IBM LoadLeveler
Version 5 Release 1

# *Resource Manager*

SC23-6790-04

IBM LoadLeveler
Version 5 Release 1

# Resource Manager

SC23-6790-04

**Note**

Before using this information and the product it supports, read the information in "Notices" on page 335.

This edition applies to version 5, release 1, modification 0 of IBM LoadLeveler (product numbers 5725-G01, 5641-LL1, 5641-LL3, 5765-L50, and 5765-LLP) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC23-6790-03.

# Contents

# Figures

# Tables

# About this information

**Attention:**

For LoadLeveler® Version 5 Release 1, changes apply to Linux.

**Disclaimer:**

The functions or features found herein may not be available on all operating systems or platforms and do not indicate the availability of these functions or features within the IBM® product or future versions of the IBM product. The development, release, and timing of any future features or functionality is at IBM's sole discretion. IBM's plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. The information mentioned is not a commitment, promise, or legal obligation to deliver any material, code or functionality. The information may not be incorporated into any contract and it should not be relied on in making a purchasing decision.

**Release notes:**

For the latest release notes, go to the Fix Central website (http://www-933.ibm.com/support/fixcentral/?productGroup0=ibm/fcpower&productGroup1=ibm/ClusterSoftware&productGroup2=ibm/power/IBM). For more information about Fix Central, see "Locating LoadLeveler software fixes and updates in Fix Central" on page xiii.

This information describes how to configure and administer the IBM LoadLeveler in a cluster environment.

## Who should use this information

Personnel who are responsible for installing, configuring and managing the cluster environment. These people are called administrators. administrative tasks include:
- Setting up configuration and administration files
- Maintaining the resource manager component

Both administrators and general users should be experienced with the UNIX commands. Administrators also should be familiar with:
- Cluster system management techniques such as SMIT, as it is used in the AIX® environment
- Networking and NFS or AFS® protocols

## Conventions and terminology used in this information

Throughout the IBM LoadLeveler product information:
- LoadLeveler for Linux on x86 Architecture includes:
  - IBM System x® Intelligent Cluster™
  - IBM System servers with Advanced Micro Devices (AMD) Opteron or Intel® Extended Memory 64 Technology (EM64T) processors
- References to Schedd are also referred to as job manager.

*Table 1. Conventions*

| Convention | Usage |
|---|---|
| **bold** | **Bold** words or characters represent system elements that you must use literally, such as commands, flags, path names, directories, file names, values, and selected menu options. |
| **bold underlined** | **Bold underlined** keywords are defaults. These take effect if you do not specify a different keyword. |
| `constant width` | Examples and information that the system displays appear in `constant-width` typeface. |
| *italic* | *Italic* words or characters represent variable values that you must supply.<br><br>*Italics* are also used for information unit titles, for the first use of a glossary term, and for general emphasis in text. |
| <*key*> | Angle brackets (less-than and greater-than) enclose the name of a key on the keyboard. For example, <**Enter**> refers to the key on your terminal or workstation that is labeled with the word *Enter*. |
| \ | In command examples, a backslash indicates that the command or coding example continues on the next line. For example:<br>`mkcondition -r IBM.FileSystem -e "PercentTotUsed > 90" \`<br>`-E "PercentTotUsed < 85" -m d "FileSystem space used"` |
| {*item*} | Braces enclose a list from which you must choose an item in format and syntax descriptions. |
| [*item*] | Brackets enclose optional items in format and syntax descriptions. |
| <**Ctrl-***x*> | The notation <**Ctrl-***x*> indicates a control character sequence. For example, <**Ctrl-c**> means that you hold down the control key while pressing <**c**>. |
| *item*... | Ellipses indicate that you can repeat the preceding item one or more times. |
| \| | • In *syntax* statements, vertical lines separate a list of choices. In other words, a vertical line means *Or*.<br>• In the left margin of the document, vertical lines indicate technical changes to the information. |

## Prerequisite and related information

The LoadLeveler publications are:

- *AIX Installation Guide*, SC23-6791
- *Linux Installation Guide*, SC23-6789
- *Using and Administering*, SC23-6792
- *Diagnosis and Messages Guide*, SC23-6793
- *Command and API Reference*, SC23-6794
- *Resource Manager*, SC23-6790

To access all LoadLeveler documentation, refer to the IBM Cluster Information Center (http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp), which contains the most recent LoadLeveler documentation in PDF and HTML formats.

A **LoadLeveler Documentation Updates** file also is maintained on this Web site. The **LoadLeveler Documentation Updates** file contains updates to the LoadLeveler documentation. These updates include documentation corrections and clarifications that were discovered after the LoadLeveler information units were published.

Both the current LoadLeveler books and earlier versions of the library are also available in PDF format from the IBM Publications Center (http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss).

To easily locate a book in the IBM Publications Center, supply the book's publication number. The publication number for each of the LoadLeveler books is listed after the book title in the preceding list.

### Locating LoadLeveler software fixes and updates in Fix Central

To locate LoadLeveler software fixes and updates, do the following:

1. Go to the Fix Central website (http://www-933.ibm.com/support/fixcentral/?productGroup0=ibm/fcpower&productGroup1=ibm/ClusterSoftware&productGroup2=ibm/power/IBM).
2. In the **Product Group** pull-down menu, under **Software**, select **Cluster software**.
3. In the **Select from Cluster software** pull-down menu, choose **LoadLeveler**.
4. In the **Installed Version** pull-down menu, select the version you have installed, for example, **5.1.0**.
5. In the **Platform** pull-down menu, select the platform, for example, **All**.
6. Click **Continue** and select an option on the **Identify fixes** panel. For example, select **Browse for fixes** and click **Continue** again.
7. On the **Select fixes panel**, select fixes to download or, to view the Restrictions list, click on **Readme**, next to the appropriate fix and open the **Known Limitations** section.

## How to send your comments

Your feedback is important in helping us to produce accurate, high-quality information. If you have any comments about this book or any other LoadLeveler documentation, send your comments by e-mail to:

mhvrcfs@us.ibm.com

Include the book title and order number, and, if applicable, the specific location of the information you have comments on (for example, a page number or a table number).

For technical information and to exchange ideas related to high performance computing, go to:

- HPC Central (http://www.ibm.com/developerworks/wikis/display/hpccentral/HPC+Central)
- HPC Central Technical Forum (http://www.ibm.com/developerworks/forums/forum.jspa?forumID=1056)

# Summary of changes

## Version 5 Release 1

The following sections summarize changes to the IBM LoadLeveler product and library.

Within each information unit in the library, a vertical line to the left of text and illustrations indicates technical changes or additions made to the previous edition of the information.

**Attention:**

For LoadLeveler Version 5 Release 1, changes apply to Linux.

**Disclaimer:**

The functions or features found herein may not be available on all operating systems or platforms and do not indicate the availability of these functions or features within the IBM product or future versions of the IBM product. The development, release, and timing of any future features or functionality is at IBM's sole discretion. IBM's plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. The information mentioned is not a commitment, promise, or legal obligation to deliver any material, code or functionality. The information may not be incorporated into any contract and it should not be relied on in making a purchasing decision.

**Release notes:**

For the latest release notes, go to the Fix Central website (http://www-933.ibm.com/support/fixcentral/?productGroup0=ibm/fcpower&productGroup1=ibm/ClusterSoftware&productGroup2=ibm/power/IBM). For more information about Fix Central, see "Locating LoadLeveler software fixes and updates in Fix Central" on page xiii.

Changes to LoadLeveler for Linux Version 5 Release 1 product and library include:

- **New information:**
  - A new job command file keyword, **first_node_tasks**, has been added to allow users to request a different task count for the first node. The value specified for **tasks_per_node** would apply to all nodes except the first node.
  - Support for LoadLeveler as the scheduler for Blue Gene®/Q has been added.
    - A new Blue Gene® command called **llbgctl** has been added. You can use the **llbgctl** command to control Blue Gene system resources.
    - A new command called **llbgstatus** has been added that returns Blue Gene system status information.

    For more information, see *LoadLeveler: Command and API Reference*.
  - LoadLeveler now permits its **execute** directory to be located in volatile storage, such as RAM disk, where the contents may not exist following the reboot of an execute node. To avoid the potential performance impact of using a shared file system, configure the **execute** directory in either local disk, or in RAM disk.
  - Support for Workload Manager (WLM) on LoadLeveler for Linux on POWER® (RHEL 6.2) and on x86_64 (SLES 11) has been added.
  - A new affinity interface is introduced into LoadLeveler. The new interface is used by IBM Parallel Environment (PE) Runtime Edition on the x86 platform.

– Island scheduling is introduced in LoadLeveler. New configuration parameters permit administrators to define islands within a LoadLeveler cluster. You can now request that your jobs run within one island or across multiple islands.
– Support for running Intel Message Passing Interface (MPI) and Open MPI jobs as batch LoadLeveler jobs has been added.
– The **llrun** command has been added to allow you to run an interactive command that uses resources allocated by LoadLeveler.
– LoadLeveler provides support for running embarrassingly parallel jobs without using an MPI runtime process manager. For more information, see the following:
  - *LoadLeveler: Command and API Reference*
  - *LoadLeveler: Using and Administering*
– Support for the energy function on the x86_64 iDataPlex® Sandy Bridge processor for SUSE Linux Enterprise Server (SLES) 11 includes the following:
  - The ability to run jobs in a lower CPU frequency. To minimize energy consumption with minimal performance degradation, LoadLeveler provides the job energy policy tag to allow you to set the acceptable performance degradation for the job.
  - Job energy reports can be generated from the accounting data by using the **llsummary** command. The job energy consumption is calculated after the job finishes.
  - An administrator can switch idle nodes to standby state to save energy. This can be done either manually or LoadLeveler can do it automatically. You can wake up the idle nodes that are in standby mode to run the workload as needed.

  To support the energy function, the following have been added:
  - Four new commands: **llrchgmstat**, **llreinit**, **llrqetag**, **llrrmetag**. See *LoadLeveler: Resource Manager* for more information about these commands.
  - Two new subroutines: **llr_change_node_powerstate** and **llr_remove_energy_tags**. See *LoadLeveler: Resource Manager* for more information about these subroutines.

  For additional information, see the following:
  - *LoadLeveler: Command and API Reference*
  - *LoadLeveler: Using and Administering*
  - *LoadLeveler for Linux: Installation Guide*
– Support for checkpoint/restart on Red Hat Enterprise Linux (RHEL) 6.2, which includes the following:
  - The system administrator or user can now set the checkpoint interval on a per job basis.
  - Task migration, which provides a method to move portions of a running parallel job from the current allocated set of nodes to a new set of nodes. To support task migration, the following have been added:
    • The **llmigrate** command
    • The **ll_migrate_job_step** subroutine

  For more information, see the following:
  - *LoadLeveler: Command and API Reference*
  - *LoadLeveler: Resource Manager*
  - *LoadLeveler: Using and Administering*

- Support for multiple endpoints has been added. The new **endpoints** keyword is supported on a class basis for all jobs submitted in that class, as well as an option on the **network** keyword. For more information, see the following:
  - *LoadLeveler: Using and Administering*
  - *LoadLeveler: Resource Manager*
- Support for running LoadLeveler on Blue Gene/Q has been added on LoadLeveler for Linux on POWER (RHEL 6.2). For more information, see the following:
  - *LoadLeveler: Command and API Reference*
  - *LoadLeveler: Using and Administering*
  - *LoadLeveler: Resource Manager*

- **Changed information:**
  - The **llrctl** command is changed to be consistent with the LoadLeveler scheduler command, **llctl**. The same keywords supported in the **llctl** command are now supported in the **llrctl** command. The **drain jobmgr** and **resume jobmgr** keywords are no longer supported in the **llrctl** command.
  - The following table summarizes the Blue Gene/Q terminology changes:

| BG/L and BG/P (previous terms) | BG/Q terms |
|---|---|
| Partitions | Blocks |
| Base partitions | Midplanes |
| Wires | Cables |
| Node cards | Node boards |
| mpirun, submit, mpiexec | runjob |

- **Deleted information:**
  - Support for using virtual IP (VIP) addresses with checkpoint or restart has been removed, so all references to VIPs or the VIP server have been removed from LoadLeveler publications.
  - The **ALLOC_EXCLUSIVE_CPU_PER_JOB** configuration file keyword has been deprecated and has been removed.
  - The following Blue Gene job command file keywords have been deprecated and removed:
    - **bg_connection** (replaced with **bg_connectivity**)
    - **bg_partition** (replaced with **bg_block**)

# Part 1. Overview of LoadLeveler concepts and operation

Once you have a basic understanding of the LoadLeveler product and its interfaces, you can find more details in the topics listed in Table 2.

*Table 2. Major topics in LoadLeveler: Resource Manager*

| To learn about: | Read the following: |
|---|---|
| Performing administrator tasks | Part 2, "Configuring and managing the LoadLeveler environment," on page 21 |
| Using LoadLeveler interfaces | Part 3, "LoadLeveler interfaces reference," on page 89 |

# Chapter 1. What is LoadLeveler resource manager?

The LoadLeveler resource manager is an application for launching and managing serial and parallel jobs on large clusters of systems. The LoadLeveler resource manager provides an extensive set of features for controlling, managing, and using cluster resources. By interfacing with the LoadLeveler resource manager API, applications can exploit the many features, without having to re-implement them.

Figure 1 shows the different environments to which LoadLeveler can run jobs. Together, these environments comprise the *LoadLeveler cluster*.

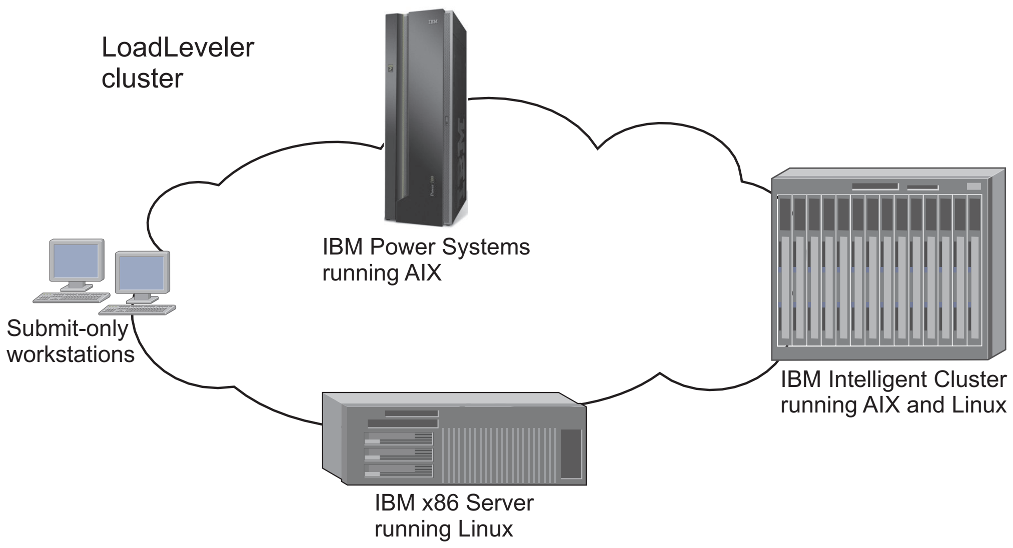
LoadLeveler cluster
IBM Power Systems running AIX
Submit-only workstations
IBM Intelligent Cluster running AIX and Linux
IBM x86 Server running Linux

*Figure 1. Example of a LoadLeveler cluster*

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

## LoadLeveler resource manager interfaces

LoadLeveler resource manager has various types of interfaces for controlling and managing jobs and cluster resources.

These interfaces include:

- Control files that define the elements, characteristics, and policies of the LoadLeveler resource manager. These files are the configuration file and the administration file. Optionally, a configuration database can replace the configuration and administration files.
- The command line interface, which gives you access to basic administrative functions.
- An application programming interface (API), which allows application programs written by users and administrators to interact with the LoadLeveler resource manager environment.

## Job definition

LoadLeveler resource manager runs your jobs on one or more machines. The definition of a **job**, in this context, is a set of **job steps**. For each job step, you can specify a different executable (the executable is the part of the job that gets processed).

A job and its job steps are defined using the LoadLeveler job command language. A job description specifies the name of the job, as well as the job steps that you want to run, and can contain other LoadLeveler statements.

You can submit batch jobs to LoadLeveler to run on specified resources. Batch jobs run in the background and generally do not require any input from the user. Batch jobs can either be **serial** or **parallel**. A serial job runs on a single machine. A parallel job is a program designed to execute as a number of individual, but related, processes on one or more of your system's nodes. When executed, these related processes can communicate with each other (through message passing or shared memory) to exchange data or synchronize their execution.

For parallel jobs, LoadLeveler resource manager interacts with the Parallel Operating Environment (POE) to allocate nodes, assign tasks to nodes, and launch tasks.

## Machine definition

Each machine in the LoadLeveler cluster performs one or more roles in managing resources and jobs. Roles performed in managing resources and jobs by each machine in the LoadLevelercluster are as follows:

- **Job Manager Machine:** When a job is submitted, it gets placed in a queue managed by a job manager machine. The job manager machine keeps persistent information about the job. This machine contacts the scheduling application and provides the job description and all job state changes to the scheduling application.
- **Executing Machine:** The machine that runs the job is known as the executing machine.
- **Resource Manager Machine:** The role of the resource manager is to collect status from all executing and job manager machines in the cluster.

Keep in mind that one machine can assume multiple roles as shown in Figure 2 on page 5.

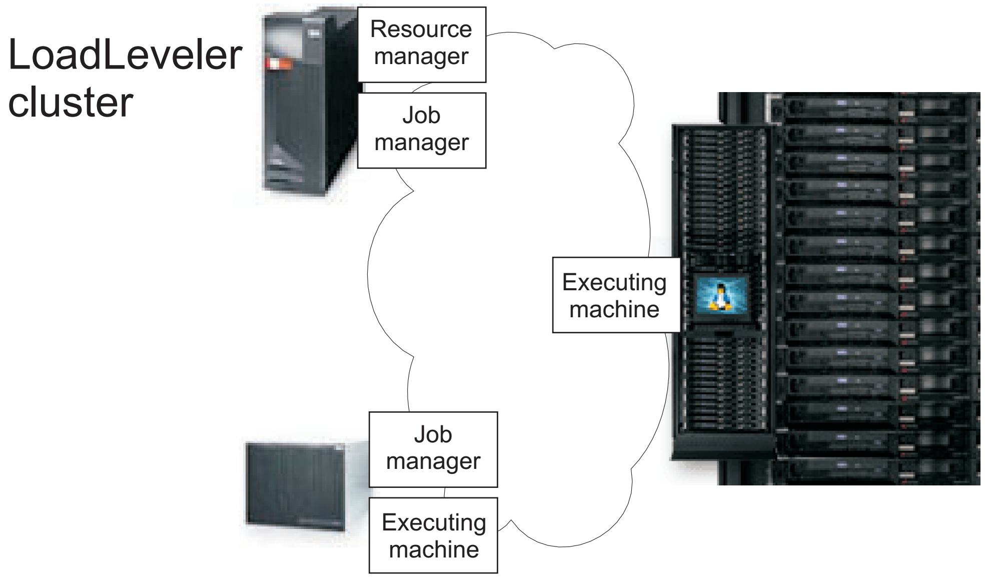

*Figure 2. Multiple roles of machines*

## How LoadLeveler resource manager interacts with a scheduling application

The LoadLeveler resource manager is intended to work with a scheduling application, which uses the API to launch and manage jobs. The scheduling application can be a daemon or a command based tool.

- Daemon – the scheduling application would be a daemon process that interacts with the resource manager. The following types are supported:
  - Event driven – the scheduling application receives the state of machines and jobs by way of events sent by the resource manager and makes scheduling decisions using that data and informs the resource manager where to run the jobs.
  - Polling – the scheduling application periodically polls the resource manager for jobs and machines, and makes scheduling decisions using that data, and informs the resource manager where to run the jobs.
- Command-based tool – the scheduling application would be a command-based tool for launching a job which would query the resource manager for machines and inform the resource manager where to run the job.

## How LoadLeveler runs jobs

LoadLeveler consists of the following two components: the **resource manager**, which is responsible for managing all machine and job resources, and the **scheduler**, which is responsible for scheduling jobs on the resources provided by the resource manager.

When a user submits a job, LoadLeveler examines the job command file to determine what resources the job will need. LoadLeveler determines which machine, or group of machines, is best suited to provide these resources, then LoadLeveler dispatches the job to the appropriate machines. To aid this process, LoadLeveler uses queues.

A **job queue** is a list of jobs that are waiting to be processed. When a user submits a job to LoadLeveler, the job is entered into an internal database, which resides on one of the machines in the LoadLeveler cluster, until it is ready to be dispatched to run on another machine.

Once LoadLeveler examines a job to determine its required resources, the job is dispatched to a machine to be processed. A job can be dispatched to either one machine, or in the case of parallel jobs, to multiple machines. Once the job reaches the executing machine, the job runs.

Jobs do not necessarily get dispatched to machines in the cluster on a first-come, first-serve basis. Instead, LoadLeveler examines the requirements and characteristics of the job and the availability of machines, and then determines the best time for the job to be dispatched.

"The LoadLeveler job cycle" on page 12 describes job flow in the LoadLeveler environment in more detail.

## How LoadLeveler daemons process jobs

LoadLeveler has its own set of daemons that control the processes moving jobs through the LoadLeveler cluster.

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

Machine and adapter configuration and status changes are now detected by the region manager and the startd daemons.

Table 3 summarizes these daemons, which are described in further detail in topics immediately following the table.

*Table 3. LoadLeveler daemons*

| Daemon | Description |
|---|---|
| LoadL_master | Referred to as the master daemon. Runs on all machines in the LoadLeveler cluster and manages other daemons. |
| LoadL_schedd | Referred to as the schedd daemon or the job manager. Receives jobs from the resource manager **llr_add_job** API and manages them on machines specified by the resource manager **llr_start_job_step** API. |
| LoadL_startd | Referred to as the startd daemon. The startd daemon performs the following functions: <br>• Monitors job and machine resources on local machines <br>• Generates local adapter configuration <br>• Forwards information to the resource manager and region manager daemons <br>• Sends heartbeats to the region manager daemon <br><br>The startd daemon spawns the starter process (**LoadL_starter**) which manages running jobs on the executing machine. |

*Table 3. LoadLeveler daemons (continued)*

| Daemon | Description |
|---|---|
| LoadL_region_mgr | Referred to as the region manager daemon. Detects and monitors machine and adapter status from the startd machines it manages and sends this information to the **LoadL_resource_mgr** daemons. |
| LoadL_resource_mgr | Referred to as the resource manager daemon. Collects all machine updates from the executing machine. Generates events to the scheduler for changes in the machine status. Maintains a list of all jobs managed by the resource manager. Responds to query requests for machine, job, and cluster information. |
| LoadL_kbdd | Referred to as the keyboard daemon. Monitors keyboard and mouse activity. |

## The master daemon

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

The **LoadL_master** binary is installed as a **setuid** program with the owner set to **root**. The **master** daemon and all daemons started by the **master** must be able to run with **root** privileges in order to switch the identity to the owner of any job being processed.

The **master** daemon determines whether to start any other daemons by checking the configured value of the **START_DAEMONS** keyword for the node. If the keyword is set to **true**, the daemons are started. If the keyword is set to **false**, the **master** daemon terminates and generates a message.

The **master** daemon will not start on a Linux machine if **SEC_ENABLEMENT** is set to **CTSEC**. If the **master** daemon does not start, no other daemons will start.

On machines designated as primary or alternate for running the region manager and resource manager, the **master** controls the failover function for these daemons. If a region manager or resource manager machine failure is detected on a primary machine, the **master** will start the appropriate daemons on an alternate machine.

The **master** daemon starts and if necessary, restarts all of the LoadLeveler daemons that the machine it resides on is configured to run. This daemon also runs the **kbdd** daemon, which monitors keyboard and mouse activity.

When the **master** daemon detects a failure on one of the daemons that it is monitoring, it attempts to restart it. Because this daemon recognizes that certain situations may prevent a daemon from running, it limits its restart attempts to the number defined for the **RESTARTS_PER_HOUR** keyword in the configuration file. If this limit is exceeded, the **master** daemon forces all daemons including itself to exit.

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

The **master** daemon may perform the following actions in response to an **llrctl** command:
- Kill all daemons and exit (**stop** keyword)
- Kill all daemons and execute a new **master** (**recycle** keyword)
- Send drain request to Schedd (**drain** keyword)
- Send resume request to Schedd and send result to caller (**resume** keyword)

## The Schedd daemon

The **Schedd** or job manager daemon receives jobs sent by the resource manager **llr_add_job** API and manages those jobs to machines specified in the resource manager **llr_start_job_step** API.

The **Schedd** daemon can be in any one of the following activity states:

**Available**
This machine is available to manage jobs.

**Drained**
The **Schedd** machine accepts no more jobs. There are no jobs in starting or running state. Jobs in the Idle state are drained, meaning they will not get dispatched.

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

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

The **Schedd** daemon performs the following functions:
- Assigns new job identifiers when requested by the job submission process.
- Receives new jobs from the **llr_add_job** API. A new job is received as a *job object* for each job step. A job object is the data structure in memory containing all the information about a job step. The **Schedd** forwards the job information to the scheduling application.
- Maintains on disk copies of jobs submitted locally (on this machine) that are either waiting or running on a remote (different) machine. A scheduling application can use this information to reconstruct the job information in the event of a failure. This information is also used for accounting purposes.
- Responds to directives sent by the resource manager API. The directives include:
  - Run a job.
  - Remove a job.
  - Send information about all jobs.
- Sends job events to the scheduler application when:
  - **Schedd** is restarting.
  - A new job arrives.
  - A job is started.
  - A job was rejected, completed, removed, or vacated. **Schedd** determines the status by examining the exit status returned by the **startd**.
- Communicates with the Parallel Operating Environment (POE) when you run an interactive POE job.
- Requests that a remote **startd** daemon end a job.

- Receives accounting information from **startd**.
- Receives requests for reservations.
- Collects resource usage data when jobs terminate and stores it as historic fair share data in the $(SPOOL) directory.
- Sends historic fair share data to the central manager when it is updated or when the **Schedd** daemon is restarted.
- Maintains and stores records of historic CPU and IBM System Blue Gene Solution utilization for users and groups known to the **Schedd**.
- Passes the historic CPU and Blue Gene utilization data to the central manager.

