*Admin_End_Favor_User.labelString	UnFavor User...
job_status*Admin_Favor_Job.labelString	Favor Job
job_status*Admin_End_Favor_Job.labelString	UnFavor Job
job_status*Admin_Set_SysHold.labelString	Syshold
job_status*Admin_Release.labelString	Release From Hold
The following resource variables are for the Help pull-down menu	
job_status*menubar*Help.labelString	Help
job_status*Help_File.labelString	File
job_status*Help_Actions.labelString **This is not part of the submit-only GUI.**	Actions
job_status*Help_Overview.labelString	Overview
job_status*Help_Windows.labelString	Windows
job_status*Help_Refresh.labelString	Refresh
job_status*Help_Sort.labelString	Sort
job_status*Help_Select.labelString	Select
job_status*Help_Admin.labelString **This is not part of the submit-only GUI.**	Admin

Modifying the Machines Window Pull-Down Menu Bar Labels

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

Resource Variable	Current Value
The following resource variables are for the File pull-down menu	
machine_status.menubar.File.labelString	File
machine_status*File_Build_Job.labelString	Build a Job...

Resource Variable	Current Value
machine_status*File_Submit_Job.labelString	Submit a Job...
machine_status*File_sep.labelString	sep
machine_status*File_Exit.labelString	Exit
The following resource variables are for the Actions pull-down menu These are not part of the submit-only GUI.	
machine_status*menubar*Actions.labelString	Actions
machine_status*Actions_Details.labelString	Details
machine_status*Actions_Find_Central_Manager.labelString	Find Central Manager
machine_status*Actions_Find_LoadL_Clients.labelString	Find Public Scheduler
machine_status*Actions_sep.labelString	sep
machine_status*Actions_SelectAll.labelString	Select All
machine_status*Actions_DeselectAll.labelString	Deselect All
The following resource variables are for the Refresh pull-down menu	
machine_status*menubar*Refresh.labelString	Refresh
machine_status*Refresh_Machines.labelString	Refresh Machines
machine_status*Refresh_Auto.labelString	Auto Refresh...
machine_status*Refresh_sep.labelString	sep
machine_status*Refresh_All.labelString	Refresh All
The following resource variables are for the Sort pull-down menu	
machine_status*menubar*Sort.labelString	Sort
machine_status*Sort_by_Name.labelString	Sort by Name
machine_status*Sort_by_Schedd.labelString	Sort by Schedd
machine_status*Sort_by_Queue.labelString	Sort by InQ
machine_status*Sort_by_Active.labelString	Sort by Act
machine_status*Sort_by_Startd.labelString	Sort by Startd
machine_status*Sort_by_Run.labelString	Sort by Run
machine_status*Sort_by_Total.labelString	Sort by Total
machine_status*Sort_by_State.labelString	Sort by State
machine_status*Sort_by_LdAvg.labelString	Sort by LdAvg
machine_status*Sort_by_Idle.labelString	Sort by Idle
machine_status*Sort_by_Arch.labelString	Sort by Arch
machine_status*Sort_by_OpSys.labelString	Sort by OpSys
machine_status*Sort_NoSort.labelString	No Sort
The following resource variables are for the Select pull-down menu	
machine_status*menubar*Select.labelString	Select
machine_status*Select_All.labelString	All
machine_status*Select_Op_Sys.labelString	By Op Sys
machine_status*Select_Arch.labelString	By Arch
machine_status*Select_State.labelString	By State

Resource Variable	Current Value
machine_status*Select_Show.labelString	Show Selection
machine_status*Select_State_Down.labelString	Down
machine_status*Select_State_Not_Down.labelString	Not Down
machine_status*Select_State_Flush.labelString	Flush
machine_status*Select_State_Suspend.labelString	Suspend
machine_status*Select_State_Drain.labelString	Drain
machine_status*Select_State_Admin_State.labelString	Admin State
machine_status*Select_State_Not_Admin.labelString	Not Admin State
The following resource variables are for the Admin pull-down menu These are not part of the submit-only GUI.	
machine_status*menubar*Admin.labelString	Admin
machine_status*menubar*Admin_StartAll.labelString	Start All
machine_status*Admin_Start.labelString	Start LoadLeveler
machine_status*Admin_Stop.labelString	Stop LoadLeveler
machine_status*Admin_sep.labelString	sep
machine_status*Admin_reConfig.labelString	reConfig
machine_status*Admin_Recycle.labelString	Recycle
machine_status*Admin_Drain.labelString	Drain
machine_status*Admin_Flush.labelString	Flush
machine_status*Admin_Suspend.labelString	Suspend
machine_status*Admin_Resume.labelString	Resume
machine_status*Admin_Capture.labelString	Capture Data
machine_status*Admin_Collect.labelString	Collect Account Data
machine_status*Admin_Report.labelString	Collect Account Report
machine_status*Admin_Version.labelString	Version
The following resource variables are for the drain and resume options	
machine_status*Admin_Drain_Daemons.labelString	Daemons
machine_status*Admin_Drain_Schedd.labelString	Schedd
machine_status*Admin_Drain_Startd.labelString	Startd
machine_status*Admin_Resume_Daemons.labelString	Daemons
machine_status*Admin_Resume_Schedd.labelString	Schedd
machine_status*Admin_Resume_Startd.labelString	Startd
The following resource variables are for the Help pull-down menu	
machine_status*menubar*Help.labelString	Help
machine_status*Help_File.labelString	File
machine_status*Help_Overview.labelString	Overview
machine_status*Help_Actions.labelString	Actions
machine_status*Help_Refresh.labelString	Refresh
machine_status*Help_Sort.labelString	Sort