## The startd daemon

The **startd** daemon monitors the status of each job, reservation, and machine in the cluster, and forwards this information to the resource manager daemon. The **startd** daemon also receives and executes job requests originating from remote machines. The startd daemon generates its adapter information and sends it to the resource and region manager daemons. It also sends UDP heartbeat packets to the region manager daemon. The master daemon starts, restarts, signals, and stops the **startd** daemon.

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

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

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

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

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

The **startd** daemon performs these functions:

- Runs a time-out procedure that includes building a snapshot of the state of the machine that includes static and dynamic data. This time-out procedure is run at the following times:
  - After a job completes.
  - According to the definition of the **POLLING_FREQUENCY** keyword in the configuration file.
- Records the following information in LoadLeveler variables and sends the information to the negotiator.
  - State (of the startd daemon)
  - EnteredCurrentState
  - Memory
  - Disk
  - KeyboardIdle
  - Cpus
  - LoadAvg
  - Machine
  - Adapter
  - AvailableClasses
- Calculates the SUSPEND, RESUME, CONTINUE, and VACATE expressions through which you can manage job status.
- Receives job requests from the **Schedd** daemon to:
  - Start a job

  - Preempt or resume a job
  - Vacate a job
  - Cancel

  When the **Schedd** daemon tells the **startd** daemon to start a job, the **startd** determines whether its own state permits a new job to run.
- Receives requests from the master (through the **llrctl** command) to:
  - Resume (**resume** keyword)
- Receives notification of keyboard and mouse activity from the **kbdd** daemon
- Periodically examines the process table for LoadLeveler jobs and accumulates resources consumed by those jobs. This resource data is used to determine if a job has exceeded its job limit and for recording in the history file.
- Sends accounting information to the job manager.

### The starter process

The **startd** daemon spawns a **starter** process after the job manager tells the **startd** daemon to start a job.

The starter process manages all the processes associated with a job step. The starter process is responsible for running the job and reporting status back to the **startd** daemon.

The starter process performs these functions:
- Processes the prolog and epilog programs as defined by the **JOB_PROLOG** and **JOB_EPILOG** keywords in the configuration. The job will not run if the prolog program exits with a return code other than zero.
- Handles authentication. This includes:
  - Authenticates AFS, if necessary
  - Verifies that the submitting user is *not* **root**
  - Verifies that the submitting user has access to the appropriate directories in the local file system.
- Runs the job by forking a child process that runs with the user ID and all groups of the submitting user. That child process creates a new process group of which it is the process group leader, and executes the user's program or a shell.

  The starter process is responsible for detecting the termination of any process that it forks. To ensure that all processes associated with a job are terminated after the process forked by the starter terminates, process tracking must be enabled. To configure LoadLeveler for process tracking, see "Tracking job processes" on page 49.
- Responds to vacate and suspend orders from the **startd**.

## The region manager daemon

The region manager daemon detects and monitors node and adapter status from all of the startd machines it manages and sends the status information to the resource manager daemon.

The **region manager** daemon will only start up if a region is defined in the configuration. If the **region manager** daemon is not configured, the adapter and node status will only come from the configuration information available to the startd daemon and will not reflect the actual connectivity of the adapter or node.

The **region manager** daemon requires specific information, such as:
- The name of the primary machine on which it will run.
- The name of the backup or alternate machine.

- The region or list of machines reporting to it.

A LoadLeveler cluster can consist of more than one region and can have more than one region manager. A region, or managed region, consists of a list of machines that will send heartbeats and report information to the region manager.

The region manager node must have similar connectivity to the network as the startd nodes it manages, so that all of its configured network interfaces are able to connect to all of the startd node's network interfaces.

The **region manager** daemon performs the following functions:
- Maintains an in-memory database containing all machines in its managed region and their adapter ports
- Requests and receives adapter configuration information from the startds every time it starts up
- Updates its in-memory database whenever it receives adapter configuration information from a startd (for example, when a startd is reconfigured)
- Creates heartbeat IP pairs between the startd (source) and the region manager (destination) ports and sends them back to the corresponding startd
- Receives heartbeats from the startds
- Determines heartbeat status based on the UDP heartbeat packets from the startd daemon
- Sends heartbeat status to the central manager and resource manager

## The resource manager daemon

The **resource manager** daemon collects all machine updates from the executing machine, generates events to the scheduling application for changes in machine status, maintains a list of all jobs managed by the resource manager, and responds to query requests for machine, job, and cluster information.

## The kbdd daemon

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

The **kbdd** daemon notifies the **startd** daemon when it detects keyboard or mouse activity; however, **kbdd** is *not* interrupt driven. It sleeps for the number of seconds defined by the **POLLING_FREQUENCY** keyword in the LoadLeveler configuration file, and then determines if X events, in the form of mouse or keyboard activity, have occurred. For more information on the configuration file, see Chapter 4, “Defining LoadLeveler resources to administer,” on page 61.

## The LoadLeveler job cycle

To illustrate the flow of job information through the LoadLeveler cluster, a description and sequence of diagrams have been provided.

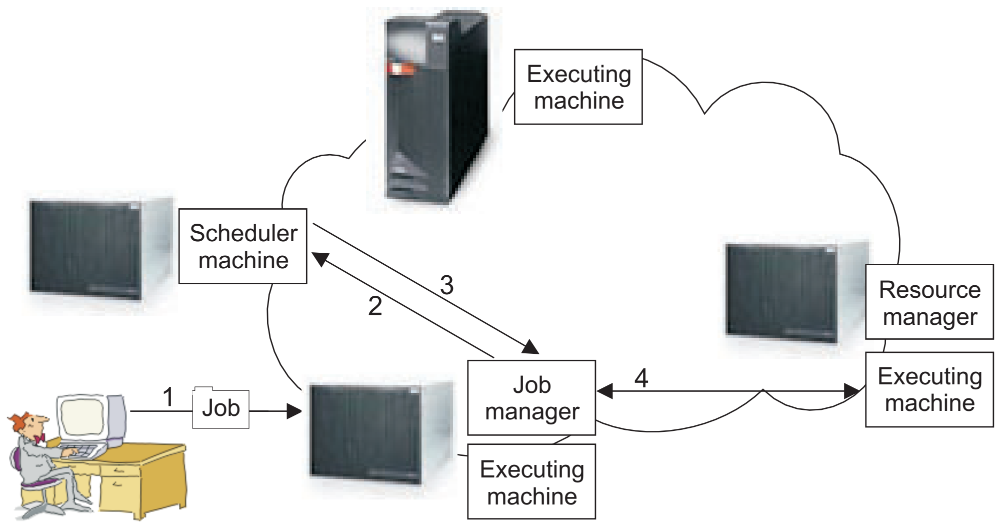

*Figure 3. High-level job flow*

In the resource manager there are machines that act as job managers and machines that serve as the executing machines. In addition, an external scheduler is running on a machine and interfacing with the resource manager. The arrows in Figure 3 illustrate the following:

- Arrow 1 indicates that a job has been submitted to the resource manager.
- Arrow 2 indicates that the job manager contacts the scheduler to inform it that a job has been submitted.
- Arrow 3 indicates that the scheduler informs the job manager on which machines to run the job.
- Arrow 4 indicates that the job manager contacts the executing machine and provides it with information regarding the job. In this case, the job manager and executing machines are different machines in the cluster, but they do not have to be different; the job manager and executing machines may be the same physical machine.

Figure 3 is broken down into the following more detailed diagrams illustrating how the resource manager processes a job. The diagrams indicate specific job states for this example, but do not list all of the possible states for resource manager jobs. A complete list of job states appears in "LoadLeveler job states" on page 15.

1. Submit a resource manager job:

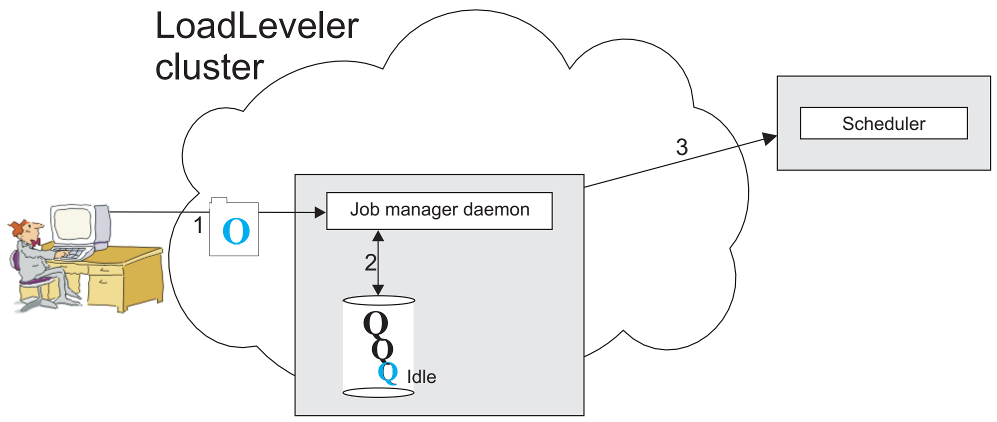

*Figure 4. Job is submitted to resource manager*

Figure 4 illustrates that the job manager daemon is running on the machine. This machine can also have the startd daemon running on it. In this figure the scheduler is a daemon running on a machine. The arrows in Figure 4 illustrate the following:

- Arrow 1 indicates that a job has been submitted to the job manager.
- Arrow 2 indicates that the job manager stores all of the relevant job information in a database.
- Arrow 3 indicates that the job manager sends the job information to the scheduler. At this point, the submitted job is in the Idle state.

2. Machines selected to run the job:

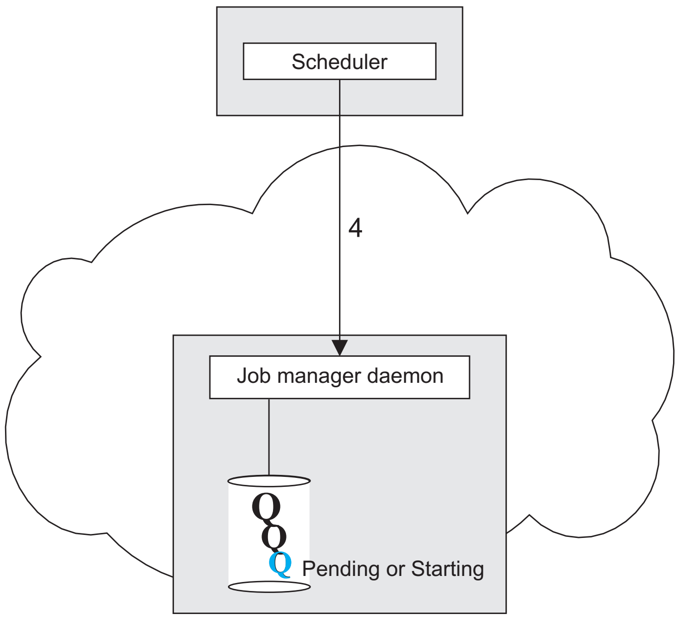

*Figure 5. Scheduler selects machines for the job*

In Figure 5, arrow 4 indicates that the scheduler selects machines for the job and informs the job manager to begin taking steps to run the job. Once this is done, the job is considered Pending or Starting.

3. Prepare to run:

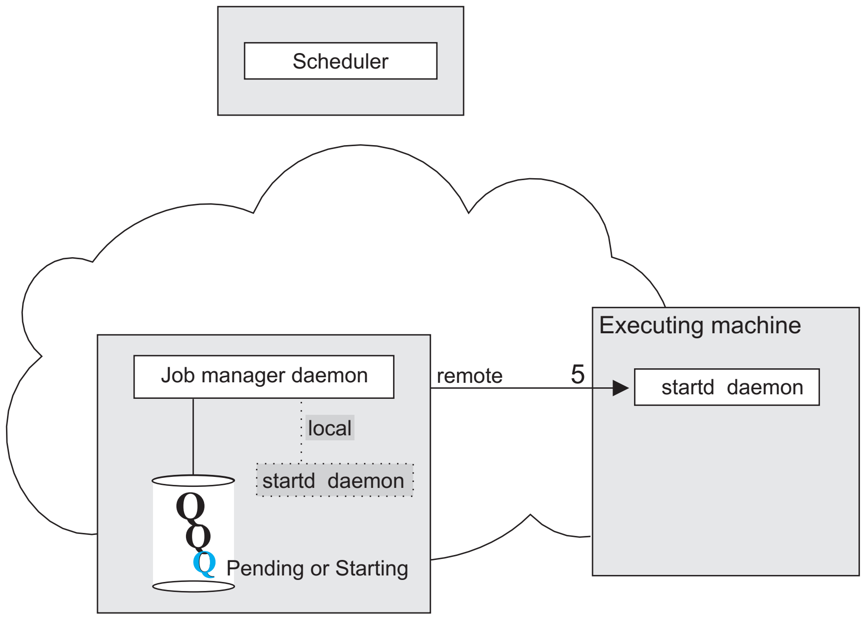

*Figure 6. Resource manger prepares to run the job*

In Figure 6, arrow 5 illustrates that the job manager contacts the startd daemon on the executing machine and requests that it start the job. The executing machine can either be a local machine (the machine to which the job was submitted) or another machine in the cluster. In this example, the local machine is **not** the executing machine.

4. Initiate job:

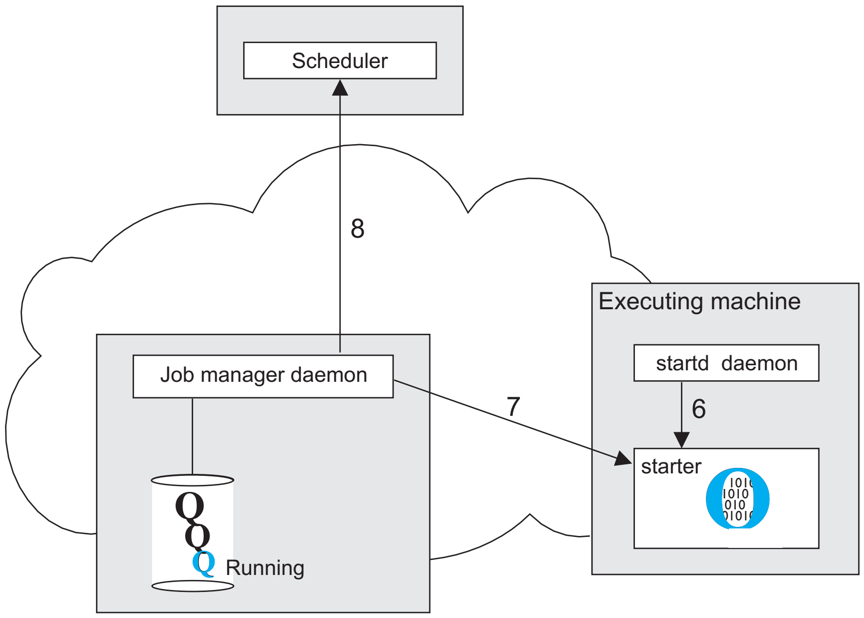

*Figure 7. Resource manager starts the job*

The arrows in Figure 7 illustrate the following:

- Arrow 6 indicates that the **startd** daemon on the executing machine spawns a **starter** process for the job.

- Arrow 7 indicates that the job manager sends the starter process the job information and the executable.
- Arrow 8 indicates that the job manager notifies the scheduler that the job has been started.

The starter forks and executes the user's job, and the starter parent waits for the child to complete. At this point, the job is in Running state.

5. Complete job:

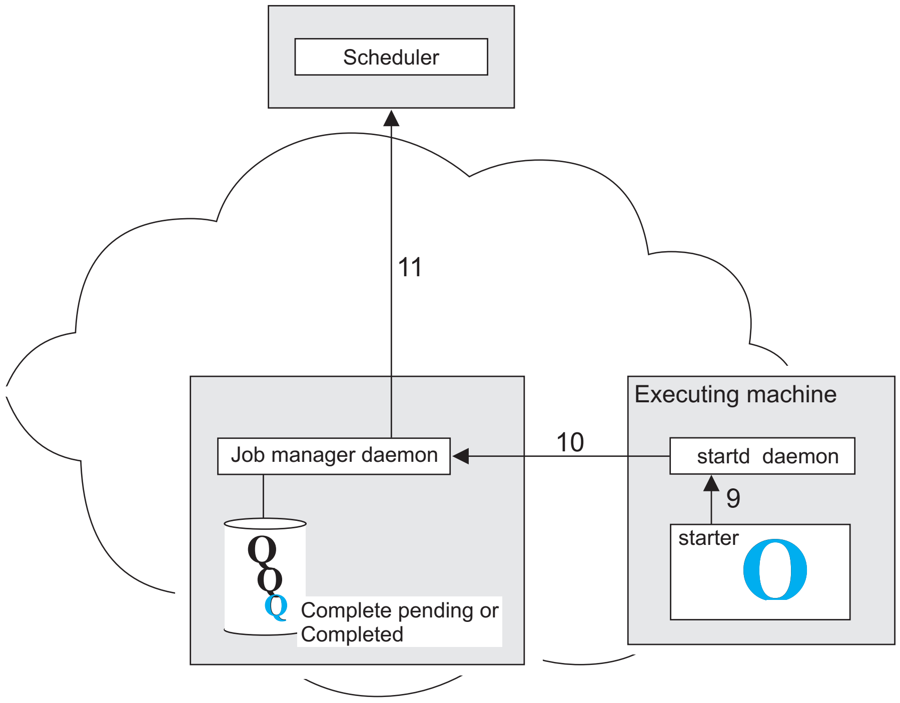

*Figure 8. LoadLeveler completes the job*

The arrows in Figure 8 illustrate the following:

- Arrow 9 indicates that when the job completes, the starter process notifies the startd daemon.
- Arrow 10 indicates that the startd daemon notifies the job manager.
- Arrow 11 indicates that the job manager forwards it to the scheduler. At this point, the job is in Completed state.

## LoadLeveler job states

As the resource manager processes a job, the job moves through various states. These states are listed in Table 4.

Some options on resource manager interfaces are valid only for jobs in certain states. For example, the **llr_start_job_step** API can only apply to jobs that are in the Idle state.

*Table 4. Job state descriptions and abbreviations*

| Job state | Abbreviation in displays / output | Description |
|---|---|---|
| Checkpointing | CK | Indicates that a checkpoint has been initiated. |

*Table 4. Job state descriptions and abbreviations (continued)*

| Job state | Abbreviation in displays / output | Description |
|---|---|---|
| Completed | C | The job has completed. |
| Idle | I | The job is being considered to run on a machine, though no machine has been selected. |
| Pending | P | The job is in the process of starting on one or more machines. (The negotiator indicates this state until the Schedd acknowledges that it has received the request to start the job. Then the negotiator changes the state of the job to Starting. The Schedd indicates the Pending state until all startd machines have acknowledged receipt of the start request. The Schedd then changes the state of the job to Starting.) |
| Preempted | E | The job is preempted. This state applies only when LoadLeveler uses the suspend method to preempt the job. |
| Rejected | X | The job is rejected. |
| Removed | RM | The job was stopped by LoadLeveler. |
| Remove Pending | RP | The job is in the process of being removed, but not all associated machines have acknowledged the removal of the job. |
| Running | R | The job is running: the job was dispatched and has started on the designated machine. |
| Starting | ST | The job is starting: the job was dispatched, was received by the target machine, and LoadLeveler is setting up the environment in which to run the job. For a parallel job, LoadLeveler sets up the environment on all required nodes. See the description of the “Pending” state for more information on when the negotiator or the Schedd daemon moves a job into the Starting state. |

## Consumable resources

Consumable resources are assets available on machines in your LoadLeveler cluster.

These assets are called "resources" because they model the commodities or services available on machines (including CPUs, real memory, virtual memory, large page memory, software licenses, disk space). They are considered "consumable" because job steps use specified amounts of these commodities when the step is running. Once the step finishes, the resource becomes available for another job step.

Consumable resources which model the characteristics of a specific machine (such as the number of CPUs or the number of specific software licenses available only on that machine) are called machine resources. Consumable resources which model resources that are available across the LoadLeveler cluster (such as floating software licenses) are called floating resources. For example, consider a configuration with 10 licenses for a given program (which can be used on any machine in the cluster). If these licenses are defined as floating resources, all 10 can be used on one machine, or they can be spread across as many as 10 different machines.

The LoadLeveler administrator can specify:
- Consumable resources to be sent to the scheduling application
- Quantity of resources available on specific machines
- Quantity of floating resources available on machines in the cluster
- Whether CPU, real memory, virtual memory, or large page resources should be enforced using Workload Manager (WLM)

Jobs submitted to the resource manager can specify the resources consumed by each task of a job step, or the resources consumed by the job on each machine where it runs, regardless of the number of tasks assigned to that machine.

## Consumable resources and Workload Manager

If the administrator has indicated that resources should be enforced, resource manager uses Workload Manager (WLM) to give greater control over CPU, real memory, virtual memory and large page resource allocation.

WLM monitors system resources and regulates their allocation to processes. These actions prevent jobs from interfering with each other when they have conflicting resource requirements. WLM achieves this control by creating different classes of service and allowing attributes to be specified for those classes.

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

When resource manager defines multiple memory resources under one WLM class, WLM uses the following order to determine if resource limits have been exceeded:
1. Real Memory Absolute Limit
2. Virtual Memory Absolute Limit
3. Large Page Limit
4. Real Memory shares or percent limit

**Note:** When real memory or CPU with either shares or percent limits are exceeded, the job processes in that class receive a lower scheduling priority until their utilization drops below the hard max limit. When virtual memory or absolute real memory limits are exceeded, the job processes are killed. When the large page limit is exceeded, any new large page requests are denied.

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

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

**Note:** A WLM class is active for the duration of a job step and is deleted when the job step completes. On AIX, there is a limit of 8192 active WLM classes per machine. Therefore, when resources are being enforced on AIX, only 8192 job steps can be running on one machine.

For additional information about integrating LoadLeveler with Workload Manager, see "Steps for integrating LoadLeveler with the Workload Manager" on page 76.

# Chapter 2. What operating systems are supported by LoadLeveler resource manager?

LoadLeveler resource manager supports three operating systems.

- **AIX 7.1**

  IBM's AIX 7.1 is an open UNIX operating environments that conform to The Open Group UNIX 98 Base Brand industry standard. AIX 7.1 provides high levels of integration, flexibility, and reliability and it operates on IBM Power® Systems and IBM Cluster 1600 servers and workstations.

  AIX 7.1 supports the concurrent operation of 32- and 64-bit applications, with key internet technologies such as Java™ and XML parser for Java included as part of the base operating system.

  A strong affinity between AIX and Linux permits popular applications developed on Linux to run on AIX 7.1 with a simple recompilation.

- **Linux**

  LoadLeveler supports the following distribution of Linux:
  - Red Hat Enterprise Linux (RHEL) 6.2 on LoadLeveler for Linux on POWER
  - SUSE Linux Enterprise Server (SLES) 11 on LoadLeveler for Linux on x86 Architecture
  - RHEL 6.2 on LoadLeveler for Linux on x86 Architecture

- **IBM System Blue Gene Solution**

  While no LoadLeveler processes actually run on the Blue Gene machine, LoadLeveler resource manager can interact with the Blue Gene machine to start and monitor jobs on the Blue Gene machine.

## LoadLeveler for AIX and LoadLeveler for Linux compatibility

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

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

Although the LoadLeveler resource manager for AIX and Linux are compatible, they do have some differences in the level of support for specific features. For further details, see the following topics:
- "Restrictions for LoadLeveler for Linux" on page 20.
- "Features not supported in LoadLeveler for Linux" on page 20.
- "Restrictions for LoadLeveler for AIX and LoadLeveler for Linux mixed clusters" on page 20.

## Restrictions for LoadLeveler for Linux

LoadLeveler for Linux supports a subset of the features that are available in the LoadLeveler for AIX product.

The following features are available, but are subject to restrictions:

- LoadLeveler resource manager for Linux supports the 64-bit LoadLeveler API library (libllrapi.so) on RHEL 6.2 and SLES 11 on IBM xSeries® servers with AMD Opteron or Intel EM64T processors
- Support for AFS file systems

  LoadLeveler for Linux support for authenticated access to AFS file systems is limited to RHEL 6.2 on IBM xSeries® servers with AMD Opteron or Intel EM64T processors.
- Support for Workload Management (WLM):
  - Hard policy for ConsumableCpus is not available for WLM for Linux.
  - Large page memory supports only a 16 MB page size.

## Features not supported in LoadLeveler for Linux

LoadLeveler for Linux supports a subset of the features that are available in the LoadLeveler for AIX product.

The following features are not supported:

- CtSec security

  LoadLeveler for AIX can exploit CtSec (Cluster Security Services) security functions. These functions authenticate the identity of users and programs interacting with LoadLeveler. These features are not available in this release of LoadLeveler resource manager for Linux.
- System error log

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

## Restrictions for LoadLeveler for AIX and LoadLeveler for Linux mixed clusters

Several restrictions apply when operating a LoadLeveler cluster that contains AIX and Linux machines.

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

- The node running the resource manager daemon must run a version of LoadLeveler resource manager equal to or higher than any LoadLeveler resource manager version being run on a node in the cluster.
- CtSec security features cannot be used.
- Jobs that use checkpointing must be sent to nodes with the same operating system for execution. This can be done by either defining and specifying job checkpointing for job classes that exist only on AIX or Linux nodes or by coding appropriate requirements expressions.

# Part 2. Configuring and managing the LoadLeveler environment

After installing IBM LoadLeveler resource manager, you may customize it by modifying both the **configuration** file and the **administration** file.

The configuration file contains many parameters that you can set or modify that will control how the resource manager operates. The administration file optionally lists and defines the machines in the cluster and the characteristics of classes.

To easily manage the resource manager, you should have one global configuration file and only one administration file, both centrally located on a machine in the cluster. Every other machine in the cluster must be able to read the configuration and administration file that are located on the central machine.

You may have multiple local configuration files that specify information specific to individual machines.

The resource manager does not prevent you from having multiple copies of administration files, but you need to be sure to update all the copies whenever you make a change to one. Having only one administration file prevents any confusion.

# Chapter 3. Configuring the LoadLeveler resource manager environment

One of your main tasks as system administrator is to configure LoadLeveler resource manager.

To configure the resource manager, you need to know the following configuration information:
- The LoadLeveler user ID and group ID
- The source of the configuration
- The location of the configuration

Configuring the resource manager involves modifying the configuration data that specify the terms under which the resource manager can use machines.

Table 5 identifies where you can find more information about using either the configuration and administration files or using a database as the source for LoadLeveler configuration, and keywords to modify the LoadLeveler environment.

*Table 5. Roadmap of tasks for LoadLeveler administrators*

| To learn about: | Read the following: |
|---|---|
| Controlling how LoadLeveler operates by customizing the global and local configuration files or a database | Chapter 3, "Configuring the LoadLeveler resource manager environment" |
| Customizing LoadLeveler resources | Chapter 4, "Defining LoadLeveler resources to administer," on page 61 |
| Additional ways to customize LoadLeveler configuration | Chapter 5, "Performing additional administrator tasks," on page 75 |

To control or monitor LoadLeveler operations by using the LoadLeveler commands and APIs, see:
- *LoadLeveler: Command and API Reference*
- Chapter 8, "Commands," on page 151
- Chapter 9, "Application programming interfaces (APIs)," on page 221

You can run your installation with default values set by LoadLeveler, or you can change any or all of them. Table 6 lists topics that discuss how you may configure the LoadLeveler environment by modifying the configuration file.

*Table 6. Roadmap of administrator tasks related to using or modifying the LoadLeveler configuration*

| To learn about: | Read the following: |
|---|---|
| Modifying LoadLeveler user and the source of the configuration | "The master configuration file" on page 24 |

Table 6. Roadmap of administrator tasks related to using or modifying the LoadLeveler configuration (continued)

| To learn about: | Read the following: |
|---|---|
| Defining major elements of the LoadLeveler configuration | • "Defining LoadLeveler administrators" on page 29<br>• "Defining a LoadLeveler cluster" on page 29<br>• "Defining LoadLeveler machine characteristics" on page 39<br>• "Defining security mechanisms" on page 39 |
| Enabling optional LoadLeveler functions | • "Gathering job accounting data" on page 43<br>• "Managing job status through control expressions" on page 47<br>• "Tracking job processes" on page 49 |
| Modifying LoadLeveler operations through installation exits | "Providing additional job-processing controls through installation exits" on page 50 |

## The master configuration file

By default, LoadLeveler uses the **loadl** user name and **loadl** group name to set the effective user ID and group ID to run LoadLeveler daemons. Also by default, LoadLeveler will use a set of configuration files as the source for configuration data. It expects that the global configuration file will be named **LoadL_config** and will be found in the **loadl** user's home directory.

To override these defaults, you must provide a master configuration file. The default master configuration file name is **/etc/LoadL.cfg**. This cannot be used to switch between two database-based LoadLeveler configurations, because **/etc/xcat/cfgloc** is also used to determine the dataspace name.

### Setting the LoadLeveler user

You can override the default LoadLeveler user name and group name by specifying the following keywords in the master configuration file:

**LoadLUserid**
Specifies the LoadLeveler user ID.

**LoadLGroupid**
Specifies the LoadLeveler group ID.

**Notes:**

1. If you change the LoadLeveler user ID to something other than **loadl**, and you are using configuration files as the source for configuration data, you will have to make sure your configuration files are owned by this ID.
2. If Cluster Security (CtSec) services is enabled, make sure you update the **unix.map** file if the **LoadLUserid** is specified as something other than **loadl**. Refer to "Steps for enabling CtSec services" on page 41 for more details.

### Setting the configuration source

LoadLeveler configuration data is obtained in one of three ways, depending on the content of the **/etc/LoadL.cfg** file:

1. By reading the LoadLeveler configuration database tables
2. By reading LoadLeveler configuration files
3. By fetching the configuration data from a LoadLeveler daemon on a different machine

One of the following keywords, **LoadLConfig**, **LoadLConfigHosts**, and **LoadLDB** can be specified in the master configuration file to designate which method will be used on a machine to obtain LoadLeveler configuration data:

**LoadLConfig**

Specifies the full path name of the global configuration file which identifies the set of configuration files. The set consists of the global configuration file, the administration file, and the local configuration file. The global configuration file must contain the location of the administration file, and may optionally contain the location of a local configuration file.

**Syntax:**

```
LoadLConfig = global configuration file path
```

**LoadLConfigHosts**

This keyword is used to designate one or more hosts that will serve LoadLeveler configuration data. LoadLeveler processes will attempt to retrieve configuration data from the hosts in the order they are listed. The hosts specified in this list must be configured to use a database for LoadLeveler configuration data.

**Syntax:**

```
LoadLConfigHosts = host1 host2 .. hostn
```

**LoadLDB**

This keyword designates an **odbc.ini** stanza name identifying the database, or data source name (DSN), to be used for LoadLeveler configuration data. The stanza used must be defined in the **/etc/odbc.ini** file. When setting up the MySQL xCAT database, set up a stanza in **/etc/odbc.ini** and specify the stanza name in the **LoadLDB** statement.

**Syntax:**

```
LoadLDB = xcatdb
```

If none of the keywords, **LoadLConfig**, **LoadLConfigHosts**, or **LoadLDB**, are specified, then by default, LoadLeveler will expect to find a global configuration file in the home directory of the LoadLeveler user ID, and will obtain configuration data from configuration files.

## Overriding the shared memory key

LoadLeveler uses a shared memory segment to cache parsed configuration data read from the LoadLeveler configuration database tables or from LoadLeveler configuration and administration files. LoadLeveler uses a key to uniquely identify its shared memory segment. Normally a default shared memory segment key generated by LoadLeveler is sufficient for this purpose. However, there is a small possibility that another application, running on the same nodes as LoadLeveler, will use the same shared memory segment key. In this case, LoadLeveler will generate a non-default key, and store it in the master configuration file, using the **LoadLConfigShmKey** keyword.

This keyword can also be used by the LoadLeveler administrator to override the default for the shared memory segment key.

**LoadLConfigShmKey**

Records the value used for the shared memory segment key when it is not possible to use the generated default key because of a conflict with another shared memory application.

**Syntax:**

```
LoadLConfigShmKey = number
```

All LoadLeveler processes that need to access configuration data will first read the master configuration file. If this keyword is specified, the process will use the specified key to access the LoadLeveler shared memory segment. If this keyword is not specified, a default key will be generated.

**Note:** The shared memory segment key does not have to be the same on all nodes of the LoadLeveler cluster.

**Default value:** No default value is set.

## File-based configuration

LoadLeveler uses file-based configuration by default. For file-based configuration, the master configuration file is optional. Three types of files are used to define the configuration:

- *Global configuration file:* This file, by default, is called the **LoadL_config** file and it contains configuration information common to all nodes in the LoadLeveler cluster. Use the **LoadLConfig** keyword in the master configuration file to specify the location and name of the file if you do not want to use the default.
- *Local Configuration File:* This file is generally called **LoadL_config.local** (although it is possible for you to rename it). This file contains specific configuration information for an individual node. The **LoadL_config.local** file is in the same format as **LoadL_config** and the information in this file overrides any information specified in **LoadL_config**. It is an optional file that you use to modify information on a local machine. Its full path name is specified in the **LoadL_config** file by using the **LOCAL_CONFIG** keyword.
- *Administration file:* The administration file optionally lists and defines the machines in the LoadLeveler cluster and the characteristics of classes, users, and groups. Its full path name is specified in the **LoadL_config** file by using the **ADMIN_FILE** keyword.

To easily manage LoadLeveler, you should have one global configuration file and only one administration file, both centrally located on a machine in the LoadLeveler cluster. Every other machine in the cluster must be able to read the configuration and administration file that are located on the central machine.

You may have multiple local configuration files that specify information specific to individual machines.

LoadLeveler does not prevent you from having multiple copies of administration files, but you need to be sure to update all the copies whenever you make a change to one. Having only one administration file prevents any confusion.

## Database configuration option

When LoadLeveler is part of the software stack in a cluster managed by xCAT, LoadLeveler's configuration can share xCAT's MySQL or DB2® database space. In this case, the master configuration file is required and the **LoadLDB** indicates the data source name.

For more information, see:
- *LoadLeveler for AIX: Installation Guide*
- *LoadLeveler for Linux: Installation Guide*

Or, go to *Setting Up MySQL as the xCAT DB* or *Setting Up DB2 as the xCAT DB* at XCAT Documentation (http://sourceforge.net/apps/mediawiki/xcat/index.php?title=XCAT_Documentation).

To initialize the configuration, run the **llconfig** or **llrconfig** command with the **-i** option. You must specify configuration files as input to this command. Administration files cannot be specified in the list. Administration files are specified by the **ADMIN_FILE** keyword in the configuration file. All of the data from the configuration files that are used for initialization are stored in the database for the default machine. To specify configuration settings for individual machines, you must update the configuration after initialization.

You can manage the database-based configuration by using the **llconfig** or **llrconfig** options or LoadLeveler's configuration editor to display and change the values of keywords and stanzas in the database.

### Understanding remotely configured nodes

When LoadLeveler is configured to use a database to contain LoadLeveler configuration data, it is possible to configure "remotely configured nodes" that do not have access to the database. This kind of configuration permits an installation to define a LoadLeveler cluster without having to provide database clients on all nodes.

The remotely configured nodes are configured by specifying the **LoadLConfigHosts** keyword in the master configuration file. This keyword is used to specify the host, or hosts, that can serve configuration data to the remotely configured node. For more information about the **LoadLConfigHosts** keyword, see “Setting the configuration source” on page 24.

There are limitations to LoadLeveler nodes that are configured in this way because they do not have direct access to the configuration data in the database and are dependent on another node for configuration data.

Not all LoadLeveler daemons can run on remotely configured nodes. It is expected that remotely configured nodes would be configured to run the **startd** daemon (and starter process) and optionally the keyboard daemon. The central manager and resource manager depend on having database access and may not be configured on a remotely configured node. Although the regional manager and schedd daemon may be configured on a remotely configured node, it is not recommended, because the use of a database for these daemons could change in the future.

There is a situation where LoadLeveler processes (daemons, commands or APIs) invoked on a remotely configured node may not always work if a configuration server is not available. A shared memory segment is used to cache configuration data. Normally, LoadLeveler processes will be able to read LoadLeveler configuration data from the shared memory segment. If the shared memory segment does not already exist, the configuration server specified by the **LoadLConfigHosts** statement must be contacted to obtain the configuation data to store in the shared memory segment. In this situation, if the configuration servers cannot be contacted, the LoadLeveler process will fail.

The shared memory segment is usually created when the first LoadLeveler process runs, and is subsequently available for use by other LoadLeveler processes. So normally you can expect to run LoadLeveler processes on remotely configured nodes. The shared memory segment can be removed by issuing the **llrctl rmshm** command. The LoadLeveler shared memory segment is also removed when uninstalling LoadLeveler and when installing updates to the LoadLeveler software.

There is one command option which always relies on the availability of a configuration server. The **llrctl reconfig** command depends on the configuration server regardless of whether a shared memory segment exists or not. The **reconfig** option must access the most current database configuration data in order to refresh the configuration data in the shared memory segment. If a configuration server cannot be contacted in this situation, the command cannot perform its function.

## Using the configuration editor

To make it easier to view, update, and maintain the LoadLeveler configuration using the database option, a form-based configuration editor is provided for administrators. This tool allows the administrator to modify configuration attributes, make updates into the configuration database, and invoke reconfiguration for just the nodes that need to pick up the changed configuration.

In the database, the configuration is kept in several tables and the configuration editor groups input for the tables to make it easier for the administrator to set up and update the configuration. The editor consists of several tabbed panels containing forms that present one or more configuration database tables and a search panel to help the administrator find the panel in which a keyword can be updated.

For information about setting up the configuration editor, see the “Software requirements” topic in the *LoadLeveler for AIX: Installation Guide* or the *LoadLeveler for Linux: Installation Guide*.

To invoke the editor, point your browser to the following URL:

http://your_server_machine/ll/llconfig_editor.pl

where your_server_machine is the node where you are running the HTTP Server.

You will need to login to the configuration editor using a database ID and password with access to make changes in the database.

To use the editor, update the fields in the forms and click the **Add**, **Update**, or **Delete** buttons to update the configuration change in the database. After you have completed all the changes you will make for the session, make sure that you click the **Reconfigure** button to reconfigure LoadLeveler so the changes are effective.

If you cannot find a keyword you want to update, use the Search panel to find the panel containing a keyword. You may enter all or part of the keyword you are looking for.

## Modifying configuration data

By taking a look at the configuration files that come with LoadLeveler, you will find that there are many parameters that you can set. In most cases, you will only have to modify a few of these parameters.

In some cases, though, depending upon the LoadLeveler nodes, network connection, and hardware availability, you may need to modify additional parameters.

All LoadLeveler commands, daemons, and processes read the configuration data at start up time. To ensure that the configuration for all LoadLeveler commands, daemons, and processes are in sync, run the reconfiguration command on all machines in the cluster each time the configuration changes.

### Defining LoadLeveler administrators

Specify the **LOADL_ADMIN** keyword with a list of user names of those individuals who will have administrative authority.

These users are able to invoke the administrator-only commands such as **llrctl**.

LoadLeveler administrators on this list also receive mail describing problems that are encountered by the master daemon. When CtSec is enabled, the **LOADL_ADMIN** list is used only as a mailing list. For more information, see "Defining security mechanisms" on page 39.

When using the database configuration option, the list of administrators specified by **LOADL_ADMIN** applies to the entire cluster; however, for file-based configuration, it applies on a node-by-node basis. When using file-based configuration, you can override the list of administrators defined in the global configuration file by specifying **LOADL_ADMIN** in the local configuration file for a node.

When using the database configuration option, the list of administrators applies to all machines in a cluster.

For information about configuration keyword syntax and other details, see Chapter 6, "Configuration keyword reference," on page 91.

### Defining a LoadLeveler cluster

It will be necessary to define the characteristics of the LoadLeveler cluster.

Table 7 on page 30 lists the topics that discuss how you can define the characteristics of the LoadLeveler cluster.

*Table 7. Roadmap for defining LoadLeveler cluster characteristics*

| To learn about: | Read the following: |
|---|---|
| Defining other cluster characteristics | • "Defining network characteristics"<br>• "Specifying file and directory locations"<br>• "Configuring recording activity and log files" on page 32<br>• "Setting up file system monitoring" on page 38 |
| Correctly specifying configuration keywords | Chapter 6, "Configuration keyword reference," on page 91 |

For information on working on with daemons and machines in a LoadLeveler cluster, see the **llrctl** command in *LoadLeveler: Command and API Reference*.

## Specifying alternate resource managers

In a keyword statement in the configuration file or database, you specified which machine would serve as the resource manager. You can also assign one or more alternate resource managers in case network communication, software, or hardware failures make the primary resource manager unusable; to do so, ensure that your keyword statement in the configuration file or configuration database includes a list of alternate resource managers:

```
RESOURCE_MGR_LIST = primary_resource_manager_machine \
                    [alternate_resource_manager_machine_list]
```

If the primary resource manager fails, the alternate resource manager then becomes the resource manager. The alternate resource manager is chosen based upon the order in which its name appears in the alternate resource manager machine list specified in the configuration file or database.

For information about configuration keyword syntax and other details, see Chapter 6, "Configuration keyword reference," on page 91.

## Defining network characteristics

A **port number** is an integer that specifies the port to use to connect to the specified daemon. You can define these port numbers in the configuration keyword or the **/etc/services** file or you can accept the defaults. LoadLeveler first looks in the configuration for these port numbers. If LoadLeveler does not find the value in the configuration, it looks in the **/etc/services** file. If the value is not found in this file, the default is used.

See Appendix E, "LoadLeveler port usage," on page 329 for more information.

## Specifying file and directory locations

The sample configuration file provided with LoadLeveler specifies default locations for all of the files and directories. You can modify their locations using the keywords shown in Table 8 on page 31. Keep in mind that the LoadLeveler installation process installs files in these directories and these files may be periodically cleaned up. Therefore, you should not keep any files that do not belong to LoadLeveler in these directories.

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

*Table 8. Default locations for all of the files and directories*

| To specify the location of the: | Specify this keyword: |
|---|---|
| Administration file | **ADMIN_FILE** |
| Local configuration file | **LOCAL_CONFIG** |
| Local directory | The following subdirectories reside in the local directory. It is possible that the local directory and LoadLeveler's home directory are the same.<br>• **COMM**<br>• **EXECUTE**<br>• **LOG**<br>• **SPOOL** and **HISTORY**<br><br>**Tip:** To maximize performance, you should keep the log, spool, and execute directories in a local file system. Also, to measure the performance of your network, consider using one of the available products, such as Toolbox/6000. On clusters with diskless nodes, the execute directory should be located in RAM disk. |
| Release directory | When a LoadLeveler daemon or process needs to start another command or process, it will use the following keywords to locate the appropriate binary. If you change the location of the LoadLeveler release directory you must change these keywords to point to the new location.<br><br>`RELEASEDIR`<br>Is the directory created during installation to contains the **bin**, **lib**, and **include** directories for the LoadLeveler code.<br><br>`BIN`<br>Is a subdirectory in the release directory that is created during installation for the binaries for the LoadLeveler daemons and commands. |
| Core dump directory | You may specify alternate directories to hold core dumps for the daemons and starter process:<br>• **MASTER_COREDUMP_DIR**<br>• **RESOURCE_MGR_COREDUMP_DIR**<br>• **REGION_MGR_COREDUMP_DIR**<br>• **SCHEDD_COREDUMP_DIR**<br>• **STARTD_COREDUMP_DIR**<br>• **STARTER_COREDUMP_DIR**<br>• **KBDD_COREDUMP_DIR**<br><br>When specifying core dump directories, be sure that the access permissions are set so the LoadLeveler daemon or process can write to the core dump directory. The permissions set for path names specified in the keywords just mentioned must allow writing by both root and the LoadLeveler ID. The permissions set for the path name specified for the STARTER_COREDUMP_DIR keyword must allow writing by root, the LoadLeveler ID, and any user who can submit LoadLeveler jobs.<br><br>The simplest way to be sure the access permissions are set correctly is to set them the same as are set for the **/tmp** directory.<br><br>If a problem with access permissions prevents a LoadLeveler daemon or process from writing to a core dump directory, then a message will be written to the log, and the daemon or process will continue using the default **/tmp** directory for core files. |

For information about configuration keyword syntax and other details, see Chapter 6, “Configuration keyword reference,” on page 91.

## Locating LoadLeveler binaries when the scheduler component is not installed

If your cluster will only run the resource manager or it will include nodes where only the resource manager component is installed, you will need to make changes to your configuration for locating the LoadLeveler binaries. The **RELEASEDIR** and **BIN** keywords are used for defining locations.

If the cluster will only run the LoadLeveler resource manager and if you use the sample file **LoadL_config**, then do *one* of the following:

Change:

```
RELEASEDIR  = /usr/lpp/LoadL/full
```

to

```
RELEASEDIR  = /usr/lpp/LoadL/resmgr/full
```

*or:*

Create the following link on each node:

```
ln -s /usr/lpp/LoadL/resmgr/full /usr/lpp/LoadL/full
```

*or:*

Run the **llrinit** command on each of the nodes to create the link.

If the cluster consists of both resource manager and scheduler nodes, but some nodes have just the resource manager component installed, use the **LoadL_config.l** sample file to create your configuration. This sample uses **RELEASEDIR= /usr/lpp/LoadL/resmgr/full** and specifies the scheduler full path for those binaries that need it.

## Configuring recording activity and log files

The LoadLeveler daemons and processes keep log files according to the specifications in the configuration file or database. Administrators can also configure the LoadLeveler daemons to store additional debugging messages in a circular buffer in memory. A number of keywords are used to describe where LoadLeveler maintains the logs and how much information is recorded in each log and buffer. These keywords, shown in Table 9 on page 33, are repeated in similar form to specify the path name of the log file, its maximum length, the size of the circular buffer, and the debug flags to be used for the log and the buffer.

“Controlling the logging buffer” on page 34 describes how administrators can configure LoadLeveler to buffer debugging messages.

“Controlling debugging output” on page 35 describes the events that can be reported through logging controls.

“Saving log files” on page 38 describes the configuration keyword to use to save logs for problem diagnosis.

For information about configuration keyword syntax and other details, see Chapter 6, “Configuration keyword reference,” on page 91.

*Table 9. Log control statements*

| **Daemon/ Process** | **Log File** *(required)* **(See note 1)** | **Max Length** *(required)* **(See note 3)** | **Debug Control** *(required)* **(See note 5 on page 34)** |
|---|---|---|---|
| **Master** | **MASTER_LOG =** *path* | **MAX_MASTER_LOG =** *bytes* [*buffer bytes*] | **MASTER_DEBUG =** *flags* [*buffer flags*] |
| **Schedd** | **SCHEDD_LOG =** *path* | **MAX_SCHEDD_LOG =** *bytes* [*buffer bytes*] | **SCHEDD_DEBUG =** *flags* [*buffer flags*] |
| **Startd** | **STARTD_LOG =** *path* | **MAX_STARTD_LOG =** *bytes* [*buffer bytes*] | **STARTD_DEBUG =** *flags* [*buffer flags*] |
| **Starter** | **STARTER_LOG** = *path* | **MAX_STARTER_LOG =** *bytes* [*buffer bytes*] | **STARTER_DEBUG =** *flags* [*buffer flags*] |
| **Kbdd** | **KBDD_LOG =** *path* | **MAX_KBDD_LOG =** *bytes* [*buffer bytes*] | **KBDD_DEBUG =** *flags* [*buffer flags*] |
| **Region_mgr** | **REGION_MGR_LOG =** *path* | **MAX_REGION_MGR_LOG =** *bytes* [*buffer bytes*] | **REGION_MGR_DEBUG =** *flags* [*buffer flags*] |
| **Resource_mgr** | **RESOURCE_MGR_LOG** = *path* | **MAX_RESOURCE_MGR_LOG =** *bytes* [*buffer bytes*] | **RESOURCE_MGR_DEBUG =** *flags* [*buffer flags*] |

where:

*buffer bytes*
: Is the size of the circular buffer. The default value is 0, which indicates that the buffer is disabled. To prevent the daemon from running out of memory, this value should not be too large. Brackets must be used to specify buffer bytes.

*buffer flags*
: Indicates that messages with *buffer flags* in addition to messages with *flags* will be stored in the circular buffer in memory. The default value is blank, which indicates that the logging buffer is disabled because no additional debug flags were specified for buffering. Brackets must be used to specify buffer flags.

**Notes:**

1. When coding the *path* for the log files, it is not necessary that all LoadLeveler daemons keep their log files in the same directory, however, you will probably find it a convenient arrangement.
2. When the database configuration option is used, the flags and buffer flags strings are limited to 255 characters. Flags or buffer flags strings longer than 255 characters will be truncated.
3. There is a maximum length, in bytes, beyond which the various log files cannot grow. Each file is allowed to grow to the specified length and is then saved to an **.old** file. The **.old** files are overwritten each time the log is saved, thus the maximum space devoted to logging for any one program will be twice the maximum length of its log file. The default length is 64 KB. To obtain records over a longer period of time, that do not get overwritten, you can use the **SAVELOGS** keyword. See "Saving log files" on page 38 for more information on extended capturing of LoadLeveler logs.

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

- **TRUNC_SCHEDD_LOG_ON_OPEN =** true|**false**
- **TRUNC_KBDD_LOG_ON_OPEN =** true|**false**
- **TRUNC_STARTER_LOG_ON_OPEN =** true|**false**
- **TRUNC_REGION_MGR_LOG_ON_OPEN =** true|**false**
- **TRUNC_RESOURCE_MGR_LOG_ON_OPEN =** true|**false**

4. LoadLeveler creates temporary log files used by the **starter** daemon. These files are used for synchronization purposes. When a job starts, a **StarterLog.***pid* file is created. When the job ends, this file is appended to the **StarterLog** file.
5. Normally, only those who are installing or debugging LoadLeveler will need to use the debug flags, described in "Controlling debugging output" on page 35. The default error logging, obtained by leaving the right side of the debug control statement null, will be sufficient for most installations.

**Controlling the logging buffer:**
LoadLeveler allows a LoadLeveler daemon to store log messages in a buffer in memory instead of writing the messages to a log file. The administrator can force the messages in this buffer to be written to the log file, when necessary, to diagnose a problem. The buffer is circular and once it is full, older messages are discarded as new messages are logged. The **llrctl dumplogs** command is used to write the contents of the logging buffer or locking records to the appropriate log file for the **Master**, **Schedd**, **Startd**, **Resource Manager**, and **Region Manager** daemons.

Buffering will be disabled if either the buffer length is 0 or no additional debug flags are specified for buffering.

See "Configuring recording activity and log files" on page 32 for log control statement specifications. See *LoadLeveler: Diagnosis and Messages Guide* for additional information on LoadLeveler log files.

**Logging buffer example:**

With the configuration keyword **SCHEDD_LOG = D_SCHEDD [D_FULLDEBUG]**, the **Schedd** daemon will write only **D_ALWAYS** and **D_SCHEDD** messages to the ${LOG}/SchedLog log file. The following messages will be stored in the buffer:
- **D_ALWAYS**
- **D_SCHEDD**
- **D_FULLDEBUG**

The maximum size of the Schedd log is 64 MB and the size of the logging buffer is 32 MB.

```
SCHEDD_LOG = ${LOG}/SchedLog
MAX_SCHEDD_LOG = 64000000 [32000000]
SCHEDD_DEBUG = D_SCHEDD [D_FULLDEBUG]
```

To write the contents of the logging buffer to SchedLog file on the machine, issue

```
llrctl dumplogs buffer
```

To write the contents of the logging buffer to the SchedLog file on **node1** in the LoadLeveler cluster, issue:

```
llrctl -h node1 dumplogs buffer
```

To write the contents of the logging buffers to the SchedLog files on all machines, issue:

```
llrctl -g dumplogs buffer
```

Note that the messages written from the logging buffer include a bracket message and a prefix to identify them easily.

```
======================BUFFER BEGIN=======================

BUFFER: message .....
BUFFER: message .....

======================BUFFER END=========================
```

**Controlling locking records:**

**Note:** This topic is for your information only because it is used primarily by IBM personnel for debugging purposes.

To turn on the locking function of a daemon, specify the **D_LOCK_TRACE** debugging flag.