Resource Variable	Current Value
machine_status*Help_Admin.labelString **This is not part of the submit-only GUI.**	Admin

Modifying the Messages Window Pull-Down Menu Bar Labels

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

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

Customizing the skel.cmd File

The **skel.cmd** file contains the default entries to use when viewing the Build a Job window. When you bring up the Build a Job window, any defaults specified in the **skel.cmd** file are filled in the fields in the Build a Job window. In addition to being able to change the name of this file, you can modify the contents of the file to alter the defaults. The following is an example of a **skel.cmd** file:

```
#!/bin/sh
#@ output = $(Executable).$(jobid).out
#@ error = $(Executable).$(jobid).err
#@ notification = error
#@ checkpoint = no
#@ restart = no
#@ queue
```

You can change the fields in the Build a Job window by typing into the fields, by modifying the **skel.cmd** file, or by specifying a new job command file using the Restore a File option from the pulldown.

If your job command file has multiple job steps, the Build a Job window is filled in (using the **skel.cmd** file) based upon the first job step identified only. The other steps are ignored.

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

© Copyright IBM Corp. 1993, 1995

A

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.

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.

node. In a network, the point where one or more functional units interconnect transmission lines. A computer location defined in a network.

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 are Bourne, C and Korn.

System Administrator. The user who is responsible for setting up, modifying, and maintaining LoadLeveler.

W

working directory. All files without a fully qualified path name are relative to this directory.

workstation. *(1) A configuration of input/output equipment at which an operator works. *(2) A terminal or microcomputer, usually one that is connected to a mainframe or to a network, at which a user can perform applications.

Index

A
account_no 62
accounting 62
adapter 72
arguments 62

B
building jobs 3, 20

C
cancelling jobs 14, 30, 38
  using the command llcancel 38
  using the GUI 30
central manager 31, 38
changing job priority 29, 37
  using the command llprio 37
  using the GUI 29
changing job step priority
  using llprio 46
checkpoint 5, 62
  system_initiated checkpointing 5, 62
  user_initiated checkpointing 5, 62
choice button 23
class 62
class information 42
comment 63
comment statements 3
core_limit 63
cpu_limit 63

D
data_limit 64
dependency 64
dialog boxes 16, 84
displaying 30, 31
  central manager 31
  machine status 30
  public submit machines 31
displaying job status 27, 37
  job status 27
  using the command llq 37
  using the GUI 27
displaying machine status 38
  using the command llstatus 38

E
editing jobs 5, 25
environment 66
error 66
executable 2, 66

F
file_limit 67

G
graphical user interface 15, 17, 83
  customizing 17
  dialog boxes 84
  menu bar 85
  pull-down menus 85
  scroll bars 85
  skel.cmd file 100
  windows 83
  Xloadl 85
  Xloadl_so 85
group 67

H
help 17
hold 67
holding jobs 13, 29, 37
  using the command llhold 37
  using the GUI 29

I
image_size 68
initialdir 68
input 68

J
job command file 3
job status 27, 48
job steps 2
job_cpu_limit 69
job_name 69
job_type 69
jobs 20, 25, 27, 29, 30
  building 20
  cancelling 30
  changing job priority 29
  displaying job status 27
  editing 25
  holding 29
  releasing a hold 29
  submitting 27

© Copyright IBM Corp. 1993, 1995

K

keywords 3, 61
  arguments 62
  checkpoint 62
  class 62
  comment 63
  core_limit 63
  cpu_limit 63
  data_limit 64
  dependency 64
  environment 66
  error 66
  executable 66
  file_limit 67
  group 67
  hold 67
  image_size 68
  initialdir 68
  input 68
  job_cpu_limit 69
  job_name 69
  job_type 69
  max_processors 69
  min_processors 70
  notification 70
  notify_user 70
  output 70
  parallel_path 71
  preferences 71
  queue 71
  requirements 71
  restart 73
  rss_limit 73
  shell 74
  stack_limit 74
  startdate 74
  step_name 74
  user_priority 75

L

llcancel 40
llclass 42
llhold 44
llprio 46
llq 48
llstatus 54
llsubmit 59

M

machine status 54
max_processors 69
menu bar 17, 85
messages 32
min_processors 70

N

notification 70
notify_user 70

O

output 70

P

parallel job command files 5
parallel_path 71
pool 24, 73
preferences 71
primary windows 83
priority 9, 46
public scheduling machines 14, 38
pull-down menus 17, 85

Q

query a job 13, 28
  llq command 48
  using the GUI 28
query class information
  llclass command 42
queue 71

R

requirements 71
restart 73
rss_limit 73

S

scroll bars 85
secondary windows 16, 83
serial job command files 4
shell 23, 74
  pool 24
skel.cmd file 100
stack_limit 74
startdate 74
status 54
step_name 74
submit-only machine 12, 13, 14
submitting jobs 11, 27, 36, 59
  using the command llsubmit 36
  using the GUI 27
system_initiated checkpointing 5, 62

U

user interface 20
user_initiated checkpointing 5, 62
user_priority 75

W

windows 15, 16, 83, 84
  layout of 84
  types of
    primary 83
    secondary 16, 83

X

Xloadl 85
Xloadl_so 85

Communicating Your Comments to IBM

LoadLeveler User's Guide
Release 2.1

Publication No. SH26-7226-03

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 (United States customers only): KGNVMC(MHVRCFS)
  - IBM Mail Exchange: USIB6TC9 at IBMMAIL
  - Internet: mhvrcfs@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

LoadLeveler User's Guide
Release 2.1

Publication No. SH26-7226-03

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
SH26-7226-03

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

IBM 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.

SH26-7226-03