**Note: D_LOCK_TRACE** messages are not handled in the same way as other debug flags are handled. Rather than being printed to the logs as they occur, these messages are kept in memory and updated when there are locking events like requesting a lock or releasing a lock. Use **llrctl dumplogs locks** to print messages to the logs about locks currently pending or held.

**Locking records examples:**

To save the locking records in the LoadLeveler daemons to the corresponding existing log files in the LOG directory of node c197blade7b02, issue:

```
llrctl -h c197blade7b02 dumplogs locks
```

With the following configuration, the locking records of the Schedd daemon will be stored:

```
SCHEDD_DEBUG = D_LOCK_TRACE
```

To write the locking records to the SchedLog file on the machine, issue:

```
llrctl dumplogs lock
```

To write the locking records to the SchedLog file on node1 in the LoadLeveler cluster, issue:

```
llrctl –h node1 dumplogs lock
```

Note that the locking records are bracketed as shown in the following example to identify them easily.

```
============ LOCKING RECORDS BEGIN ============

LOCKID   TID  STAT  TYPE   SC   TIMESTAMP  LINE FUNCTION | DESCRIPTION
                                                           ...
                                                           ...
============ LOCKING RECORDS END ============
```

**Controlling debugging output:**
You can control the level of debugging output logged by LoadLeveler programs.

The following flags are presented here for your information, though they are used primarily by IBM personnel for debugging purposes:

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

**D_ACCOUNT_DETAIL**
Logs detailed accounting information about processes. If used, it may slow down the network and increase the size of log files.

**D_ADAPTER**
Logs messages related to adapters.

**D_AFS**
Logs information related to AFS credentials.

**D_CKPT**
Logs information related to checkpoint and restart.

**D_CONFIG**
Displays messages detailing configuration processing during the start up or reconfiguration of a LoadLeveler process.

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

**D_DATABASE**
Logs information pertaining to database interactions.

**D_DISPATCHING_CYCLE**
Traces the dispatching cycle from the cluster's startup. This debug flag can be specified only in the **TRACE** keyword (see the **TRACE** keyword on page 119 for more information).

**D_EXPR**
Logs steps in parsing and evaluating control expressions.

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

**D_HIERARCHICAL**
Used to enable messages relating to problems related to the transmission of hierarchical messages. A hierarchical message is sent from an originating node to lower ranked receiving nodes.

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

**D_JOB_LIFECYCLE**
Enables the administrator to grant requests for job lifecycle traces. This debug flag is only used by the **TRACE** keyword (see the **TRACE** keyword on page 119 for more information).

**D_KERNEL**
Activates diagnostics for errors involving the process tracking kernel extension.

**D_LOAD**
Displays the load average on the startd machine.

**D_LOCKING**
Logs requests to acquire and release locks. This keyword is deprecated in favor of the **D_LOCK_TRACE** keyword.

**D_LOCK_TRACE**
Activates the lock tracing function, which maintains a list of locking events in memory. Only events for locks not released are kept.

The **llrctl dumplogs locks** command is used to move the locking records from memory into the logs (see "llrctl - Control LoadLeveler daemons" on page 166 for more information).

**D_LXCPUAFNT**
Logs messages related to Linux CPU affinity. This flag is only valid for the **startd** daemon.

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

**D_ODBC_DETAIL**
: Logs information pertaining to the ODBC interface.

**D_PCRED**
: Directs that extra debug should be written to a file if the `setpcred()` function call fails.

**D_PERF**
: Logs performance-related information in LoadLeveler daemon logs. Turning on **D_PERF** will produce microsecond time stamps for each entry in the daemon log.

**D_PERF_DETAIL**
: Logs detailed performance-related information in LoadLeveler daemon logs. Turning on **D_PERF_DETAIL** will produce microsecond time stamps for each entry in the daemon log.

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

**D_QUEUE**
: Logs changes to the job queue.

**D_REFCOUNT**
: Logs activity associated with reference counting of internal LoadLeveler objects.

**D_REGIONMGR**
: Displays how the region manager works internally.

**D_SCHEDD**
: Displays how the Schedd works internally.

**D_SDO**
: Displays messages detailing LoadLeveler objects being transmitted between daemons and commands.

**D_SECURITY**
: Logs information related to Cluster Security (CtSec) services identities.

**D_SPOOL**
: Logs information related to the usage of databases in the LoadLeveler **spool** directory.

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

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

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

**D_STREAM**
: Displays messages detailing socket I/O.

**D_SWITCH**
: Logs entries related to switch activity and LoadLeveler Switch Table Object data.

**D_THREAD**
: Displays the ID of the thread producing the log message. The thread ID is displayed immediately following the date and time. This flag is useful for debugging threaded daemons.

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

For example:

```
SCHEDD_DEBUG = D_SCHEDD  D_XDR
```

Causes the job manager to log information about its processing of jobs and the exchanging of xdr messages with other LoadLeveler daemons. These flags will primarily be of interest to LoadLeveler implementers and debuggers.

The **LL_COMMAND_DEBUG** environment variable can be set to a string of debug flags the same way as the ***_DEBUG** configuration keywords are set. Normally, LoadLeveler commands and APIs do not print debug messages, but with this environment variable set, the requested classes of debugging messages will be logged to stderr. For example:

```
LL_COMMAND_DEBUG="D_ALWAYS D_STREAM" llrstatus
```

will cause the **llrstatus** command to print out debug messages related to I/O to stderr.

**Saving log files:** By default, LoadLeveler stores only the two most recent iterations of a daemon's log file (<*daemon name*>Log, and <*daemon name*>Log.old).

Occasionally, for problem diagnosing, users will need to capture LoadLeveler logs over an extended period. Users can specify that all log files be saved to a particular directory by using the **SAVELOGS** keyword. Be aware that LoadLeveler does not provide any way to manage and clean out all of those log files, so users must be sure to specify a directory in a file system with enough space to accommodate them. This file system should be separate from the one used for the LoadLeveler log, spool, and execute directories.

Each log file is represented by the name of the daemon that generated it, the exact time the file was generated, and the name of the machine on which the daemon is running. When you list the contents of the **SAVELOGS** directory, the list of log file names looks like this:

```
StarterLogNov02.16:09:19.622387.c163n10.ppd.pok.ibm.com
StarterLogNov02.16:09:51.499823.c163n10.ppd.pok.ibm.com
StarterLogNov02.16:10:30.876546.c163n10.ppd.pok.ibm.com
SchedLogNov02.16:09:05.543677.c163n10.ppd.pok.ibm.com
SchedLogNov02.16:09:26.688901.c163n10.ppd.pok.ibm.com
SchedLogNov02.16:09:47.443556.c163n10.ppd.pok.ibm.com
SchedLogNov02.16:10:12.712680.c163n10.ppd.pok.ibm.com
SchedLogNov02.16:10:37.342156.c163n10.ppd.pok.ibm.com
StartLogNov02.16:09:05.697753.c163n10.ppd.pok.ibm.com
StartLogNov02.16:09:26.881234.c163n10.ppd.pok.ibm.com
StartLogNov02.16:09:47.231234.c163n10.ppd.pok.ibm.com
StartLogNov02.16:10:12.125556.c163n10.ppd.pok.ibm.com
StartLogNov02.16:10:37.961486.c163n10.ppd.pok.ibm.com
```

For information about configuration keyword syntax and other details, see Chapter 6, “Configuration keyword reference,” on page 91.

## Setting up file system monitoring

You can use the file system keywords to monitor the file system space or inodes used by LoadLeveler for:

- Logs
- Saving executables
- Spool information
- History files

You can also use the file system keywords to take preventive action and avoid problems caused by running out of file system space or inodes. This is done by setting the frequency that LoadLeveler checks the file system free space or inodes and by setting the upper and lower thresholds that initialize system responses to the free space available. By setting a realistic span between the lower and upper thresholds, you will avoid excessive system actions.

The file system monitoring keywords are:
- **FS_INTERVAL**
- **FS_NOTIFY**
- **FS_SUSPEND**
- **FS_TERMINATE**
- **INODE_NOTIFY**
- **INODE_SUSPEND**
- **INODE_TERMINATE**

For information about configuration keyword syntax and other details, see Chapter 6, "Configuration keyword reference," on page 91.

## Defining LoadLeveler machine characteristics

You can use these keywords to define the characteristics of machines in the LoadLeveler cluster.

The following keywords are used by the resource manager during startup:
- **SCHEDD_RUNS_HERE**
- **SCHEDD_SUBMIT_AFFINITY**
- **STARTD_RUNS_HERE**
- **START_DAEMONS**
- **X_RUNS_HERE**

The following keywords are not used by the resource manager, but are made available to the scheduling application:
- **ARCH**
- **CLASS**
- **FEATURE**

For information about configuration keyword syntax and other details, see Chapter 6, "Configuration keyword reference," on page 91.

## Defining security mechanisms

LoadLeveler can be configured to control authentication and authorization of LoadLeveler functions by using Cluster Security (CtSec) services, a subcomponent of Reliable Scalable Cluster Technology (RSCT), which uses the host-based authentication (HBA) as an underlying security mechanism.

LoadLeveler permits only one security service to be configured at a time. You can skip this topic if you do not plan to use this security feature or if you plan to use the credential forwarding provided by the **llgetdce** and **llsetdce** program pair. Refer to "Using the alternative program pair: llgetdce and llsetdce" on page 50 for more information.

LoadLeveler for Linux does not support CtSec security.

Table 10 on page 40 lists the topics that explain LoadLeveler daemons and how you may define their characteristics and modify their behavior.

Table 10. Roadmap of configuration tasks for securing LoadLeveler operations

| To learn about: | Read the following: |
|---|---|
| Securing LoadLeveler operations using cluster security services | • "Configuring LoadLeveler to use cluster security services" <br> • "Steps for enabling CtSec services" on page 41 <br> • "Limiting which security mechanisms LoadLeveler can use" on page 43 |
| Correctly specifying configuration keywords | Chapter 6, "Configuration keyword reference," on page 91 |

## Configuring LoadLeveler to use cluster security services

Cluster security (CtSec) services allows a software component to authenticate and authorize the identity of one of its peers or clients. When configured to use CtSec services, LoadLeveler will:

- Authenticate the identity of users and programs interacting with LoadLeveler.
- Authorize users and programs to use LoadLeveler services. It prevents unauthorized users and programs from misusing resources or disrupting services.

To use CtSec services, all nodes running LoadLeveler must first be configured as part of a cluster running Reliable Scalable Cluster Technology (RSCT). For details on CtSec services administration, see *IBM Reliable Scalable Cluster Technology: Administration Guide*, SA22-7889.

CtSec services are designed to use multiple security mechanisms and each security mechanism must be configured for LoadLeveler. At the present time, directions are provided only for configuring the host-based authentication (HBA) security mechanism for LoadLeveler's use. If CtSec is configured to use additional security mechanisms that are not configured for LoadLeveler's use, then the LoadLeveler configuration keyword **SEC_IMPOSED_MECHS** must be specified. This keyword is used to limit the security mechanisms that will be used by CtSec services to only those that are configured for use by LoadLeveler.

Authorization is based on user identity. When CtSec services are enabled for LoadLeveler, user identity will differ depending on the authentication mechanism in use. A user's identity in UNIX host-based authentication is the user's network identity which is comprised of the user name and host name, such as user_name@host.

LoadLeveler uses CtSec services to authorize owners of jobs, administrators and LoadLeveler daemons to perform certain actions. CtSec services uses its own identity mapping file to map the clients' network identity to a local identity when performing authorizations. A typical local identity is the user name without a hostname. The local identities of the LoadLeveler administrators must be added as members of the group specified by the keyword **SEC_ADMIN_GROUP**. The local identity of the LoadLeveler user name must be added as the sole member of the group specified by the keyword **SEC_SERVICES_GROUP**. The LoadLeveler Services and Administrative groups, those identified by the keywords **SEC_SERVICES_GROUP** and **SEC_ADMIN_GROUP**, must be the same across all nodes in the LoadLeveler cluster. To ensure consistency in performing tasks which require owner, administrative or daemon privileges across all nodes in the

LoadLeveler cluster, user network identities must be mapped identically across all nodes in the LoadLeveler cluster. If this is not the case, LoadLeveler authorizations may fail.

**Steps for enabling CtSec services:**
It is necessary to enable LoadLeveler to use CtSec services. To enable LoadLeveler to use CtSec services, perform the following steps:

1. Include, in the Trusted Host List, the host names of all hosts with which communications may take place. If LoadLeveler tries to communicate with a host not on the Trusted Host List the message: The host identified in the credentials is not a trusted host on this system will occur. Additionally, the system administrator should ensure that public keys are manually exchanged between all hosts in the LoadLeveler cluster. Refer to *IBM Reliable Scalable Cluster Technology: Administration Guide*, SA22-7889 for information on setting up Trusted Host Lists and manually transferring public keys.
2. Create user IDs. Each LoadLeveler administrator and the LoadLeveler user ID need to be created, if they don't already exist. You can do this through SMIT or the **mkuser** command.
3. Ensure that the **unix.map** file contains the correct value for the service name **ctloadl** which specifies the LoadLeveler user name. If you have configured LoadLeveler to use **loadl** as the LoadLeveler user name, either by default or by specifying **loadl** in the **LoadLUserid** keyword of the **/etc/LoadL.cfg** file, nothing needs to be done. The default map file will contain the **ctloadl** service name already assigned to **loadl**. If you have configured a different user name in the **LoadLUserid** keyword of the **/etc/LoadL.cfg** file, you will need to make sure that the **/var/ct/cfg/unix.map** file exists and that it assigns the same user name to the **ctloadl** service name. If the **/var/ct/cfg/unix.map** file does not exist, create one by copying the default map file **/usr/sbin/rsct/cfg/unix.map**. Do not modify the default map file.

   If the value of the **LoadLUserid** and the value associated with **ctloadl** are not the same a security services error which indicates a UNIX identity mismatch will occur.
4. Add entries to the global mapping file of each machine in the LoadLeveler cluster to map network identities to local identities. This file is located at: **/var/ct/cfg/ctsec_map.global**. If this file doesn't yet exist, you should copy the default global mapping file to this location—don't modify the default mapping file. The default global mapping file, which is shared among all CtSec services exploiters, is located at **/usr/sbin/rsct/cfg/ctsec_map.global**. See *IBM Reliable Scalable Cluster Technology for AIX: Technical Reference*, SA22-78900 for more information on the mapping file.

   When adding names to the global mapping file, enter more specific entries ahead of the other, less specific entries. Remember that you must update the global mapping file on each machine in the LoadLeveler cluster, and each mapping file has to be updated with the security services identity of each member of the **administrator** group, the **services** group, and the users. Therefore, you would have entries like this:

   ```
   unix:brad@mach1.pok.ibm.com=bradleyf
   unix:brad@mach2.pok.ibm.com=bradleyf
   unix:brad@mach3.pok.ibm.com=bradleyf
   unix:marsha@mach2.pok.ibm.com=marshab
   unix:marsha@mach3.pok.ibm.com=marshab
   unix:loadl@mach1.pok.ibm.com=loadl
   unix:loadl@mach2.pok.ibm.com=loadl
   unix:loadl@mach3.pok.ibm.com=loadl
   ```

However, if you're sure your LoadLeveler cluster is secure, you could specify mapping for all machines this way:

```
unix:brad@*=bradleyf
unix:marsha@*=marshab
unix:loadl@*=loadl
```

This indicates that the UNIX network identity of the users **brad**, **marsha** and **loadl** will map to their respective security services identities on every machine in the cluster. Refer to *IBM Reliable Scalable Cluster Technology for AIX: Technical Reference*, SA22-7800 for a description of the syntax used in the **ctsec_map.global** file.

5. Create UNIX groups. The LoadLeveler **administrator** group and **services** group need to be created for every machine in the cluster and should contain the local identities of members. This can be done either by using SMIT or the **mkgroup** command.

   For example, to create the group lladmin which lists the LoadLeveler administrators:

   ```
   mkgroup "users=sam,betty,loadl" lladmin
   ```

   These groups must be created on each machine in the LoadLeveler cluster and must contain the same entries.

   To create the group llsvcs which lists the identity under which LoadLeveler daemons run using the default id of loadl:

   ```
   mkgroup users=loadl llsvcs
   ```

   These groups must be created on each machine in the LoadLeveler cluster and must contain the same entries.

6. Add or update these keywords in the LoadLeveler configuration:

   ```
   SEC_ENABLEMENT=CTSEC
   SEC_ADMIN_GROUP=name of lladmin group
   SEC_SERVICES_GROUP=group name that contains identities of LoadLeveler daemons
   ```

   The **SEC_ENABLEMENT=CTSEC** keyword indicates that CtSec services mechanism should be used. **SEC_ADMIN_GROUP** points to the name of the UNIX group which contains the local identities of the LoadLeveler administrators. The **SEC_SERVICES_GROUP** keyword points to the name of the UNIX group which contains the local identity of the LoadLeveler daemons. All LoadLeveler daemons run as the LoadLeveler user ID. Refer to step 5 for discussion of the **administrators** and **services** groups.

7. Update the **.rhosts** file in each user's home directory. This file is used to identify which UNIX identities can run LoadLeveler jobs on the local host machine. If the file does not exist in a user's home directory, you must create it. The **.rhosts** file must contain entries which specify all host and user combinations allowed to submit jobs which will run as the local user, either explicitly or through the use of wildcards.

   Entries in the **.rhosts** file are specified this way:

   ```
   HostNameField [UserNameField]
   ```

   Refer to *IBM AIX Files Reference*, SC23-4168 for further details about the **.rhosts** file format.

*Tips for configuring LoadLeveler to use CtSec services:* When using CtSec services for LoadLeveler, each machine in the LoadLeveler cluster must be set up properly.

CtSec authenticates network identities based on trust established between individual machines in a cluster, based on local host configurations. Because of this it is possible for most of the cluster to run correctly but to have transactions from certain machines experience authentication or authorization problems.

If unexpected authentication or authorization problems occur in a LoadLeveler cluster with CtSec enabled, check that the steps in “Steps for enabling CtSec services” on page 41 were correctly followed for each machine in the LoadLeveler cluster.

If any machine in a LoadLeveler cluster is improperly configured to run CtSec you may see that:

- Users cannot perform user tasks (such as cancel) for jobs they submitted.

  Either the machine the job was submitted from or the machine the user operation was submitted from (or both) do not contain mapping files for the user that specify the same security services identity. The user should attempt the operation from the same machine the job was submitted from and record the results. If the user still cannot perform a user task on a job they submitted, then they should contact the LoadLeveler administrator who should review the steps in “Steps for enabling CtSec services” on page 41.
- LoadLeveler daemons fail to communicate.

  When LoadLeveler daemons communicate they must first authenticate each other. If the daemons cannot authenticate a message will be put in the daemon log indicating an authentication failure. Ensure the Trusted Hosts List on all LoadLeveler nodes contains the correct entries for all of the nodes in the LoadLeveler cluster. Also, make sure that the LoadLeveler Services group on all nodes of the LoadLeveler cluster contains the local identity for the LoadLeveler user name. The **ctsec_map.global** must contain mapping rules to map the LoadLeveler user name from every machine in the LoadLeveler cluster to the local identity for the LoadLeveler user name. An example of what may happen when daemons fail to communicate is that an alternate central manager may take over while the primary central manager is still active. This can occur when the alternate central manager does not trust the primary central manager.

**Limiting which security mechanisms LoadLeveler can use:** As more security mechanisms become available, they must be configured for LoadLeveler's use. If there are security mechanisms configured for CtSec that are not configured for LoadLeveler's use, then the LoadLeveler configuration keyword **SEC_IMPOSED_MECHS** must specify the mechanisms configured for LoadLeveler.

## Gathering job accounting data

Your organization may have a policy of charging users or groups of users for the amount of resources that their jobs consume.

You can do this using resource manager's accounting feature. Using this feature, you can produce accounting reports that contain job resource information for completed serial and parallel job steps.

The accounting record for a job step will contain separate sets of resource usage data for each time a job step is dispatched to run. For example, the accounting record for a job step that is vacated and then started again will contain two sets of resource usage data. The first set of resource usage data is for the time period when the job step was initially dispatched until the job step was vacated. The second set of resource usage data is for the time period for when the job step is dispatched after the vacate until the job step completes.

The following keywords allow you to control accounting functions:
- **ACCT**
- **GLOBAL_HISTORY**
- **HISTORY_PERMISSION**
- **JOB_ACCT_Q_POLICY**
- **JOB_LIMIT_POLICY**

For example, the following section of the configuration file specifies that the accounting function is turned on. It also identifies the directory containing the global history files:

```
ACCT              = A_ON
GLOBAL_HISTORY    = $(SPOOL)
```

Table 11 lists the topics related to configuring, gathering and using job accounting data.

*Table 11. Roadmap of tasks for gathering job accounting data*

| To learn about: | Read the following: |
|---|---|
| Configuring LoadLeveler to gather job accounting data | • "Collecting job resource data on serial and parallel jobs"<br>• "Collecting job resource data based on machines"<br>• "Collecting job resource data based on events" on page 45<br>• "Collecting the accounting information and storing it into files" on page 45<br>• Table 12 on page 47 |
| Managing accounting data | • "Reading accounting records" on page 46<br>• "Correlating AIX and LoadLeveler accounting records" on page 46<br>• "llracctmrg - Collect machine history files" on page 153 |
| Correctly specifying configuration keywords | Chapter 6, "Configuration keyword reference," on page 91 |

## Collecting job resource data on serial and parallel jobs

Information on completed serial and parallel job steps is gathered using the UNIX *wait3* system call. Information on non-completed serial and parallel jobs is gathered in a platform-dependent manner by examining data from the UNIX process.

Accounting information on a completed serial job step is determined by accumulating resources consumed by that job on the machines that ran the job. Similarly, accounting information on completed parallel job steps is gathered by accumulating resources used on all of the nodes that ran the job step.

## Collecting job resource data based on machines

LoadLeveler can collect job resource usage information for every machine on which a job may run. A job may run on more than one machine because it is a parallel job or because the job is vacated from one machine and rescheduled to another machine.

To enable recording of resources by machine, you need to specify **ACCT = A_ON A_DETAIL** in the configuration.

The machine's speed is part of the data collected. With this information, an installation can develop a charge back program which can charge more or less for resources consumed by a job on different machines. For more information on a machine's speed, refer to the machine stanza information. See "Defining machines" on page 61.

## Collecting job resource data based on events

In addition to collecting job resource information based upon machines used, you can gather this information based upon an event or time that you specify. For example, you may want to collect accounting information at the end of every work shift or at the end of every week or month. To collect accounting information on all machines in this manner, use the **llrctl** command with the **capture** parameter:

```
llrctl -g capture eventname
```

*eventname* is any string of continuous characters (no white space is allowed) that defines the event about which you are collecting accounting data. For example, if you were collecting accounting data on the *graveyard* work shift, your command could be:

```
llrctl -g capture graveyard
```

This command allows you to obtain a snapshot of the resources consumed by active jobs up to and including the moment when you issued the command. If you want to capture this type of information on a regular basis, you can set up a crontab entry to invoke this command regularly. For example:

```
#  sample crontab for accounting
#  shift crontab 94/8/5
#
# Set up three shifts, first, second, and graveyard shift.
#  Crontab entries indicate the end of shift.
#
#M  H d m day command
#
000 08 * *  * /usr/bin/llrctl -g capture graveyard
00 16 * *  * /usr/bin/llrctl -g capture first
00 00 * *  * /usr/bin/llrctl -g capture second
```

| For more information on the **llrctl** command, see "llrctl - Control LoadLeveler daemons" on page 166. For more information on the collection of accounting records, see the **llrq** command, see "llrq - Query job information" on page 179.

## Collecting job resource information based on user accounts

If your installation is interested in keeping track of resources used on an account basis, then all jobs submitted to the resource manager should specify an account number. The account number can be specified with the **account_no** keyword. Interactive POE jobs can specify an account number using the **LOADL_ACCOUNT_NO** environment variable.

## Collecting the accounting information and storing it into files

LoadLeveler resource manager stores the accounting information that it collects in a file called *history* in the spool directory of the machine that manages this job, the job manager machine. Data on parallel jobs are also stored in the *history* files.

Resource information collected on the job is constrained by the capabilities of the wait3 system call. Information for processes which fork child processes will include data for those child processes as long as the parent process waits for the child process to terminate. Complete data may not be collected for jobs which are not composed of simple parent/child processes. For example, if you have a job which invokes an rsh command to execute a function on another machine, the resources consumed on the other machine will not be collected as part of the LoadLeveler accounting data.

LoadLeveler accounting uses the following types of files:
- The local history file which is local to each job manager machine is where job resource information is first recorded. These files are usually named *history* and are located in the spool directory of each job manager machine, but you may specify an alternate name with the **HISTORY** keyword in either the global or local configuration file or the configuration database.
- The global history file is a combination of the history files from some or all of the machines in the LoadLeveler cluster merged together. The command **llracctmrg** is used to collect files together into a global file. As the files are collected from each machine, the local history file for that machine is reset to contain no data. The file is named *globalhist.YYYYMMDDHHmm*. You may specify the directory in which to place the file when you invoke the **llracctmrg** command or you can specify the directory with the **GLOBAL_HISTORY** keyword in the configuration file or database. The default value set up in the sample configuration file is the local spool directory.

## Reading accounting records

You can write an application using the resource manager query API or the **llr_get_history** API to read the accounting records.

## Correlating AIX and LoadLeveler accounting records

For jobs running on AIX systems, you can use a job accounting key to correlate AIX accounting records with LoadLeveler accounting records. The job accounting key uniquely identifies each job step. LoadLeveler derives this key from the job key and the date and time at which the job entered the queue. The key is associated with the starter process for the job step and any of its child processes.

For checkpointed jobs, LoadLeveler does not change the job accounting key, regardless of how it restarts the job step. Jobs restarted from a checkpoint file or through a new job step retain the job accounting key for the original job step.

To access the job accounting key for a job step, you can use the following interfaces:
- The **llr_get_history** subroutine. For details about using this subroutine, see the "llr_get_history subroutine" on page 224.
- The **llr_get_data** subroutine, through the **LLR_StepAcctKey** specification. For details about using this subroutine, see the "llr_get_data subroutine" on page 235.

For information about AIX accounting records, see the system accounting topic in *AIX System Management Guide: Operating System and Devices*.

## Example: Setting up job accounting files

You can perform all of the steps included in this sample procedure or just the ones that apply to your situation. The sample procedure shown in Table 12 on page 47 walks you through the process of collecting account data.

1. Edit the configuration keywords according to the following table:

*Table 12. Collecting account data - modifying configuration keywords*

| Edit this keyword: | To: |
|---|---|
| ACCT | Turn accounting and account validation on and off and specify detailed accounting. |
| GLOBAL_HISTORY | Specify a directory in which to place the global history files. |

2. Specify machines and their weights by using the **speed** keyword in a machine's machine stanza in the administration file.

   Also, if you have in your cluster machines of differing speeds and you want LoadLeveler accounting information to be normalized for these differences, specify **cpu_speed_scale=true** in each machine's respective machine stanza.

   For example, suppose you have a cluster of two machines, called A and B, where Machine B is three times as fast as Machine A. Machine A has **speed=1.0**, and Machine B has **speed=3.0**. Suppose a job runs for 12 CPU seconds on Machine A. The same job runs for 4 CPU seconds on Machine B. When you specify **cpu_speed_scale=true**, the accounting information collected on Machine B for that job shows the normalized value of 12 CPU seconds rather than the actual 4 CPU seconds.
3. Merge multiple files collected from each machine into one file, using the **llracctmrg** command.
4. Report job information on all the jobs in the history file, using the **llsummary** command. See the **llsummary** command in *LoadLeveler: Command and API Reference* for more information.

## Managing job status through control expressions

You can control running jobs by using five control functions as Boolean expressions in the configuration.

These functions are useful primarily for serial jobs. You define the expressions, using normal C conventions, with the following functions:
- START
- SUSPEND
- CONTINUE
- VACATE
- KILL

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

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

When evaluating the START expression to determine if the machine can accept jobs, **Class != "Z"** evaluates to true only if `Z` is not in the class definition. This means that if two different classes are defined on a machine, **Class != "Z"** (where `Z` is one of the defined classes) always evaluates to false when specified in the START expression and, therefore, the machine will not be considered to start jobs.

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

For additional information about:

- Time-related variables that you may use for this keyword, see "Variables to use for setting times" on page 127.
- Coding these control expressions in the configuration keywords, see Chapter 6, "Configuration keyword reference," on page 91.

## How control expressions affect jobs

After the resource manager starts a job, the job can be in any of several states. Figure 9 shows how the control expressions can affect the state a job is in. The rectangles represent job or daemon states (Idle, Completed, Running, Suspended, and Vacating) and the diamonds represent the control expressions (Start, Suspend, Continue, Vacate, and Kill).

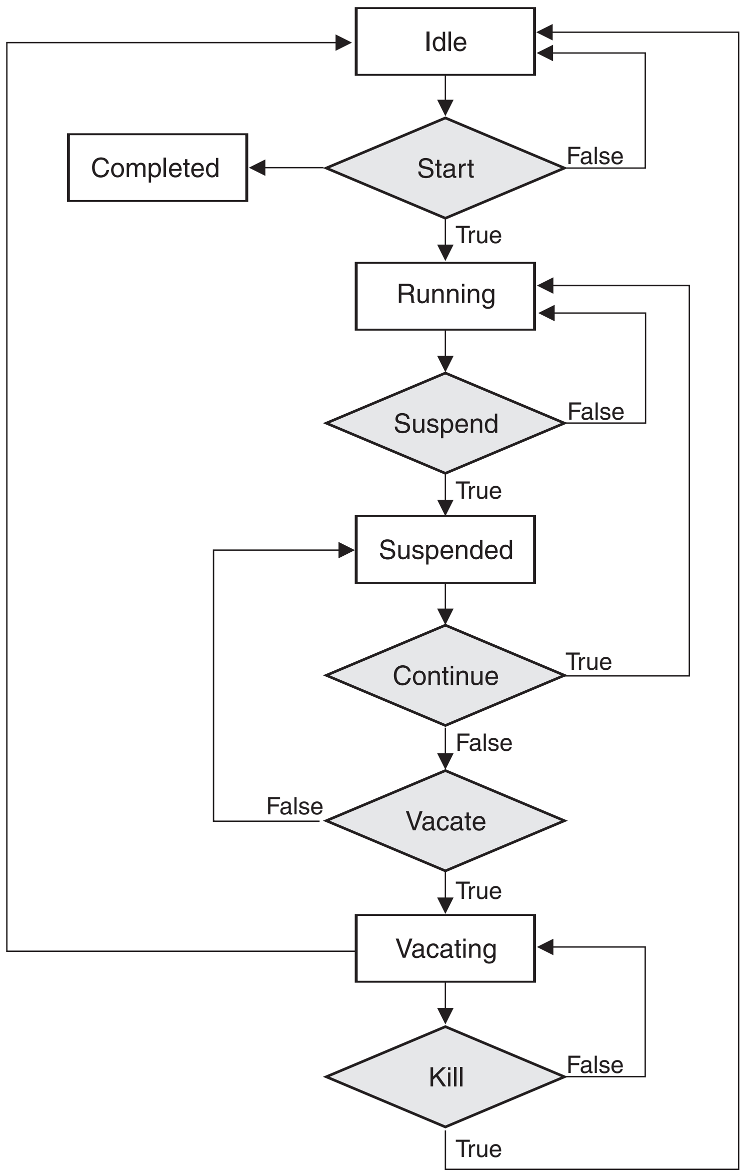

*Figure 9. How control expressions affect jobs*

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

## Tracking job processes

When a job terminates, its orphaned processes may continue to consume or hold resources, thereby degrading system performance, or causing jobs to hang or fail.

Process tracking allows the resource manager to cancel any processes (throughout the entire cluster), left behind when a job terminates. Process tracking is required to do preemption.

When process tracking is enabled, all child processes are terminated when the main process terminates. This will include any background or orphaned processes started in the prolog, epilog, user prolog, and user epilog.

Process tracking on LoadLeveler for Linux is supported on RHEL 6.2.

There are two keywords used in specifying process tracking:

**PROCESS_TRACKING**
: To activate process tracking, set **PROCESS_TRACKING=TRUE** in the LoadLeveler global configuration file or database. By default, **PROCESS_TRACKING** is set to **FALSE**.

**PROCESS_TRACKING_EXTENSION**
: This keyword is for AIX only. This keyword specifies the path to the loadable kernel module **LoadL_pt_ke** in the local or global configuration file or database. If the **PROCESS_TRACKING_EXTENSION** keyword is not supplied, then LoadLeveler will search the **$HOME/bin** default directory.

The process tracking kernel extension is not unloaded when the **startd** daemon terminates on systems running AIX. Therefore, if a mismatch in the version of the loaded kernel extension and the installed kernel extension is found when the **startd** starts up the daemon will exit. In this case, a reboot of the node is needed to unload the currently loaded kernel extension. If you install a new version of the LoadLeveler resource manager, which contains a new version of the kernel extension, you may need to reboot the node.

For information about configuration keyword syntax and other details, see Chapter 6, “Configuration keyword reference,” on page 91.

## Handling switch-table errors

Configuration keywords can be used to control how the resource manager responds to switch-table errors.

You may use the following configuration keywords to control how LoadLeveler responds to switch-table errors:
- ACTION_ON_SWITCH_TABLE_ERROR
- DRAIN_ON_SWITCH_TABLE_ERROR
- RESUME_ON_SWITCH_TABLE_ERROR_CLEAR

For information about configuration keyword syntax and other details, see Chapter 6, “Configuration keyword reference,” on page 91.

## Providing additional job-processing controls through installation exits

LoadLeveler allows administrators to further configure the environment through installation exits.

Table 13 lists these additional job-processing controls.

*Table 13. Roadmap of administrator tasks accomplished through installation exits*

| To learn about: | Read the following: |
|---|---|
| Writing a pair of programs to override the default LoadLeveler DCE authentication method | "Handling DCE security credentials" |
| Writing a program to refresh an AFS token when a job starts | "Handling an AFS token" on page 51 |
| Writing programs to run before and after job requests | "Writing prolog and epilog programs" on page 52 |
| Overriding the LoadLeveler default mail notification method | "Using your own mail program" on page 57 |
| Correctly specifying configuration keywords | Chapter 6, "Configuration keyword reference," on page 91 |
| Writing an installation exit that can determine the frequency to use to run a job | "Determining the frequency to use to run a job" on page 57 |

### Handling DCE security credentials

You can write a pair of programs to override the default LoadLeveler DCE authentication method. To enable the programs, use the **DCE_AUTHENTICATION_PAIR** keyword in your configuration file or database:

- As an alternative, you can also specify the program pair:

```
DCE_AUTHENTICATION_PAIR = $(BIN)/llgetdce, $(BIN)/llsetdce
```

Specifying the **DCE_AUTHENTICATION_PAIR** keyword enables LoadLeveler support for forwarding DCE credentials to LoadLeveler jobs. You may override the default function provided by LoadLeveler to establish DCE credentials by substituting your own programs.

**Using the alternative program pair: llgetdce and llsetdce:** The program pair, **llgetdce** and **llsetdce**, forwards DCE credentials by copying credential cache files from the submitting machine to the executing machines. While this technique may require less overhead, it has been known to produce credentials on the executing machines which are not fully capable of being forwarded by rsh commands. This is the only pair of programs offered in earlier releases of LoadLeveler.

**Forwarding DCE credentials:** An example of a credentials object is a character string containing the DCE principle name and a password.

*program1* writes the following to standard output:
- The length of the handle to follow
- The handle

If *program1* encounters errors, it writes error messages to standard error.

*program2* receives the following as standard input:

- The length of the handle to follow
- The same handle written by *program1*

*program2* writes the following to standard output:

- The length of the login context to follow
- An exportable DCE login context, which is the idl_byte array produced from the sec_login_export_context DCE API call. For more information, see the DCE Security Services API chapter in the *Distributed Computing Environment for AIX: Application Development Reference*.
- A character string suitable for assigning to the KRB5CCNAME environment variable This string represents the location of the credentials cache established in order for *program2* to export the DCE login context.

If *program2* encounters errors, it writes error messages to standard error. The parent process, the LoadLeveler starter process, writes those messages to the starter log.

For examples of programs that enable DCE security credentials, see the **samples/lldce** subdirectory in the release directory.

## Handling an AFS token

You can write a program, run by the scheduler, to refresh an AFS token when a job is started. To invoke the program, use the **AFS_GETNEWTOKEN** keyword in your configuration file.

Before running the program, LoadLeveler sets up standard input and standard output as pipes between the program and LoadLeveler. LoadLeveler also sets up the following environment variables:

**LOADL_STEP_OWNER**
: The owner (UNIX user name) of the job

**LOADL_STEP_COMMAND**
: The name of the command the user's job step invokes.

**LOADL_STEP_CLASS**
: The class this job step will run.

**LOADL_STEP_ID**
: The step identifier, generated by LoadLeveler.

**LOADL_JOB_CPU_LIMIT**
: The number of CPU seconds the job is limited to.

**LOADL_WALL_LIMIT**
: The number of wall clock seconds the job is limited to.

LoadLeveler writes the following current AFS credentials, in order, over the standard input pipe:

- The **ktc_principal** structure indicating the service.
- The **ktc_principal** structure indicating the client.
- The **ktc_token** structure containing the credentials.

The ktc_principal structure is defined in the AFS header file **afs_rxkad.h**. The ktc_token structure is defined in the AFS header file **afs_auth.h**.

LoadLeveler expects to read these same structures in the same order from the standard output pipe, except these should be refreshed credentials produced by the installation exit.

The installation exit can modify the passed credentials (to extend their lifetime) and pass them back, or it can obtain new credentials. LoadLeveler takes whatever is returned and uses it to authenticate the user prior to starting the user's job.

## Writing prolog and epilog programs

An administrator can write *prolog* and *epilog* installation exits that can run before and after a LoadLeveler job runs, respectively.

Prolog and epilog programs fall into two types:

- Those that run as the LoadLeveler user ID.
- Those that run in a user's environment.

Depending on the type of processing you want to perform before or after a job runs, specify one or more of the following configuration keywords, in any combination:

- To run a prolog or epilog program under the LoadLeveler user ID, specify `JOB_PROLOG` or `JOB_EPILOG`, respectively.
- To run a prolog or epilog program under the user's environment, specify `JOB_USER_PROLOG` or `JOB_USER_EPILOG`, respectively.

You do not have to provide a prolog/epilog pair of programs. You may, for example, use only a prolog program that runs under the LoadLeveler user ID.

For details about specific keyword syntax and use in the configuration, see Chapter 6, "Configuration keyword reference," on page 91.

**Note:** If process tracking is enabled and your prolog or epilog program invokes the **mailx** command, set the **sendwait** variable to prevent the background mail process from being killed by process tracking.

A user environment prolog or epilog runs with AFS authentication if installed and enabled. For security reasons, you must code these programs on the machines where the job runs *and* on the machine that schedules the job. If you do not define a value for these keywords, the user environment prolog and epilog settings on the executing machine are ignored.

The user environment prolog can set environment variables for the job by sending information to standard output in the following format:

```
env id = value
```

Where:

| | |
|---|---|
| **id** | Is the name of the environment variable |
| **value** | Is the value (setting) of the environment variable |

**Note:** Each line of output can contain a maximum of 65,534 characters. All lines containing more than 65,534 characters will be ignored.

For example, the user environment prolog sets the environment variable **STAGE_HOST** for the job:

```
#!/bin/sh
echo env STAGE_HOST=shd22
```

**Coding conventions for prolog programs:**
The prolog program is invoked by the starter process. Once the starter process invokes the prolog program, the program obtains information about the job from environment variables.

**Syntax:**

```
prolog_program
```

Where *prolog_program* is the name of the prolog program as defined in the JOB_PROLOG keyword.

No arguments are passed to the program, but several environment variables are set. For more information on these environment variables, see the topic, "Run-time environment variables" in *LoadLeveler: Using and Administering*.

The real and effective user ID of the prolog process is the LoadLeveler user ID. If the prolog program requires root authority, the administrator must write a secure C or Perl program to perform the desired actions. You should *not* use shell scripts with set uid permissions, since these scripts may make your system susceptible to security problems.

**Return code values:**

**0** The job will begin.

If the prolog program is ended with a signal, the job does not begin and a message is written to the starter log.

**Sample prolog programs:**

- **Sample of a prolog program for korn shell:**

```
#!/bin/ksh
#
# Set up environment
set -a
. /etc/environment
. /.profile
export PATH="$PATH:/loctools/lladmin/bin"
export LOG="/tmp/$LOADL_STEP_OWNER.$LOADL_STEP_ID.prolog"
#
# Do set up based upon job step class
#
case $LOADL_STEP_CLASS in
    # A OSL job is about to run, make sure the osl filesystem is
    # mounted. If status is negative then filesystem cannot be
    # mounted and the job step should not run.
    "OSL")
      mount_osl_files >> $LOG
    if [ status = 0 ]
        then EXIT_CODE=1
      else
        EXIT_CODE=0
      fi
      ;;
# A simulation job is about to run, simulation data has to
# be made available to the job. The status from copy script must
# be zero or job step cannot run.
"sim")

      copy_sim_data >> $LOG
if [ status = 0 ]
        then EXIT_CODE=0
      else
        EXIT_CODE=1
      fi
      ;;
# All other job will require free space in /tmp, make sure
# enough space is available.
*)
```

```
      check_tmp >> $LOG
      EXIT_CODE=$?
      ;;
esac
# The job step will run only if EXIT_CODE == 0
exit $EXIT_CODE
```

- **Sample of a prolog program for C shell:**

```
#!/bin/csh
#
# Set up environment
source /u/loadl/.login
#
setenv PATH  "${PATH}:/loctools/lladmin/bin"
setenv LOG "/tmp/${LOADL_STEP_OWNER}.${LOADL_STEP_ID}.prolog"
#
# Do set up based upon job step class
#
switch ($LOADL_STEP_CLASS)
    # A OSL job is about to run, make sure the osl filesystem is
    # mounted. If status is negative then filesystem cannot be
    # mounted and the job step should not run.
    case "OSL":
      mount_osl_files >> $LOG
      if ($status < 0 ) then
        set EXIT_CODE = 1
      else
        set EXIT_CODE = 0
      endif
      breaksw
# A simulation job is about to run, simulation data has to
# be made available to the job. The status from copy script must
# be zero or job step cannot run.
case "sim":
    copy_sim_data >> $LOG
    if ($status == 0 ) then
      set EXIT_CODE = 0
    else
      set EXIT_CODE = 1
    endif
    breaksw
# All other job will require free space in /tmp, make sure
# enough space is available.
default:
    check_tmp >> $LOG
    set EXIT_CODE = $status
    breaksw
endsw

# The job step will run only if EXIT_CODE == 0
exit $EXIT_CODE
```

**Coding conventions for epilog programs:**
The installation-defined epilog program is invoked after a job step has completed. The purpose of the epilog program is to perform any required clean up such as unmounting file systems, removing files, and copying results. The exit status of both the prolog program and the job step is set in environment variables.

**Syntax:**

```
epilog_program
```

Where *epilog_program* is the name of the epilog program as defined in the **JOB_EPILOG** keyword.

No arguments are passed to the program but several environment variables are set. These environment variables are described in the topic, "Run-time environment variables" in *LoadLeveler: Using and Administering*. In addition, the following environment variables are set for the epilog programs:

**LOADL_PROLOG_EXIT_CODE**
The exit code from the prolog program. This environment variable is set only if a prolog program is configured to run.

**LOADL_USER_PROLOG_EXIT_CODE**
The exit code from the user prolog program. This environment variable is set only if a user prolog program is configured to run.

**LOADL_JOB_STEP_EXIT_CODE**
The exit code from the job step.

**Note:** To interpret the exit status of the prolog program and the job step, convert the string to an integer and use the macros found in the **sys/wait.h** file. These macros include:
- WEXITSTATUS: gives you the exit code
- WTERMSIG: gives you the signal that terminated the program
- WIFEXITED: tells you if the program exited
- WIFSIGNALED: tells you if the program was terminated by a signal

The exit codes returned by the WEXITSTATUS macro are the valid codes. However, if you look at the raw numbers in **sys/wait.h**, the exit code may appear to be 256 times the expected return code. The numbers in **sys/wait.h** are the wait3 system calls.

**Sample epilog programs:**

- **Sample of an epilog program for korn shell:**

```
#!/bin/ksh
#
# Set up environment
set -a
. /etc/environment
. /.profile
export PATH="$PATH:/loctools/lladmin/bin"
export LOG="/tmp/$LOADL_STEP_OWNER.$LOADL_STEP_ID.epilog"
#
if [ [ -z $LOADL_PROLOG_EXIT_CODE ] ]
then
echo "Prolog did not run" >> $LOG
else
echo "Prolog exit code = $LOADL_PROLOG_EXIT_CODE" >> $LOG
fi
#
if [ [ -z $LOADL_USER_PROLOG_EXIT_CODE ] ]
  then
   echo "User environment prolog did not run" >> $LOG
  else
   echo "User environment exit code = $LOADL_USER_PROLOG_EXIT_CODE" >> $LOG
fi
#
if [ [ -z $LOADL_JOB_STEP_EXIT_CODE ] ]
  then
   echo "Job step did not run" >> $LOG
  else
   echo "Job step exit code = $LOADL_JOB_STEP_EXIT_CODE" >> $LOG
fi
#
#
# Do clean up based upon job step class
#
case $LOADL_STEP_CLASS in
```

```
  # A OSL job just ran, unmount the filesystem.
  "OSL")
    umount_osl_files >> $LOG
    ;;
  # A simulation job just ran, remove input files.
  # Copy results if simulation was successful (second argument
  # contains exit status from job step).
  "sim")
    rm_sim_data >> $LOG
    if [ $2 = 0 ]
      then copy_sim_results >> $LOG
    fi
    ;;
# Clean up /tmp
*)
  clean_tmp >> $LOG
  ;;
esac
```

- **Sample of an epilog program for C shell:**

```
#!/bin/csh
#
# Set up environment
source /u/loadl/.login
#
setenv PATH  "${PATH}:/loctools/lladmin/bin"
setenv LOG "/tmp/${LOADL_STEP_OWNER}.${LOADL_STEP_ID}.prolog"
#
if ( ${?LOADL_PROLOG_EXIT_CODE} ) then
echo "Prolog exit code = $LOADL_PROLOG_EXIT_CODE" >> $LOG
else
echo "Prolog did not run" >> $LOG
endif
#
if ( ${?LOADL_USER_PROLOG_EXIT_CODE} ) then
    echo "User environment exit code = $LOADL_USER_PROLOG_EXIT_CODE" >> $LOG
  else
    echo "User environment prolog did not run" >> $LOG
endif
#
if ( ${?LOADL_JOB_STEP_EXIT_CODE} ) then
    echo "Job step exit code = $LOADL_JOB_STEP_EXIT_CODE" >> $LOG
  else
    echo "Job step did not run" >> $LOG
endif
#
# Do clean up based upon job step class
#
switch ($LOADL_STEP_CLASS)
  # A OSL job just ran, unmount the filesystem.
  case "OSL":
    umount_osl_files >> $LOG
    breaksw
# A simulation job just ran, remove input files.
# Copy results if simulation was successful (second argument
# contains exit status from job step).
case "sim":
  rm_sim_data >> $LOG
  if ($argv{2} == 0 ) then
    copy_sim_results >> $LOG
  endif
  breaksw
# Clean up /tmp
default:
  clean_tmp >> $LOG
  breaksw
endsw
```

## Using your own mail program

You can write a program to override the LoadLeveler default mail notification method. You can use this program, for example, to display your own messages to users when a job completes, or to automate tasks such as sending error messages to a network manager.

The syntax for the program is the same as it is for standard UNIX mail programs; the command is called with the following arguments:

- **-s** to indicate a subject.
- A pointer to a string containing the subject.
- A pointer to a string containing a list of mail recipients.

The mail message is taken from standard input.

To enable this program to replace the default mail notification method, use the **MAIL** keyword in the configuration file. For details about specific keyword syntax and use in the configuration, see Chapter 6, "Configuration keyword reference," on page 91.

## Determining the frequency to use to run a job

You can write an installation exit that can determine the frequency to use to run a job. The first time a job is submitted, it runs in the nominal frequency. LoadLeveler gathers the hardware counters for the job and generates the energy tag. When the job runs again, LoadLeveler runs the specified installation exit program to determine the frequency to use to run the job. The energy tag name that was generated during the first run is passed to the user program by the **LL_ENERGY_TAG_NAME** environment variable. The user program can query the energy tag information by using the LoadLeveler API if needed.

The external frequency program is invoked by the Startd process before the job runs.

**Syntax:**

*external_program*

where:

*external_program*
: Is the name of the program as defined in the **EXT_ENERGY_POLICY_PROGRAM** keyword. No arguments are passed to the program.

The **LL_ENERGY_TAG_NAME** environment variable is exported to the program and can be used to query the energy data generated for this tag by LoadLeveler. For more information about the environment variable, see the topic, "Run-time environment variables" in *LoadLeveler: Using and Administering*.

**Return code values:**

**>0** The calculated frequency for running the job.

LoadLeveler uses the returned value as the frequency to use for running the job. If the program returns 0 or a negative value, the job will run at the nominal frequency.

**Sample external frequency program:**

The following is a sample C program, which returns the frequency to use to run a job:

```
/*
 * -------------------------------------------------------------------------
 * Determine the frequency of the job by the user program
 * --------------------------------------------------------------------------
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>
#include <assert.h>
#include <errno.h>
#include <stdint.h> // for uint64_t
#include "llrapi.h"

int
main(int argc, char *argv[]) {
    int frequency = -1;

    // query the energy tag info if needed.
        llr_resmgr_handle_t *rm_handle;
        llr_query_handle_t *q_handle = NULL;
        llr_element_t *err_object;
        llr_element_t **object_list;
        llr_element_t *etag = NULL;
        llr_data_list_t policy_list;
        llr_etag_policy policy;
        void* p = (void *)&policy;
        int rc;
        int i;
        llr_query_filter_t etag_q_filter[1];
        int etag_filter_cnt = 1;
        char **etag_name = (char**)malloc(2*sizeof(char*));
        bzero(etag_name, 2* sizeof(char *));
        assert(argc > 1);
        *etag_name = strdup(argv[1]);
        etag_q_filter[0].filter_type = LLR_QUERY_ETAGNAME; /* take tag name as the query filter */
        etag_q_filter[0].filter_data = etag_name;

        err_object = NULL;
        rc = llr_init_resmgr(LLR_API_VERSION, &rm_handle, &err_object);
        if (rc != LLR_API_OK) {
                fprintf(stderr, "Resource manager API can not be intialized.\n");
                if (err_object) {
                        llr_error(&err_object, LLR_ERROR_PRINT_STDERR);
                }
                exit(-1);
        }

        rc = llr_query_set(rm_handle, &q_handle, LLR_ENERGYTAG_QUERY, etag_filter_cnt, \
        etag_q_filter, &err_object);
        if (rc != LLR_API_OK) {
                fprintf(stderr, "Failed to set up job query.\n");
                if (err_object) {
                        llr_error(&err_object, LLR_ERROR_PRINT_STDERR);
                }
                exit(-1);
        }

        rc =  llr_query_get_data(rm_handle, q_handle, LLR_QUERY_RESOURCE_MANAGER, NULL, \
        &object_list, &err_object);
        if (rc != LLR_API_OK) {
                if (err_object) {
                        llr_error(&err_object, LLR_ERROR_PRINT_STDERR);
                }
                fprintf(stderr, "llr_query_get_data failed.\n");
                exit(-1);
        }

        if (object_list==NULL || object_list[0]==NULL) {
                fprintf(stdout, "No policy record found.\n");
                exit(0);
        }

        memset(&policy, 0, sizeof(policy));
        for (i=0; object_list&&(etag = object_list[i])!=NULL; i++) {
            rc += llr_get_data(rm_handle, etag, LLR_ETagGetPolicyList, &policy_list, \
            &err_object); p = &policy;
            rc += llr_get_data(rm_handle, &policy_list, LLR_ETagGetFirstPolicy, (void*)&p, \
            &err_object); while (p != NULL) {
                fprintf(stdout, "user=%s, etag_name=%s, freq=%d KHZ, est. energy=%f KWH, \
                est. time=%d s, perfd=%f%%, energy saving=%f%%\n", policy.username, \
                policy.energy_tag_name, policy.frequency, policy.predict_power, \
                policy.predict_elapse_time, policy.perf_degrad_pct*100, \
                policy.energy_saving_pct*100); memset(&policy, 0, sizeof(policy))\
                ;p = &policy; rc += llr_get_data(rm_handle, &policy_list, \
                LLR_ETagGetNextPolicy,(void*)&p, &err_object);
           }

           err_object = NULL;
           if (rc != LLR_API_OK) {
                if (err_object) {
                        llr_error(&err_object, LLR_ERROR_PRINT_STDERR);
                }
                fprintf(stderr, "llr_get_data failed.\n");
                exit(-1);
           }
        }
        rc =  llr_query_free_data(rm_handle, &q_handle, &err_object);
        llr_free_resmgr(&rm_handle, &err_object);

// calculate the frequency by user logic
frequency = valid_frequency;

 return frequency;
}
```

To compile the source code, use this command:

```
gcc -o myengprog -lllrapi -I /opt/ibmll/LoadL/resmgr/full/include myengprog.c
```

# Chapter 4. Defining LoadLeveler resources to administer

After installing LoadLeveler, you may customize it by modifying the **administration** file, or if the database configuration option is used, by using the configuration editor to modify the tables for the machines, machine_groups, classes, users, and groups.

For file-based configuration, the administration file optionally lists and defines the machines in the LoadLeveler cluster and the characteristics of classes, users, and groups.

LoadLeveler does not prevent you from having multiple copies of administration files, but you need to be sure to update all the copies whenever you make a change to one. Having only one administration file prevents any confusion.

Table 14 lists the LoadLeveler resources you may define by modifying the administration file.

*Table 14. Roadmap of tasks for modifying the LoadLeveler administration file*

| To learn about: | Read the following: |
|---|---|
| Defining LoadLeveler resources to administer | • "Dynamic adapter discovery" on page 65<br>• "LoadLeveler adapter and node status monitoring" on page 65<br>• "Defining classes" on page 66<br>• "Defining users" on page 70<br>• "Defining groups" on page 71 |
| Correctly specifying administration file keywords | Chapter 6, "Configuration keyword reference," on page 91 |

## Defining machines

Characteristics of a machine may be defined in a machine stanza, a machine group stanza, or a machine sub-stanza of a machine group.

A machine_group stanza groups together machines with similar characteristics. Using machine_groups reduces the need to specify each machine in the cluster in the administration file because each machine_group stanza can specify a **machine_list** that supports machine range expressions similar to the syntax supported by xCAT.

If you do not specify a machine or machine_group stanza for a machine in the cluster, the machines can still communicate with one another and run jobs, but the machine is assigned the default values specified in the default machine_group stanza. If there is no default stanza, the machine is assigned default values set by resource manager.

If you set the **MACHINE_AUTHENTICATE** keyword to true in the configuration file, then for each machine that LoadLeveler includes in the cluster you must either create a machine stanza or include the machine in a machine_group using either a machine substanza or the **machine_list** keyword.

In the LoadLeveler configuration, machine names are stored in lower case, so all references to the machine names must be in lower case.

## Planning considerations for defining machines

There are several matters to consider before customizing the administration file. Before customizing the administration file, consider the following:

- **Node availability**

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

  To run jobs on any machine in the LoadLeveler cluster, a user needs the same uid (the user ID number for a user) and gid (the group ID number for a group) on every machine in the cluster.

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

  If you do not have a user ID on one machine, your jobs will not run on that machine.

## Machine_group stanza format and keyword summary

Machine_group stanzas take the following format. The default values for keywords appear in bold:

```
label: {
   type = machine_group
```

The following keywords are used by the resource manager and are made available to the scheduling application:

```
   adapter_list = adapter_name...
```

The following keywords are not used by the resource manager, but they are made available to the scheduling application:

```
   class = class_name(count) class_name(count) ... class_name(count)
   feature = feature_name...
   mode = batch | interactive | general
   pool_list = pool_numbers
   resources = name(count) name(count) ... name(count)
```

The following keywords are used by the resource manager during start up:

```
   machine_list = range_expression
   name_server = list
   power_management_policy = start_time;duration | off
   prestarted_starters = number
   region = region_name
   schedd_fenced = true | false
   schedd_host = true | false
   schedd_runs_here = true | false
   startd_runs_here = true | false
```

The following keywords are used by the resource manager accounting function:

```
   cpu_speed_scale = true | false
   speed = number
}
```

**Notes:**

1. Each of these machine stanza keywords apply to all machines within the group. For example, specifying **startd_runs_here = true** in a machine_group stanza means that every machine in that machine group has **startd_runs_here** set to **true**.
2. The **machine_list** keyword is optional when there are machine substanzas, otherwise **machine_list** must be present.
3. A machine can belong to only one machine_group. Machines can also be defined in separate machine stanzas outside of any machine_group.

## Machine substanza format and keyword summary

The following example shows a substanza of type machine. The default values for keywords appear in bold:

```
label: {
  type = machine_group
  ...
  label: {
  type = machine
  feature = feature_name...
  schedd_fenced = true | false
  schedd_host = true | false
  schedd_runs_here = true | false
  startd_runs_here = true | false
 }
}
```

**Notes:**

1. Not all keywords that are permitted in machine and machine_group stanzas may be specified in a machine substanza. If it is necessary to override a machine_group setting for a machine, then that machine should be defined in its own machine stanza outside of any machine_group.
2. A machine may appear as a machine substanza and in the machine_list within the same machine_group stanza.

## Machine stanza format and keyword summary

Machine stanzas take the following format. The default values for keywords appear in bold:

```
label: type = machine
```

The following keyword is used by the resource manager and is made available to the scheduling application:

```
adapter_list = adapter_name...
```

The following keywords are not used by the resource manager, but they are made available to the scheduling application:

```
  class = class_name(count) class_name(count) ... class_name(count)
  feature = feature_name...
  machine_mode = batch | interactive | general
  pool_list = pool_numbers
  resources = name(count) name(count) ... name(count)
```

The following keywords are used by the resource manager during start up:

**name_server** = *list*
**power_management_policy** = *start_time;duration* | **off**
**prestarted_starters** = *number*
**region** = *region_name*
**schedd_fenced** = true | **false**
**schedd_host** = true | **false**
**schedd_runs_here** = **true** | false
**startd_runs_here** = **true** | false

The following keywords are used by the resource manager accounting function:

**cpu_speed_scale** = true | **false**
**speed** = *number*

## Default values for machine_group and machine stanzas

A special machine_group stanza with the name of default can be specified to define the values for keywords for all other machine_group and machine stanzas. Any keyword not explicitly defined in the default machine_group stanza is assigned a default value by LoadLeveler. The rules governing these default values include:

- A machine inherits the attributes from its machine_group. If a machine does not belong to a machine_group, it inherits the attributes from the default machine_group.
- A default machine_group stanza must come before any other machine_group stanzas.
- The **machine_list** keyword and machine substanzas are not allowed in a default machine_group stanza.
- If there are any machine_group stanzas present in the administration file at all, then a default machine stanza cannot be specified (it will be considered an error). Both machine and machine_group stanzas inherit the values specified in the default machine_group stanza.
- A nondefault machine_group stanza serves as the default stanza of all the machines its **machine_list** covers.
- The explicitly defined machine substanza will also use the machine_group stanza's keywords as default, even if it is not included in the range the **machine_list** represents.

## Example of machine_group and machine stanzas

This machine stanza example may apply to your situation:

```
default: {
 type = machine_group
 machine_mode = general
 pool_list = 1 7
 startd_runs_here = true
 schedd_runs_here = false
}

MG1: {
 type = machine_group
 schedd_host = true
 resources = ConsumableCpus(all)
 machine_list = x330n01-x330n99,-x330n10,-x330n20
 x330n50: {
   type = machine
   schedd_runs_here = true
   startd_runs_here = false
```

```
  }
}

MG2: {
 Type = machine_group
 Pool_list = 1
 resources = ConsumableCpus(4)
 machine_list = x330n10,x330n20
}
```

In this example, x330n50 is the only machine in the cluster where a Schedd daemon will be started, and no Startd daemon will run there. X330n10 and x330n20 are defined as belonging only to pool 1 and having only 4 ConsumableCpus available, while all other machines in the cluster belong to both pool 1 and pool 7 and all machines in MG1 have all of their CPUs available as ConsumableCpus.

## Dynamic adapter discovery

Adapters are dynamically discovered by the LoadLeveler startd daemon by querying the system configuration and the Protocol Network Services Daemon (PNSD). Startd sends the adapter configuration to the resource manager and region manager daemons on start up and reconfiguration. User space jobs will be supported using InfiniBand or the Host Fabric Interface (HFI) adapters, if PNSD is installed on the nodes. If PNSD is not installed, all adapters will be treated as Ethernet adapters.

The Startd daemon polls the system configuration and PNSD every (POLLS_PER_UPDATE * POLLING_FREQUENCY) seconds to pick up any new information. Any changes discovered are sent to the negotiator, resource manager, and region manager.

If only certain adapter interfaces are to be used for a machine, then the **adapter_list** keyword under the machine or **machine_group** stanza can be used to list the adapter interface names in the order that will be used for scheduling jobs. If this keyword is not specified in the machine or machine group configuration, then all discovered adapters will be used.

**Notes for InfiniBand adapters:**

1. LoadLeveler distributes the switch adapter windows of the InfiniBand adapter equally among its ports and the allocation is not adjusted should all of the resources on one port be consumed.
2. If one InfiniBand port is in use exclusively, no other ports on the InfiniBand adapter can be used for any other job.
3. Because InfiniBand adapters do not support rCxt blocks, jobs that request InfiniBand adapters and rCxt blocks with the **rcxtblocks** keyword on the network statement will remain in the idle state.

## LoadLeveler adapter and node status monitoring

Machine and adapter configuration and status changes will be detected by the region manager and the startd daemons. If the region manager daemon is not configured, the adapter and node status will only come from the configuration information available to the startd daemon and will not reflect the actual connectivity of the adapter or node.

**Note:** The region manager node must have similar connectivity to the network as the executing machine it manages, so that all of its configured network interfaces are able to connect to all of the executing machine's network interfaces.

The startd daemon generates its adapter configuration and local status and sends the information to the region manager, central manager, and resource manager. Adapter evaluations are done by the startd during the **polls_per_update** * **polling_frequency** intervals. The polling keywords will trigger how often the adapter information is updated. Node and adapter information and status will be sent to the daemons if changes were detected by the startd. The startd also sends heartbeats to the region manager over each of its configured network interfaces if a region is configured.

The region manager maintains the machine and adapter status for all the nodes in its managed region. The region manager will mark the adapter down after a period of **adapter_heartbeat_interval** * **adapter_heartbeat_retries** if no heartbeat is received. When an adapter comes up, the region manager will receive the heartbeat from the startd and will immediately mark it as up. The region manager will send heartbeat status changes to the central manager and the resource manager.

## Defining classes

The information in a class stanza defines characteristics for that class.

These characteristics can include the quantities of consumable resources that may be used by a class per machine or cluster.

Within a class stanza, you can have optional user substanzas that define policies that apply to a user's job steps that need to use this class. For more information about user substanzas, see . For information about user stanzas, see “Defining users” on page 70.

### Using limit keywords

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

Limits include both a **hard limit** and a **soft limit**. When a hard limit is exceeded, the job is usually terminated. When a soft limit is exceeded, the job is usually given a chance to perform some recovery actions. Limits are enforced either per process or per job step, depending on the type of limit. For parallel jobs steps, which consist of multiple tasks running on multiple machines, limits are enforced on a per task basis.

The class stanza includes the **limit** keywords shown in Table 15, which allow you to control the amount of resources used by a job step or a job process.

*Table 15. Types of limit keywords*

| Limit | How the limit is enforced |
|---|---|
| **as_limit** | Per process |
| **ckpt_time_limit** | Per job step |
| **core_limit** | Per process |
| **cpu_limit** | Per process |

*Table 15. Types of limit keywords (continued)*

| Limit | How the limit is enforced |
|---|---|
| **data_limit** | Per process |
| **default_wall_clock_limit** | Per job step |
| **file_limit** | Per process |
| **job_cpu_limit** | Per job step |
| **locks_limit** | Per process |
| **memlock_limit** | Per process |
| **nofile_limit** | Per process |
| **nproc_limit** | Per user |
| **rss_limit** | Per process |
| **stack_limit** | Per process |
| **wall_clock_limit** | Per job step |

For example, a common limit is the **cpu_limit**, which limits the amount of CPU time a single process can use. If you set **cpu_limit** to five hours and you have a job step that forks five processes, each process can use up to five hours of CPU time, for a total of 25 CPU hours. Another limit that controls the amount of CPU used is **job_cpu_limit**. For a serial job step, if you impose a **job_cpu_limit** of five hours, the entire job step (made up of all five processes) cannot consume more than five CPU hours. For information on using this keyword with parallel jobs, see *LoadLeveler: Using and Administering*.

You can specify limits in either the class stanza of the administration file or in the job command file. The lower of these two limits will be used to run the job even if the system limit for the user is lower. For more information, see:

- "Enforcing limits"
- Chapter 7, "Administration keyword reference," on page 129

## Enforcing limits

LoadLeveler depends on the underlying operating system to enforce process limits. Users should verify that a process limit such as **rss_limit** is enforced by the operating system, otherwise setting it in LoadLeveler will have no effect.

**Exceeding job step limits:** When a hard limit is exceeded LoadLeveler sends a *non-trappable* signal (except in the case of a parallel job) to the process group that LoadLeveler created for the job step. When a soft limit is exceeded, LoadLeveler sends a *trappable* signal to the process group. Any job application that intends to trap a signal sent by LoadLeveler must ensure that all processes in the process group set up the appropriate signal handler.

All processes in the job step normally receive the signal. The exception to this rule is when a child process creates its own process group. That action isolates the child's process, and its children, from any signals that LoadLeveler sends. Any child process creating its own process group is still known to process tracking. So, if process tracking is enabled, all the child processes are terminated when the main process terminates.

Table 16 on page 68 summarizes the actions that the **LoadL_starter** daemon takes when a job step limit is exceeded.

*Table 16. Enforcing job step limits*

| Type of Job | When a Soft Limit is Exceeded | When a Hard Limit is Exceeded |
|---|---|---|
| Serial | SIGXCPU or SIGKILL issued | SIGKILL issued |
| Parallel | SIGXCPU issued to both the user program and to the parallel daemon | SIGTERM issued |

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

**Enforcing per process limits:** For per process limits, what happens when your job reaches and exceeds either the soft limit or the hard limit depends on the operating system you are using. When a job forks a process that exceeds a per process limit, such as the CPU limit, the operating system (not LoadLeveler) terminates the process by issuing a SIGXCPU. As a result, you will not see an entry in the LoadLeveler logs indicating that the process exceeded the limit. The job will complete with a 0 return code. LoadLeveler can only report the status of any processes it has started.

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

**How LoadLeveler uses hard limits:** Consider these details on how LoadLeveler uses hard limits. See Table 17 for more information on specifying limits.

*Table 17. Setting limits*

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

## Class stanza format and keyword summary

Class stanzas are optional. Class stanzas take the following format. Default values for keywords appear in bold.

```
label: type = class
```

The following keywords are used by the resource manager when processing a new job request:

```
default_network.protocol = type[, usage[, mode[,comm_level[, instances=<number \
|max> [, rcxtblocks=number]]]]]
default_resources = name(count) name(count)...name(count)
default_node_resources = name(count) name(count)...name(count)
env_copy = all | master
exclude_groups = list
exclude_users = list
max_node = number
max_protocol_instances = number
max_node_resources = name(count) name(count)...name(count)
total_tasks = number
```

The following keywords are used by the resource manager when running a job:

```
as_limit= hardlimit,softlimit
ckpt_time_limit = hardlimit,softlimit
collective_groups = number
core_limit = hardlimit,softlimit
cpu_limit = hardlimit,softlimit
data_limit = hardlimit,softlimit
default_wall_clock_limit =hardlimit,softlimit
endpoints = number
file_limit = hardlimit,softlimit
imm_send_buffers = number
include_groups = list
include_users = list
job_cpu_limit = hardlimit,softlimit
locks_limit = hardlimit,softlimit
memlock_limit = hardlimit,softlimit
nice = value
nofile_limit = hardlimit,softlimit
nproc_limit = hardlimit,softlimit
restart = yes | no
rss_limit = hardlimit,softlimit
smt = yes | no | as_is
stack_limit = hardlimit,softlimit
wall_clock_limit = hardlimit,softlimit
```

## Examples: Class stanzas

Any of these class stanza examples may apply to your situation.

- **Example 1: Creating a class that excludes certain users**

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

- **Example 2: Creating a class for small-size jobs**

```
small:  type=class                              # class for small jobs
cpu_limit=00:02:00                              # 2 minute limit
data_limit=30mb                                 # max 30 MB data segment
default_resources=ConsumableVirtualMemory(10mb) # resources consumed by each
ConsumableCpus(1) resA(3) floatinglicenseX(1)   # task of a small job step if
                                                # resources are not explicitly
                                                # specified in the job command file
ckpt_time_limit=3:00,2:00                       # 3 minute hardlimit,
                                                # 2 minute softlimit
core_limit=10mb                                 # max 10 MB core file
file_limit=50mb                                 # max file size 50 MB
```

```
stack_limit=10mb                              # max stack size 10 MB
rss_limit=35mb                                # max resident set size 35 MB
include_users = bob sally                     # authorized users
```

- **Example 3: Creating a class for medium-size jobs**

```
medium: type=class            # class for medium jobs
cpu_limit=00:10:00            # 10 minute run time limit
data_limit=80mb,60mb          # max 80 MB data segment
                              # soft limit 60 MB data segment
ckpt_time_limit=5:00,4:30     # 5 minute hardlimit,
                              # 4 minute 30 second softlimit to checkpoint
core_limit=30mb               # max 30 MB core file
file_limit=80mb               # max file size 80 MB
stack_limit=30mb              # max stack size 30 MB
rss_limit=100mb               # max resident set size 100 MB
job_cpu_limit=1800,1200       # hard limit is 30 minutes,
                              # soft limit is 20 minutes
```

- **Example 4: Creating a class for large-size jobs**

```
large:  type=class             # class for large jobs
cpu_limit=00:10:00             # 10 minute run time limit
data_limit=120mb               # max 120 MB data segment
default_resources=ConsumableVirtualMemory(40mb)         # resources consumed
ConsumableCpus(2) resA(8) floatinglicenseX(1) resB(1)   # by each task of
                               # a large job step if resources are not
                               # explicitly specified in the job command file
ckpt_time_limit=7:00,5:00      # 7 minute hardlimit,
                               # 5 minute softlimit to checkpoint
core_limit=30mb                # max 30 MB core file
file_limit=120mb               # max file size 120 MB
stack_limit=unlimited          # unlimited stack size
rss_limit=150mb                # max resident set size 150 MB
job_cpu_limit = 3600,2700      # hard limit 60 minutes
                               # soft limit 45 minutes
wall_clock_limit=12:00:00,11:59:55 # hard limit is 12 hours
```

- **Example 5: Creating a class for master node machines**

```
sp-6hr-sp:  type=class         # class for master node machines
ckpt_time_limit=25:00,20:00    # 25 minute hardlimit,
                               # 20 minute softlimit to checkpoint
cpu_limit = 06:00:00           # 6 hour limit
job_cpu_limit = 06:00:00       # hard limit is 6 hours
core_limit = 1mb               # max 1MB core file
```

## Defining users

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

### User stanza format and keyword summary

User stanzas take the following format:

```
label: type = user
```

The following keywords are used by the resource manager when processing a new job request:

```
account = list
default_class = list
default_group = group name
default_interactive_class = class name
env_copy = all | master
max_node = number
total_tasks = number
```

For more information about the keywords listed in the user stanza format, see Chapter 7, “Administration keyword reference,” on page 129.

## Examples: User stanzas

Any of the following user stanzas may apply to your situation.

- **Example 1**

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

  ```
  # Define user stanzas
  fred:  type = user
  default_class = class_a
  ```
- **Example 2**

  This example explains how a default interactive class for a parallel job is set by presenting a series of user stanzas and class stanzas. This example assumes that users do not specify the LOADL_INTERACTIVE_CLASS environment variable.

  ```
  default: type =user
           default_interactive_class = red
           default_class = blue

  carol:   type = user
           default_class = single double
           default_interactive_class = ijobs

  steve:   type = user
           default_class = single double

  ijobs:   type = class
           wall_clock_limit = 08:00:00

  red:     type = class
           wall_clock_limit = 30:00
  ```

  If the user Carol submits an interactive job, the job is assigned to the default interactive class called **ijobs**. The job is assigned a wall clock limit of 8 hours. If the user Steve submits an interactive job, the job is assigned to the **red** class from the default user stanza. The job is assigned a wall clock limit of 30 minutes.
- **Example 3**

  In this example, if Jane does not specify a job class in her job command file, the default job class **small_jobs** is used. This user stanza does not specify the maximum number of nodes that Jane can request for a job, so this value defaults to the value defined in the default stanza. Also, suppose Jane is a member of the primary UNIX group “staff.” Jobs submitted by Jane will use the default LoadLeveler group “staff.” Lastly, Jane can use three different account numbers.

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

# Defining groups

LoadLeveler groups are another way of granting control to the system administrator.

Although a LoadLeveler group is independent from a UNIX group, you can configure a LoadLeveler group to have the same users as a UNIX group by using the **include_users** keyword. If you do not specify a value for the group keyword in the job command file, the default group for the user is used. If a default group is not defined for the user, LoadLeveler uses the group, **No_Group**.

### Group stanza format and keyword summary

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

```
  label: type = group
```

The following keywords are used by the resource manager when processing a new job request:

```
account = list
default_class = list
default_group = group name
default_interactive_class = class name
env_copy = all | master
max_node = number
total_tasks = number
```

For more information about the keywords listed in the group stanza format, see Chapter 7, "Administration keyword reference," on page 129.

### Example: Group stanzas

The following group stanza may apply to your situation.

In this example, the group name is **department_a**. There are three members in this group.

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

## Defining regions

The region stanza defines the managed region on the cluster for the region managers.

The region stanza is part of the **LoadL_admin** file and contains the **region_mgr_list** keyword. The **region_mgr_list** contains a list of machine names. The first entry is the primary region manager while the remainder are alternate region managers. The primary and alternate region managers will use the same failover scheme as the central manager.

Every machine listed in the **region_mgr_list** must be unique and must belong to the region.

A machine can belong to only one region. Every machine in a machine group belongs to the same region. A region cannot be specified in a machine substanza.

If required, a default region stanza must be specified explicitly. The default region is assigned to machines and machine groups that do not specify a region.

If there are any regions defined, then every machine or machine group must be assigned to a valid region either explicitly or by default.

The region manager daemon will only start if there is a region stanza defined. LoadLeveler will not start if the regions for the cluster are not defined correctly.

**Note:** The region manager node must have similar connectivity to the network as the executing machine it manages, so that all of its configured network interfaces are able to connect to all of the executing machine's network interfaces.

## Region stanza format and keyword summary

Region stanzas take the following format.

*label:* **type = region**
**region_mgr_list** = *list*

## Examples: Region stanzas

Any of these region stanza examples may apply to your situation.

- **Example 1**

  In this example, MachineGroupB is in the RegionB region. The primary machine to run the region manager daemon is c197blade4b06. c197blade4b11 is the backup or alternate node on which the region manager will come up if the primary node goes down.

  ```
  default: type = region
          region_mgr_list = c197blade4b22 c197blade4b24

  RegionA: type = region
          region_mgr_list = c197blade4b02 c197blade4b04

  RegionB: type = region
          region_mgr_list = c197blade4b06 c197blade4b11

  MachineGroupA: type = machine_group
                 machine_list = c197blade4b[01-05]
                 region = RegionA
                 ...

  MachineGroupB: type = machine_group
                 machine_list = c197blade4b[06-18]
                 region = RegionB
                 ...

  MachineGroupC: type = machine_group
                 machine_list = c197blade4b[20-24]
                 region = default
                 ...
  ```

- **Example 2**

  For an individual machine, the region keyword will specify the region to which the machine belongs. The region stanza will then show the primary region manager and the list of alternate region managers for that region.

  ```
  RegionA: type = region
          region_mgr_list = c197blade4b02 c197blade4b04

  c197blade4b02: type = machine
                 region = RegionA
                 ...

  c197blade4b03: type = machine
                 region = RegionA
                 ...

  c197blade4b04: type = machine
                 region = RegionA
                 ...
  ```

- **Example 3**

  This example shows that a default region stanza can be created so that the machine group and machine stanza can default to this region stanza if a region keyword is not specified in the machine group or machine stanza.

  ```
  default: type = region
                  region_mgr_list = c197blade4b22 c197blade4b24

  # Machine group defaults to default region stanza
  MachineGroupA: type = machine
                 machine_list = c197blade4b[10-26]
                 ...

  # Machine defaults to default region stanza
  c197blade4b05: type = machine
  ```

# Chapter 5. Performing additional administrator tasks

There are additional ways to modify the LoadLeveler environment that either require an administrator to customize the configuration database or both the configuration and administration files, or require the use of the LoadLeveler commands or APIs.

Table 18 lists additional ways to modify the LoadLeveler environment.

*Table 18. Roadmap of additional administrator tasks*

| To learn about: | Read the following: |
|---|---|
| Setting up the environment for parallel jobs | "Setting up the environment for parallel jobs" |
| Working with the workload balancing component | "Steps for integrating LoadLeveler with the Workload Manager" on page 76 |
| Enabling LoadLeveler's checkpoint/restart function | "LoadLeveler support for checkpointing jobs" on page 78 |
| Enabling LoadLeveler's checkpoint/restart function for jobs using MetaCluster HPC | "Submitting a MetaCluster HPC checkpoint job to LoadLeveler" on page 81 |
| Moving job records from a down Schedd to another Schedd within the local cluster | • "Procedure for recovering a job spool" on page 86<br>• **llrmovespool** (see "llrmovespool - Move job records" on page 177) |
| Enabling energy aware job support | • "Working with energy aware jobs" on page 87<br>• "S3 state support" on page 88 |
| Correctly specifying configuration and administration file keywords | • Chapter 6, "Configuration keyword reference," on page 91<br>• Chapter 7, "Administration keyword reference," on page 129 |
| Managing LoadLeveler operations | |
| • Querying status | • **llrq** (see "llrq - Query job information" on page 179)<br>• **llrstatus** (see "llrstatus - Query machine information" on page 200) |
| • Changing the state of submitted jobs | • **llcancel**<br>• **llhold**<br><br>See *LoadLeveler: Command and API Reference* for command descriptions. |

## Setting up the environment for parallel jobs

Additional administration tasks apply to parallel jobs.

This topic describes the following administration tasks that apply to parallel jobs:
- Scheduling support
- Reducing job launch overhead
- Submitting interactive POE jobs

- Setting up a class
- Setting up a parallel master node

### Steps for reducing job launch overhead for parallel jobs

Administrators may define a number of LoadLeveler starter processes to be ready and waiting to handle job requests. Having this pool of ready processes reduces the amount of time LoadLeveler needs to prepare jobs to run. You also may control how environment variables are copied for a job. Reducing the number of environment variables that LoadLeveler has to copy reduces the amount of time LoadLeveler needs to prepare jobs to run.

**Before you begin:** You need to know:

- How many jobs might be starting at the same time. This estimate determines how many starter processes to have LoadLeveler start in advance, to be ready and waiting for job requests.
- The type of parallel jobs that typically are used. If IBM Parallel Environment (PE) is used for parallel jobs, PE copies the user's environment to all executing nodes. In this case, you may configure LoadLeveler to avoid redundantly copying the same environment variables.

Perform the following steps to configure LoadLeveler to reduce job launch overhead for parallel jobs.

1. In the local or global configuration file, specify the number of starter processes for LoadLeveler to automatically start before job requests are submitted. Use the PRESTARTED_STARTERS keyword to set this value.

   **Tip:** The default value of 1 should be sufficient for most installations.
2. If typical parallel jobs use a facility such as Parallel Environment, which copies user environment variables to all executing nodes, set the **env_copy** keyword in the class, user, or group stanzas to specify that LoadLeveler only copy user environment variables to the master node by default.

   **Rules:**
   - Users also may set this keyword in the job command file. If the **env_copy** keyword is set in the job command file, that setting overrides any setting in the administration file. For more information, see the topic "Step for controlling whether LoadLeveler copies environment variables to all executing nodes" in *LoadLeveler: Using and Administering*.
   - If the **env_copy** keyword is set in more than one stanza in the administration file, LoadLeveler determines the setting to use by examining all values set in the applicable stanzas. See the table in the **env_copy** keyword in Chapter 7, “Administration keyword reference,” on page 129 to determine what value LoadLeveler will use.
3. Notify LoadLeveler daemons by issuing the **llrctl** command with either the **reconfig** or **recycle** keyword. Otherwise, LoadLeveler will not process the modifications you made to the configuration and administration files.

When you are done with this procedure, you can use the POE stderr and LoadLeveler logs to trace actions during job launch.

## Steps for integrating LoadLeveler with the Workload Manager

Another administrative setup task you must consider is whether you want to enforce resource usage of **ConsumableCpus**, **ConsumableMemory**, **ConsumableVirtualMemory**, and **ConsumableLargePageMemory**.

If you want to control these resources, Workload Manager (WLM) can be integrated with LoadLeveler to balance workloads at the machine level. When you are using WLM, workload balancing is done by assigning relative priorities to job processes. These job priorities prevent one job from monopolizing system resources when that resource is under contention.

To integrate LoadLeveler and WLM, perform the following steps:

1. As required for your use, define the applicable options for **ConsumableCpus**, **ConsumableMemory**, **ConsumableVirtualMemory**, or **ConsumableLargePageMemory** in the **ENFORCE_RESOURCE_USAGE** global configuration keyword. This enables enforcement of these consumable resources by WLM.
2. Define **hard**, **soft** or **shares** in the **ENFORCE_RESOURCE_POLICY** configuration keyword. This defines what policy is used by LoadLeveler for CPUs and real memory when setting WLM class resource entitlements.
3. (Optional) Set the **ENFORCE_RESOURCE_MEMORY** configuration keyword to **true**. This setting allows WLM to limit the real memory usage of a WLM class as precisely as possible. When a class exceeds its limit, all processes in the class are killed.

   **Rule: ConsumableMemory** must be defined in the **ENFORCE_RESOURCE_USAGE** keyword in the global configuration file, or LoadLeveler does not consider the **ENFORCE_RESOURCE_MEMORY** keyword to be valid.

   **Tips:**

   - When set to true, the **ENFORCE_RESOURCE_MEMORY** keyword overrides the policy set through the **ENFORCE_RESOURCE_POLICY** keyword for **ConsumableMemory** only. The **ENFORCE_RESOURCE_POLICY** keyword value still applies for **ConsumableCpus**.
   - **ENFORCE_RESOURCE_MEMORY** may be set in either the global or the local configuration file. In the global configuration file, this keyword sets the default value for all the machines in the LoadLeveler cluster. If the keyword also is defined in a local file, the local setting overrides the global setting.
4. Using the **resources** keyword in a machine stanza in the administration file, define the CPU, real memory, virtual memory, and large page machine resources available for user jobs.
   - The **ConsumableCpus** reserved word accepts a count value of "all." This indicates that the initial resource count will be obtained from the Startd machine update value for CPUs.
   - If no resources are defined for a machine, then no enforcement will be done on that machine.
   - If the count specified by the administrator is greater than what the Startd update indicates, the initial count value will be reduced to match what the Startd reports.
   - For CPUs and real memory, if the count specified by the administrator is less than what the Startd update indicates, the WLM resource shares assigned to a job will be adjusted to represent that difference. In addition, a WLM softlimit will be defined for each WLM class. For example, if the administrator defines 8 CPUs on a 16 CPU machine, then a job requesting 4 CPUs will get a share of 4 and a softlimit of 50%.
   - Use caution when determining the amount of real memory available for user jobs. A certain percentage of a machine's real memory will be dedicated to the Default and System WLM classes and will not be included in the calculation of real memory available for users jobs.

- On AIX, start LoadLeveler with the **ENFORCE_RESOURCE_USAGE** keyword enabled and issue **wlmstat -v -m**. Look at the npg column to determine how much memory is being used by these classes.
- On Linux, start LoadLeveler with the **ENFORCE_RESOURCE_USAGE** keyword enabled and issue **cat /cgroup/memory/memory.usage_in_bytes** to determine how much memory is being used by these classes.

- **ConsumableVirtualMemory** and **ConsumableLargePageMemory** are hard max limit values.
  - WLM considers the **ConsumableVirtualMemory** value to be real memory plus large page plus swap space.
  - The **ConsumableLargePageMemory** value should be a value equal to the multiple of the pagesize. For example, 16MB (page size) * 4 pages = 64MB.

## LoadLeveler support for checkpointing jobs

Checkpointing is a method of periodically saving the state of a job step so that if the step does not complete it can be restarted from the saved state.

When checkpointing is enabled, checkpoints can be initiated from within the application at major milestones, or by the user, administrator or LoadLeveler external to the application.

Once a job step has been successfully checkpointed, if that step terminates before completion, the checkpoint file can be used to resume the job step from its saved state rather than from the beginning. When a job step terminates and is removed from the LoadLeveler job queue, it can be restarted from the checkpoint file by submitting a new job and setting the **restart_from_ckpt= yes** job command file keyword. When a job is terminated and remains on the LoadLeveler job queue, the job step will automatically be restarted from the latest valid checkpoint file when the job is dispatched by the scheduling application.

To find out more about checkpointing jobs, use the information in Table 19.

*Table 19. Roadmap of tasks for checkpointing jobs*

| **Subtask** | **Associated instructions (see . . . )** |
|---|---|
| Preparing the LoadLeveler environment for checkpointing and restarting jobs | • “Checkpoint keyword summary”<br>• “Planning considerations for checkpointing jobs” on page 79 |
| Checkpointing and restarting jobs | • See the topic "Checkpointing a job" in *LoadLeveler: Using and Administering*<br>• “Removing old checkpoint files” on page 84 |
| Correctly specifying configuration and administration file keywords | • Chapter 6, “Configuration keyword reference,” on page 91<br>• Chapter 7, “Administration keyword reference,” on page 129 |

### Checkpoint keyword summary

The following is a summary of keywords associated with the checkpoint and restart function.

- **Configuration file keywords**
  - **CKPT_CLEANUP_INTERVAL**
  - **CKPT_CLEANUP_PROGRAM**
  - **CKPT_EXECUTE_DIR**

  - **MAX_CKPT_INTERVAL**
  - **MIN_CKPT_INTERVAL**

  For more information about these keywords, see Chapter 6, “Configuration keyword reference,” on page 91.
- **Administration file keywords**
  - **ckpt_dir**
  - **ckpt_time_limit**

  For more information about these keywords, see Chapter 7, “Administration keyword reference,” on page 129.
- **Job command file keywords**
  - **checkpoint**
  - **ckpt_dir**
  - **ckpt_execute_dir**
  - **ckpt_subdir**
  - **ckpt_time_limit**
  - **restart_from_ckpt**

  For more information about these keywords, see the topic, "Job command file keyword descriptions" in *LoadLeveler: Using and Administering*.

## Planning considerations for checkpointing jobs

Review the following guidelines before you submit a checkpointing job:

- **Plan for jobs that you will restart on different nodes**

  If you plan to migrate jobs (restart jobs on a different node or set of nodes), you should understand the difference between writing checkpoint files to a local file system versus a global file system (such as AFS or GPFS™). The **ckpt_dir** and **ckpt_subdir** keywords in the job command file allow you to write to either type of file system. If you are using a local file system, before restarting the job from checkpoint, make certain that the checkpoint files are accessible from the machine on which the job will be restarted.

  POE provides the ability to checkpoint and later restart the entire set of programs that make up a parallel application. POE and LoadLeveler use the IBM MetaCluster Checkpoint Restart (MDCR) function, and its associated components, to coordinate the checkpointing and restarting of jobs. On AIX, MDCR invokes the appropriate Application Workload Partition (WPAR) commands on each node of a parallel job and coordinates the checkpoint and restart of those jobs. For more information about AIX WPAR, refer to the IBM AIX Information Center (**http://publib.boulder.ibm.com/infocenter/systems/scope/aix/index.jsp**).

  On AIX, MDCR and its associated components are installed with PE as part of the **installp** process. During the **installp** post-processing phase, PE adds entries to the **/etc/security/privcmds** file for the POE and PMD executables. For more information, see the topic about POE installation effects in *IBM Parallel Environment Runtime Edition: Installation*.

  After POE installation, the system administrator needs to perform a number of tasks to provide users with the ability to checkpoint and restart parallel jobs. These tasks include the following:
  - Authorizing users for checkpointing (AIX only)
  - Defining the directories to be used for checkpointing

  **Notes:**

1. Various limitations apply to checkpointing with PE. For example, checkpointing is only supported on User Space jobs. For a complete list of the restrictions, see *IBM Parallel Environment Runtime Edition: MPI Programming Guide*.
2. For more information about POE installation, see *IBM Parallel Environment Runtime Edition: Installation*.

- **Reserve adequate disk space**

  Checkpoint files require a significant amount of disk space. The checkpoint will fail if the directory where the checkpoint files are written does not have adequate space. Since the old set of checkpoint files are not deleted until the new set of files are successfully created, the checkpoint directory should be large enough to contain two sets of checkpoint files. You can make an accurate size estimate only after you have run your job and noticed the size of the checkpoint file that is created.
- **Plan for staging executables**

  If you want to stage the executable for a job step, use the **ckpt_execute_dir** keyword to define the directory where LoadLeveler will save the executable. This directory cannot be the same as the current location of the executable file, or LoadLeveler will not stage the executable.

  You may define the **ckpt_execute_dir** keyword in either the configuration file or the job command file. To decide where to define the keyword, use the information in Table 20.

*Table 20. Deciding where to define the directory for staging executables*

| If the ckpt_execute_dir keyword is defined in: | Then the following information applies: |
|---|---|
| The configuration file only | • LoadLeveler stages the executable file in a new subdirectory of the specified directory. The name of the subdirectory is the job step ID.<br>• The user is the owner of the subdirectory and has permission 700.<br>• If the user issues the **llckpt** command with the **-k** option, LoadLeveler deletes the staged executable.<br>• LoadLeveler will delete the subdirectory and the staged executable when the job step ends. |
| The job command file only<br>Both the configuration and job command files | • LoadLeveler stages the executable file in the directory specified in the job command file.<br>• The user is the owner of the file and has execute permission for it.<br>• The user is responsible for deleting the staged file after the job step ends. |
| Neither file (the keyword is not defined) | LoadLeveler does not stage the executable file for the job step. |

- **Set your checkpoint file size to the maximum**

  To make sure that your job can write a large checkpoint file, assign your job to a job class that has its file size limit set to the maximum (unlimited). In the administration file, set up a class stanza for checkpointing jobs with the following entry:

  ```
  file_limit = unlimited,unlimited
  ```

  This statement specifies that there is no limit on the maximum size of a file that your program can create.
- **Choose a unique checkpoint file name**

To prevent another job step from writing over your checkpoint file with another checkpoint file, make certain that your checkpoint file name is unique. The **ckpt_dir** and **ckpt_subdir** keywords give you control over the location and name of these files.

### Additional planning considerations for checkpointing MetaCluster HPC jobs on AIX

Before you can checkpoint MetaCluster HPC jobs, you must install LoadLeveler for AIX and MetaCluster HPC.

For MetaCluster HPC installation information, see *Parallel Environment Runtime Edition for AIX: Installation* in the IBM Cluster Information Center (http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp).

### Checkpoint and restart limitations

Use of the checkpoint and restart function has certain limitations. If you are planning to use the checkpoint and restart function, you need to be aware of the types of programs that cannot be checkpointed as well as the restrictions related to the operating system, nodes, tasks, threads, and so on.

For more information about checkpoint and restart limitations, see the *IBM Parallel Environment Runtime Edition MPI Programming Guide* in the IBM Cluster Information Center (http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp) or the IBM Publications Center (http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss).

## Submitting a MetaCluster HPC checkpoint job to LoadLeveler

Several LoadLeveler job command, configuration, and administration keywords are available to support the LoadLeveler/MetaCluster HPC checkpoint and restart operations.

They will be discussed in greater detail in other topics of this information. The purpose of this topic is to simply highlight the fact that a *regular* LoadLeveler job command file can be converted to a *checkpointable* job command file by adding the **checkpoint** and **ckpt_subdir** specifications to the file as shown in “job_1.cmd - A checkpointable job command file.”

### job_1.cmd - A checkpointable job command file

This checkpointable job command file may apply to your situation.

```
#!/bin/ksh
# # A parallel job command file using Metacluster HPC checkpoint/restart
# @ job_type=parallel
# @ step_name = ckpt_1
# @ class = small
# @ checkpoint = yes
# @ ckpt_subdir = /gpfs/cpr/job1
# @ initialdir = /home/llbld
# @ restart = no
# @ cpu_limit = 120
# @ input = /dev/null
# @ output = job1.$(Host).$(Cluster).$(Process).out
# @ error = job1.$(Host).$(Cluster).$(Process).err
# @ notification = error
# @ queue

/usr/bin/poe /gpfs/user/ckpt/ mpi_lapi_3 -t 4 -s 100 -ilevel 2 -pmdlog no
```

In the **job_1.cmd** file, the following has been added to inform LoadLeveler that the job is checkpointable:

```
# @ checkpoint = yes
```

To specify the directory where the checkpoint files will be stored, the following was also added:

```
# @ ckpt_subdir = /gpfs/cpr/job1
```

Note here an important difference in the way LoadLeveler handles checkpoint and noncheckpoint jobs. Because **job_1.cmd** is associated with a checkpoint job, LoadLeveler does not copy this script to the machine selected by LoadLeveler to run the job. This is in contrast to the case of a noncheckpoint job where the script is always copied to the LoadLeveler **execute** directory of the *running* machine. As a result, the **job_1.cmd** file must be accessible to all machines in the LoadLeveler cluster that can run this job. One way of accomplishing this objective is to put it on a shared file system. If this is not possible, you should consider using the **ckpt_execute_dir** keyword. For additional information on this keyword, see “Using the ckpt_execute_dir keyword” on page 84.

Use the **llr_control_job** API to checkpoint a running job. For more information, see the “llr_control_job subroutine” on page 304

## Making periodic checkpoints

LoadLeveler supports periodic checkpointing of user applications if the configuration file **MIN_CKPT_INTERVAL** and **MAX_CKPT_INTERVAL** keywords are defined and the job command file **checkpoint** keyword is set to the value **interval**.

### Example - checkpoint every 30 seconds

Using this example, when the job is run under LoadLeveler, a checkpoint will be made every 30 seconds.

If the LoadLeveler global configuration file contains the following statements:

```
MIN_CKPT_INTERVAL    = 30
MAX_CKPT_INTERVAL    = 30
```

and the job command file contains the statements:

```
# @ checkpoint = interval
# @ ckpt_subdir = /gpfs/cpr/job1
```

then when the job is run under LoadLeveler, a checkpoint will be made every 30 seconds. At each checkpoint, the checkpoint information of the job is saved at one of the following directories:

```
/gpfs/cpr/job1/ckpt_0
/gpfs/cpr/job1/ckpt_1
/gpfs/cpr/job1/ckpt_current -> /gpfs/cpr/job1/ckpt_0
```

The two directories are reused in an endless loop. The **ckpt_current** file is a symbolic link pointing to the directory containing the last successful checkpoint.

### Example - checkpoint interval

This example of the checkpoint interval may apply to your situation.

If the LoadLeveler global configuration file contains the following statements:

```
MIN_CKPT_INTERVAL    = 30
MAX_CKPT_INTERVAL    = 300
```

and the job command file contains the statement:

```
# @ checkpoint = interval
# @ ckpt_subdir = /gpfs/cpr/job1
```

then when the job is run under LoadLeveler, a first checkpoint will be made after 30 seconds. The second checkpoint is made after a time **interval = 2 x MIN_CKPT_INTERVAL**. The time interval between checkpoints will keep on doubling in value until the **MAX_CKPT_INTERVAL** value of 300 seconds is reached. As in "Example - checkpoint every 30 seconds" on page 82, the checkpoint information of the job is saved at either **/gpfs/cpr/job1/ckpt_0** or **ckpt_1** with the **ckpt_current** file pointing to the last successful checkpoint.

## Using the ckpt_dir and ckpt_subdir keywords

The **ckpt_dir** keyword is both a LoadLeveler administration file keyword and a job command file keyword. The **ckpt_subdir** keyword is a job command file keyword only.

The full path name of the directory used to store the LoadLeveler/MetaCluster checkpoint information is a concatenation of the values of **ckpt_dir** and **ckpt_subdir**. In this topic, the use of these keywords is illustrated with a number of examples.

### Example - storing checkpoint information for the job step in the /gpfs/user_1/ckpt_test1 directory

This example of storing checkpoint information for the job step in the /gpfs/user_1/ckpt_test1 directory may apply to your situation.

In this example, the job command file contains the following specifications:

```
# @ ckpt_dir  = /gpfs/MetaC/CKPT
# @ ckpt_subdir = /gpfs/user_1/ckpt_test1
```

Note that the value of **ckpt_subdir** is a string starting with "**/**". Since **ckpt_subdir** specifies a fully qualified path name, the **ckpt_dir** keyword is ignored. LoadLeveler stores checkpoint information for the job step in the **/gpfs/user_1/ckpt_test1** directory.

### Example - storing checkpoint information for the job step in the /gpfs/MetaC/CKPT/ckpt_test1 directory

This example of storing checkpoint information for the job step in the /gpfs/MetaC/CKPT/ckpt_test1 directory may apply to your situation.

In this example, the job command file contains the following specifications:

```
# @ ckpt_dir  = /gpfs/MetaC/CKPT
# @ ckpt_subdir = ckpt_test1
```

LoadLeveler stores checkpoint information for the job step in the **/gpfs/MetaC/CKPT/ckpt_test1** directory, which is a concatenation of the values associated with **ckpt_dir** and **ckpt_subdir**.

### Example - storing checkpoint information for the job in the /gpfs/MetaC/CKPT_small/ckpt_test1 directory

This example of storing checkpoint information for the job in the /gpfs/MetaC/CKPT_small/ckpt_test1 directory may apply to your situation.

In this example, the class stanza of the class **small** in the LoadLeveler administration file contains this specification:

```
ckpt_dir = /gpfs/MetaC/CKPT_small
```

The job command file contains the following specifications:

```
# @ ckpt_subdir = ckpt_test1
# @ class = small
```

LoadLeveler stores checkpoint information for the job in the **/gpfs/MetaC/ CKPT_small/ckpt_test1** directory, which is a concatenation of the value associated with **ckpt_dir** for class **small** and the value of **ckpt_subdir**.

### Example - storing checkpoint information for the job in the /gpfs/MetaC/test99.hostname3.pok.ibm.com.905.1.ckpt directory

In this example, the **ckpt_dir** keyword is not specified in the LoadLeveler administration file or the job command file.

The **ckpt_subdir** keyword is not specified in the job command file. The job command file contains the following specifications:

```
# @ job_name = test99
# @ class = small_job
# @ checkpoint = yes
# @ initialdir = /gpfs/MetaC
```

Assuming that the job is run with the LoadLeveler assigned job step ID of **hostname3.pok.ibm.com.905.1**, LoadLeveler stores checkpoint information for the job in the **/gpfs/MetaC/test99.hostname3.pok.ibm.com.905.1.ckpt** directory. This is because the default value of **ckpt_dir** is the initial working directory and the default value of **ckpt_subfile** is **[***jobname***].***job_step_id***.ckpt**.

**Note:** You must specify a value for **ckpt_subdir** when **restart_from_ckpt=yes**. LoadLeveler cannot generate a default value for **ckpt_subdir** on restart because any default would be based on the job step ID of this new job and **ckpt_subdir** for a restarted job must point to the location of the checkpoint files of some previously run job.

## Removing old checkpoint files

To keep your system free of checkpoint files that are no longer necessary, LoadLeveler provides two keywords to help automate the process of removing these files:

- **CKPT_CLEANUP_PROGRAM**
- **CKPT_CLEANUP_INTERVAL**

Both keywords must contain valid values to automate this process. For information about configuration file keyword syntax and other details, see Chapter 6, “Configuration keyword reference,” on page 91.

## Using the ckpt_execute_dir keyword

The **ckpt_execute_dir** keyword is both a LoadLeveler configuration file keyword and a job command file keyword.

When used as a job command file keyword, it specifies the directory where the job step executable will be saved for a checkpointable job. In this topic, the use of this keyword as a job command file keyword is illustrated with a number of examples.

### Example - using the executable keyword

In this example, both the **executable** keyword and the **ckpt_execute_dir** keywords are specified.

When this **job_2.cmd** is submitted to LoadLeveler, the **/home/llbld/my_bin/ my_application** file is copied at dispatch time to the **/gpfs/user/ckpt_bin** directory and **/gpfs/user/ckpt_bin/my_application** is the full path name of the application that will be run by LoadLeveler.

the **/home/llbld/my_bin/my_application** file is copied at dispatch time to the **/gpfs/user/ckpt_bin** directory and **/gpfs/user/ckpt_bin/my_application** is the full path name of the application that will be run by LoadLeveler.

If a checkpoint and terminate operation is made, the checkpoint information is saved in the **/gpfs/cpr/job2** directory. The **/gpfs/user/ckpt_bin/my_application** file is *not* deleted by LoadLeveler because it may be needed for restart operations. It is your responsibility to manage the files in the **ckpt_execute_dir** directory and to remove any files that are no longer needed.

**job_2.cmd - A parallel checkpoint job using the executable keyword and the ckpt_execute_dir keyword:**

This parallel checkpoint job using the executable keyword and the ckpt_execute_dir keyword may apply to your situation.

```
#!/bin/ksh
# @ job_type=parallel
# @ step_name = ckpt_1
# @ class = small
# @ checkpoint = yes
# @ ckpt_subdir = /gpfs/cpr/job2
# @ executable = /home/llbld/my_bin/my_application
# @ ckpt_execute_dir = /gpfs/user/ckpt_bin
# @ initialdir = /home/llbld
# @ restart = no
# @ cpu_limit = 120
# @ input  = /dev/null
# @ output = job1.$(Host).$(Cluster).$(Process).out
# @ error  = job1.$(Host).$(Cluster).$(Process).err
# @ notification = error
# @ queue
```

## Example - no executable keyword

In this example the executable keyword is not specified in the **job_3.cmd** file and **ckpt_execute_dir** has the value **/gpfs/user/ckpt_bin**. Since the executable keyword is not specified the **job_3.cmd** script itself is the executable. When this job is submitted to LoadLeveler the **job_3.cmd** file is copied to the **/gpfs/user/ckpt_bin** directory at dispatch time and value **/gpfs/user/ckpt_bin /job_3.cmd** is the full path name of the application that will be run by LoadLeveler.

As in "Example - using the executable keyword," when this job terminates the **/gpfs/user/ckpt_bin /job_3.cmd** file is not deleted by LoadLeveler because it may be needed for restart operations. It is your responsibility to manage the files in the **ckpt_execute_dir** directory and to remove any files that are no longer needed.

### job_3.cmd - A parallel checkpoint job using the ckpt_execute_dir keyword, but not the executable keyword

This parallel checkpoint job using the ckpt_execute_dir keyword, but not the executable keyword may apply to your situation.

```
#!/bin/ksh
# @ job_type=parallel
# @ step_name = ckpt_1
# @ class = small
# @ checkpoint = yes
# @ ckpt_subdir = /gpfs/cpr/job2
# @ ckpt_execute_dir = /gpfs/user/ckpt_bin
# @ initialdir = /home/llbld
# @ restart = no
# @ cpu_limit = 120
# @ input = /dev/null
# @ output = job1.$(Host).$(Cluster).$(Process).out
# @ error = job1.$(Host).$(Cluster).$(Process).err
# @ notification = error
# @ queue

/usr/bin/poe /gpfs/user/ckpt/ mpi_lapi_3 -t 4 -s 100 -ilevel 2 -pmdlog no
```

## Procedure for recovering a job spool

The **llrmovespool** command is intended for recovery purposes only.

Jobs being managed by a down Schedd are unable to clean up resources or move to completion. These jobs need their job records transferred to another Schedd. The **llrmovespool** command moves the job records from the spool of one managing Schedd to another managing Schedd in the local cluster. All moved jobs retain their original job identifiers.

It is very important that the Schedd that created the job records to be moved is not running during the move operation. Jobs within the job queue database will be unrecoverable if the job queue is updated during the move by any process other than the **llrmovespool** command.

The **llrmovespool** command operates on a set of job records, these records are updated as the command executes. When a job is successfully moved, the records for that job are deleted. Job records that are not moved because of a recoverable failure, like the original Schedd not being fenced, may have the **llrmovespool** command executed against them again. It is very important that a Schedd never reads the job records from the spool being moved. Jobs will be unrecoverable if more than one Schedd is considered to be the managing Schedd.

The procedure for recovering a job spool is:

1. Move the files located in the spool directory to be transferred to another directory before entering the **llrmovespool** command in order to guarantee that no other Schedd process is updating the job records.
2. Add the statement **schedd_fenced=true** to the machine stanza of the original Schedd node in order to guarantee that the central manager ignores connections from the original managing Schedd, and to prevent conflicts from arising if the original Schedd is restarted after the **llrmovespool** command has been run. See the **schedd_fenced=true** keyword in Chapter 7, “Administration keyword reference,” on page 129 for more information.
3. Reconfigure the central manager node so that it recognizes that the original Schedd is "fenced".
4. Issue the **llrmovespool** command providing the spool directory where the job records are stored. The command displays a message that the transfer has started and reports status for each job as it is processed. For more information, see “llrmovespool - Move job records” on page 177 and the “llr_move_spool subroutine” on page 309.

## Energy aware job support

Using the energy function, a job can run with a lower CPU frequency to save energy.

You can set an acceptable performance degradation (**max_perf_decrease_allowed**) or required energy saving (**energy_saving_req**) for the job in the job command file. LoadLeveler will choose a suitable CPU frequency for the job or reject its submission based on the specified value.

The energy function requires database support.

See the following:
- *LoadLeveler for Linux: Installation Guide* for information about setting up the optional LoadLeveler energy function
- "Working with energy aware jobs" for the steps on how to use the energy keywords in the job command file

### Working with energy aware jobs

The energy policy tag (**energy_policy_tag**) helps LoadLeveler identify the energy data associated with a job. With the energy data, LoadLeveler can decide which frequency should be used to run the job with minimal performance degradation.

The energy policy tag identifies the energy associated with a job. The energy data includes:
- Power consumption and the elapsed time when run in the nominal frequency
- The estimated power consumption
- The elapsed time in other frequencies
- The percentage of performance degradation

When you set the energy policy tag in the job command file, the energy data is generated and stored in the database when the job runs for the first time. When the job is submitted again with the same energy policy tag, the same policy will be used. When you submit a job using the energy functions the first time, be sure to keep the energy tag name unique among the tags you have generated.

To set the energy keywords in the job command file to use the energy function, follow these steps:
1. Provide a unique identifier for the **energy_policy_tag** when a job is submitted the first time. For example:
   ```
   # user.cmd
   #@ energy_policy_tag = my_long_running_job
   ```
   LoadLeveler generates the energy data associated with this energy tag for the job when the job runs. To query the energy data using the energy tag, issue:
   ```
   llrqetag –e my_long_running_job
   ```
2. You can set an acceptable level of performance degradation in the job command file and resubmit the job to run at a lower energy level. Add the following to your job command file and submit it again:
   ```
   # user.cmd
   #@ energy_policy_tag = my_long_running_job
   #@ max_perf_decrease_allowed = 20
   ```

After the job finishes, LoadLeveler will calculate the energy consumption for the job.

3. Issue the following command to get the energy consumption information for your jobs from the accounting data:
   ```
   llsummary -p
   ```

## S3 state support

The S3 state refers to a standby state where RAM remains powered. S3 state support allows the system administrator to switch an idle node to standby state to save energy. LoadLeveler provides a time-based policy to the administrator to decide the start time and duration for the idle node to enter S3 state. The policy is configured for machines or machine_group stanzas using the keyword **power_management_policy**. After the administrator sets up the policy for the compute node, LoadLeveler checks the policy and switches the state of the node automatically at the specified policy start time. The node will switch back to working state (S0) after that time frame has elapsed. An action to take if a machine fails to change to the standby state can be set using the configuration keyword **SUSPEND_CONTROL**.

The CPU does not execute instructions in standby (S3) state, so the node cannot resume from S3 state. The idle node state change is initiated by the resource manager daemon. LoadLeveler calls the xCAT **rpower** command to accomplish the state change.

# Part 3. LoadLeveler interfaces reference

The topics in the LoadLeveler interfaces reference provide the details you need to know to correctly use the IBM LoadLeveler interfaces for specifying keywords in the LoadLeveler control files.

# Chapter 6. Configuration keyword reference

The configuration contains many parameters that you can set or modify to control how LoadLeveler operates.

For a file-based configuration, you can control LoadLeveler's operation either:
- Across the cluster, by modifying the global configuration file, **LoadL_config**, or
- Locally, by modifying the **LoadL_config.local** file on individual machines.

For a database-based configuration, you can control LoadLeveler's operation either:
- Across the cluster, by modifying values in tables for the cluster or for the default machine, or
- By modifying the records for individual machines in tables the database.

Table 21 shows the configuration subtasks:

*Table 21. Configuration subtasks*

| Subtask | Associated information (see . . . ) |
|---|---|
| To find out what administrator tasks you can accomplish by using the configuration | Chapter 3, "Configuring the LoadLeveler resource manager environment," on page 23 |
| To learn how to correctly specify the contents of a configuration | • "Configuration keyword syntax"<br>• "Configuration keyword descriptions" on page 93<br>• "User-defined keywords" on page 121<br>• "LoadLeveler variables" on page 123 |

## Configuration keyword syntax

For files-based configuration, the information in both the **LoadL_config** and the **LoadL_config.local** files is in the form of a statement. These statements are made up of *keywords* and *values*.

There are three types of configuration keywords:
- Keywords, described in "Configuration keyword descriptions" on page 93.
- User-defined variables, described in "User-defined keywords" on page 121.
- LoadLeveler variables, described in "LoadLeveler variables" on page 123.

Configuration file statements take one of the following formats:

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

Statements in the form *keyword=value* are used primarily to customize an environment. Statements in the form *keyword:value* are used by LoadLeveler to specify expressions.

Keywords are *not* case sensitive. This means you can enter them in lower case, upper case, or mixed case.

**Note:** For the *keyword=value* form, if the keyword is of a boolean type and only **true** and **false** are valid input, a value string starting with **t** or **T** is taken as **true**; all other values are taken as **false**.

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

In the configuration file, comments must be on a separate line from keyword statements.

You can use the following types of constants and operators in configuration keywords.

## Numerical and alphabetical constants

These are the numerical and alphabetical constants.

Constants may be represented as:
- Boolean expressions
- Signed integers
- Floating point values
- Strings enclosed in double quotes (" ").

## Mathematical operators

You can use the following C operators.

The operators are listed in order of precedence. All of these operators are evaluated from left to right:
- !
- * /
- - +
- < <= > >=
- == !=
- &&
- ||

## 64-bit support for configuration file keywords and expressions

Administrators can assign 64-bit integer values to selected configuration keywords.

**floating_resources**
Consumable resources associated with the **floating_resources** keyword may be assigned 64-bit integer values. Fractional and unit specifications are not allowed. The predefined ConsumableCpus, ConsumableMemory, ConsumableLargePageMemory, and ConsumableVirtualMemory may not be specified as floating resources.

**Example:**

```
floating_resources = spice2g6(9876543210123) db2_license(1234567890)
```

**MACHPRIO expression**
The LoadLeveler variables: Disk, ConsumableCpus, ConsumableMemory, ConsumableVirtualMemory, ConsumableLargePageMemory, PagesScanned, Memory, VirtualMemory, FreeRealMemory, and PagesFreed may be used in a MACHPRIO expression. They are 64-bit integers and 64-bit arithmetic is used to evaluate them.

**Example:**

```
MACHPRIO: (Memory + FreeRealMemory) - (LoadAvg*1000 + PagesScanned)
```

## Configuration keyword descriptions

This topic provides an alphabetical list of the keywords you can use in a LoadLeveler configuration.

It also provides examples of statements that use these keywords.

**ACCT**

Turns the accounting function on or off.

**Syntax:**

```
ACCT = flag ...
```

The available flags are:

**A_DETAIL**
: Enables extended accounting. Using this flag causes LoadLeveler to record detail resource consumption by machine and by events for each job step.

**A_ENERGY**
: Turns energy data recording on.

**A_RES**
: Turns reservation data recording on.

**A_OFF**
: Turns accounting data recording off.

**A_ON**
: Turns accounting data recording on. If specified without the **A_DETAIL** flag, the following is recorded:
  - The total amount of CPU time consumed by the entire job
  - The maximum memory consumption of all tasks (or nodes).

**A_VALIDATE**
: Turns account validation on.

**Default value: A_OFF**

**Examples:**

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

```
ACCT = A_ON A_DETAIL
```

This example specifies that accounting should be turned on and that extended accounting data and energy consumption data should be collected.

```
ACCT = A_ON A_DETAIL A_ENERGY
```

**ACTION_ON_SWITCH_TABLE_ERROR**

Points to an administrator supplied program that will be run when **DRAIN_ON_SWITCH_TABLE_ERROR** is set to **true** and a switch table unload error occurs.

**Syntax:**

```
ACTION_ON_SWITCH_TABLE_ERROR = program
```

**Default value:** The default is to not run a program.

**ADAPTER_HEARTBEAT_INTERVAL**

Specifies the amount of time, in seconds, that defines the heartbeat interval between the region manager and startd. This keyword is used by the resource manager component only.

**Syntax:**

```
ADAPTER_HEARTBEAT_INTERVAL = interval
```

where:

*interval* is in seconds.

**Default value:** The default is 30 seconds. If a value of 0 is specified, the default will be used.

**ADAPTER_HEARTBEAT_PORT**

Specifies the port number on which the region manager listens for heartbeats from startd. This keyword is used by the resource manager component only.

**Syntax:**

```
ADAPTER_HEARTBEAT_PORT = port
```

where:

*port* is a positive whole number.

**Default value:** The default is 9684.

**ADAPTER_HEARTBEAT_RETRIES**

Specifies the number of heartbeat intervals that the region manager will wait before declaring the adapter on startd as down. This keyword is used by the resource manager component only.

**Syntax:**

```
ADAPTER_HEARTBEAT_RETRIES = retries
```

where:

*retries* is a positive whole number.

**Default value:** The default is 2. If a value of 0 is specified, the default will be used.

**ADMIN_FILE**

Points to the administration file containing user, class, group, machine_group, machine, cluster, and region stanzas.This keyword is not used for the database configuration option.

**Syntax:**

```
ADMIN_FILE = directory
```

**Default value:** `$(tilde)/admin_file`

**AFS_GETNEWTOKEN**

Specifies a filter that, for example, can be used to refresh an AFS token.

**Syntax:**

```
AFS_GETNEWTOKEN = full_path_to_executable
```

Where *full_path_to_executable* is an administrator-supplied program that receives the AFS authentication information on standard input and writes the new information to standard output. The filter is run when the job is scheduled to run and can be used to refresh a token which expired when the job was queued.

**Default value:** The default is to not run a program.

**ARCH**

Indicates the standard architecture of the system. The architecture you specify here must be specified in the same format in the **requirements** and **preferences** statements in job command files. The administrator defines the character string for each architecture.

**Syntax:**

```
ARCH = string
```

**Default value:** Use the command **llrstatus -l** to view the default.

**Example:** To define a machine as an RS/6000®, the keyword would look like:

```
ARCH = R6000
```

**BIN**

Defines the directory where LoadLeveler binaries are kept.

**Syntax:**

```
BIN = $(RELEASEDIR)/bin
```

**Default value:** `$(tilde)/bin`

**CKPT_CLEANUP_INTERVAL**

Specifies the interval, in seconds, at which the **Schedd** daemon will run the program specified by the **CKPT_CLEANUP_PROGRAM** keyword.

**Syntax:**

```
CKPT_CLEANUP_INTERVAL = number
```

*number* must be a positive integer.

**Default value:** -1

**CKPT_CLEANUP_PROGRAM**

Identifies an administrator-provided program which is to be run at the interval specified by the **ckpt_cleanup_interval** keyword. The intent of this program is to delete old checkpoint files created by jobs running under LoadLeveler during the checkpoint process.

**Syntax:**

```
CKPT_CLEANUP_PROGRAM = program
```

Where *program* is the fully qualified name of the program to be run. The program must be accessible and executable by LoadLeveler.

A sample program to remove checkpoint files is provided in the `/usr/lpp/LoadL/full/samples/llckpt/rmckptfiles.c` file.

**Default value:** No default value is set.

**CKPT_EXECUTE_DIR**

Specifies the directory where the job step's executable will be saved for checkpointable jobs. You can specify this keyword in either the configuration keyword or the job command file; different file permissions are required depending on where this keyword is set. When used as a configuration keyword, it specifies a "base" directory. For each job step, a subdirectory with the name *job_step_id* is created and the job step's executable is copied to this subdirectory. For additional information, see "Planning considerations for checkpointing jobs" on page 79.

**Syntax:**

```
CKPT_EXECUTE_DIR = directory
```

This directory cannot be the same as the current location of the executable file, or LoadLeveler will not stage the executable. In this case, the user must have execute permission for the current executable file.

**Default value:** By default, the executable of a checkpointable job step is not staged.

**Note:** The staged executables are not deleted because they may be needed for restart operations. It is your responsibility to manage the files in the **ckpt_execute_dir** directory and remove any files that are no longer needed.

**CLASS**

Determines whether a machine will accept jobs of a certain job class. For parallel jobs, you must define a class instance for each task you want to run on a node using one of two formats:

- The format, **CLASS** = *class_name (count)*, defines the **CLASS** names using a statement that names the classes and sets the number of tasks for each class in parenthesis.

  With this format, the following rules apply:
  - Each class can have only one entry
  - If a class has more than one entry or there is a syntax error, the entire **CLASS** statement will be ignored
  - If the **CLASS** statement has a blank value or is not specified, it will be defaulted to **No_Class (1)**
  - The number of instances for a class specified inside the parenthesis **( )** must be an unsigned integer. If the number specified is 0, it is correct syntactically, but the class will not be defined in LoadLeveler
  - If the number of instances for all classes in the **CLASS** statement are 0, the default **No_Class(1)** will be used
- The format, **CLASS** = { "*class1*" "*class2*" "*class2*" "*class2*"}, defines the **CLASS** names using a statement that names each class and sets the number of tasks for each class based on the number of times that the class name is used inside the **{}** operands.

**Note:** With both formats, the class names list is blank delimited.

You can have a maximum of 1024 characters in the class statement. You cannot use **allclasses** or **data_stage** as a class name, since these are reserved LoadLeveler keywords.

**Syntax:**

```
CLASS = { "class_name" ... } | {"No_Class"} | class_name (count) ...
```

**Default value: {"No_Class"}**

**Note:** This keyword is deprecated in the configuration file and has been moved to the administration file.

**CLIENT_TIMEOUT**

Specifies the maximum time, in seconds, that a daemon waits for a response over TCP/IP from a process. If the waiting time exceeds the specified amount, the daemon tries again to communicate with the process. In general, you should use the default setting unless you are experiencing delays due to an excessively loaded network. If so, you should try increasing this value.

**Syntax:**

```
CLIENT_TIMEOUT = number
```

**Default value:** The default is 30 seconds.

**COMM**

Specifies a local directory where LoadLeveler keeps special files used for UNIX domain sockets for communicating among LoadLeveler daemons running on the same machine. This keyword allows the administrator to choose a different file system other than /tmp for these files. If you change the COMM option you must stop and then restart LoadLeveler using the **llrctl** command.

**Syntax:**

```
COMM = local directory
```

**Default value:** The default location for the files is **/tmp**.

**CONTINUE**

Determines whether suspended jobs should continue execution.

**Syntax:**

```
CONTINUE: expression that evaluates to T or F (true or false)
```

When **T**, suspended LoadLeveler jobs resume execution on the machine.

**Default value:** No default value is set.

For information about time-related variables that you may use for this keyword, see "Variables to use for setting times" on page 127.

**DCE_AUTHENTICATION_PAIR**

Specifies a pair of installation supplied programs that are used to authenticate DCE security credentials.

**Restriction:** DCE security is not supported by LoadLeveler for Linux.

**Syntax:**

```
DCE_AUTHENTICATION_PAIR = program1, program2
```

Where *program1* and *program2* are LoadLeveler- or installation-supplied programs that are used to authenticate DCE security credentials. *program1* obtains a handle (an opaque credentials object), at the time the job is submitted, which is used to authenticate to DCE. *program2* uses the handle obtained by *program1* to authenticate to DCE before starting the job on the executing machines.

**Default value:** See "Handling DCE security credentials" on page 50 for information about defaults.

**DRAIN_ON_SWITCH_TABLE_ERROR**

Specifies whether the **startd** should be drained when the switch table fails to unload. This will flag the administrator that intervention may be required to unload the switch table. When **DRAIN_ON_SWITCH_TABLE_ERROR** is set to true, the **startd** will be drained when the switch table fails to unload.

**Syntax:**

```
DRAIN_ON_SWITCH_TABLE_ERROR = true | false
```

**Default value: false**

**DSTG_MAX_STARTERS**

Specifies a machine-specific limit on the number of data staging initiators. Since each task of a data staging job step consumes one initiator from the

**data_stage** class on the specified machine, **DSTG_MAX_STARTERS** provides the maximum number of data staging tasks that can run at the same time on the machine.

**Syntax:**

```
DSTG_MAX_STARTERS = number
```

**Notes:**

1. If you have not set the **DSTG_MAX_STARTERS** value in either the global or local configuration files, there will not be any data staging initiators on the specified machine. In this configuration, the executing machine will not be allowed to perform data staging tasks.
2. The value specified for **DSTG_MAX_STARTERS** will be the number of initiators available for the built-in **data_stage** class on that machine.
3. The value specified for **MAX_STARTERS** will not limit the value specified for **DSTG_MAX_STARTERS**.

**Default value: 0**

**Note:** This keyword is deprecated in the configuration file and has been moved to the administration file.

**ENFORCE_RESOURCE_MEMORY**

Specifies whether the Workload Manager is configured to limit, as precisely as possible, the real memory usage of a WLM class. For this keyword to be valid, ConsumableMemory must be set through the **ENFORCE_RESOURCE_USAGE** keyword.

**Syntax:**

```
ENFORCE_RESOURCE_MEMORY = true | false
```

**Default value: false**

**ENFORCE_RESOURCE_POLICY**

Specifies what type of resource entitlements will be assigned to the Workload Manager classes. If the value specified is **shares**, it means a share value is assigned to the class based on the job step's requested resources (one unit of resource equals one share). This is the default policy. If the value specified is **soft**, it means a percentage value is assigned to the class based on the job step's requested resources and the total machine resources. This percentage can be exceeded if there is no contention for the resource. If the value specified is **hard**, it means a percentage value is assigned to the class based on the job step's requested resources and the total machine resources. This percentage cannot be exceeded regardless of the contention for the resource. This keyword is only valid for CPU and real memory with either shares or percent limits. If desired, this keyword can be used in the **LoadL_config.local** file to set up a different policy for each machine. The **ENFORCE_RESOURCE_USAGE** keyword must be set for this keyword to be valid.

**Syntax:**

```
ENFORCE_RESOURCE_POLICY = hard |soft | shares
```

**Default value: shares**

**ENFORCE_RESOURCE_USAGE**

Specifies whether the Workload Manager is used to enforce CPU and memory resources. This keyword accepts either a value of **deactivate** or a list of one or more of the following predefined resources:

- **ConsumableCpus**

- **ConsumableMemory**
- **ConsumableVirtualMemory**
- **ConsumableLargePageMemory**

Either memory or CPUs or both can be enforced. If **deactivate** is specified, LoadLeveler will deactivate Workload Manager on all the nodes in the LoadLeveler cluster.

**Syntax:**

```
ENFORCE_RESOURCE_USAGE = name name ... name | deactivate
```

**EXECUTE**

Specifies the local directory to store the executables of jobs submitted by other machines.

**Syntax:**

```
EXECUTE = local directory/execute
```

**Default value:** `$(tilde)/execute`

**EXT_ENERGY_POLICY_PROGRAM**

Specifies a user-supplied executable to be run by Startd to determine the CPU frequency to use when running the job. For a job that has the energy function enabled, LoadLeveler will pass the job policy tag **LL_ENERGY_TAG_NAME** environment variable to the program. The output of the program is the frequency value the job will use.

When this keyword is enabled, LoadLeveler will ignore the **MAX_PERF_DECREASE_ALLOWED**, **ENERGY_SAVING_REQ**, and **ADJUST_WALL_CLOCK_LIMIT** keywords that are defined in the job command file. To accommodate the longer running time at a lower frequency, ensure the **WALL_CLOCK_LIMIT** is set high enough or change it by using the **llmodify** command.

**Syntax:**

```
EXT_ENERGY_POLICY_PROGRAM = full_path_to_executable
```

where:

*full_path_to_executable*
: Is a user-supplied program that calculates the frequency the job will use.

**Default value:** NULL

**FAILOVER_HEARTBEAT_INTERVAL**

Specifies the amount of time, in seconds, that defines how frequently the primary and alternate central manager, resource manager, or region manager communicate with each other.

**Syntax:**

```
FAILOVER_HEARTBEAT_INTERVAL = seconds
```

**Default value:** The default value is 300 seconds or 5 minutes.

**FAILOVER_HEARTBEAT_RETRIES**

Specifies the number of heartbeat intervals that an alternate resource manager will wait before declaring that the primary central manager, resource manager, or region manager is not operating.

**Syntax:**

```
FAILOVER_HEARTBEAT_RETRIES = number
```

**Default value:** The default value is 6.

**FEATURE**

Specifies an optional characteristic can be used to match jobs with machines. You can specify unique characteristics for any machine using this keyword. You can have a maximum of 1024 characters in the feature statement.

**Syntax:**

```
Feature = {"string" ...}
```

**Default value:** No default value is set.

**Example:** If a machine has licenses for installed products ABC and XYZ in the local configuration file, you can enter the following:

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

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

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

**Note:** This keyword is deprecated in the configuration file and has been moved to the administration file.

**FLOATING_RESOURCES**

Specifies which consumable resources are available collectively on all of the machines in the LoadLeveler cluster. The count for each resource must be an integer greater than or equal to zero, and each resource can only be specified once in the list. If a resource is specified incorrectly with the **FLOATING_RESOURCES** keyword, then that resource will be ignored. All other correctly specified resources will be accepted. **ConsumableCpus**, **ConsumableMemory**, **ConsumableVirtualMemory**, and **ConsumableLargePageMemory** may not be specified as floating resources.

**Syntax:**

```
FLOATING_RESOURCES = name(count) name(count) ... name(count)
```

**Default value:** No default value is set.

**FS_INTERVAL**

Defines the number of minutes used as the interval for checking free file system space or inodes. If your file system receives many log messages or copies large executables to the LoadLeveler spool, the file system will fill up quicker and you should perform file size checking more frequently by setting the interval to a smaller value. LoadLeveler will not check the file system if the value of FS_INTERVAL is:

- Set to zero
- Set to a negative integer

**Syntax:**

```
FS_INTERVAL = minutes
```

**Default value:** If FS_INTERVAL is not specified but any of the other file-system keywords (FS_NOTIFY, FS_SUSPEND, FS_TERMINATE, INODE_NOTIFY, INODE_SUSPEND, INODE_TERMINATE) are specified, the FS_INTERVAL value will default to 5 and the file system will be checked. If no file-system or inode keywords are set, LoadLeveler does not monitor file systems at all.

For more information related to using this keyword, see “Setting up file system monitoring” on page 38.

**FS_NOTIFY**

Defines the lower and upper amounts, in bytes, of free file-system space at which LoadLeveler is to notify the administrator:

- If the amount of free space becomes less than the lower threshold value, LoadLeveler sends a mail message to the administrator indicating that logging problems may occur.
- When the amount of free space becomes greater than the upper threshold value, LoadLeveler sends a mail message to the administrator indicating that problem has been resolved.

**Syntax:**

```
FS_NOTIFY = lower threshold, upper threshold
```

Specify space in bytes with the unit B. A metric prefix such as K, M, or G may precede the B. The valid range for both the lower and upper thresholds are -1B and all positive integers. If the value is set to -1, the transition across the threshold is not checked.

**Default value:** In bytes: 1KB, -1B

For more information related to using this keyword, see "Setting up file system monitoring" on page 38.

**FS_SUSPEND**

Defines the lower and upper amounts, in bytes, of free file system space at which LoadLeveler drains and resumes the Schedd and startd daemons running on a node.

- If the amount of free space becomes less than the lower threshold value, then LoadLeveler drains the Schedd and the startd daemons if they are running on a node. When this happens, logging is turned off and mail notification is sent to the administrator.
- When the amount of free space becomes greater than the upper threshold value, LoadLeveler signals the Schedd and the startd daemons to resume. When this happens, logging is turned on and mail notification is sent to the administrator.

**Syntax:**

```
FS_SUSPEND = lower threshold, upper threshold
```

Specify space in bytes with the unit B. A metric prefix such as K, M, or G may precede the B. The valid range for both the lower and upper thresholds are -1B and all positive integers. If the value is set to -1, the transition across the threshold is not checked.

**Default value:** In bytes: -1B, -1B

For more information related to using this keyword, see "Setting up file system monitoring" on page 38.

**FS_TERMINATE**

Defines the lower and upper amounts, in bytes, of free file system space at which LoadLeveler is terminated. This keyword sends the SIGTERM signal to the Master daemon which then terminates all LoadLeveler daemons running on the node.

- If the amount of free space becomes less than the lower threshold value, all LoadLeveler daemons are terminated.
- An upper threshold value is required for this keyword. However, since LoadLeveler has been terminated at the lower threshold, no action occurs.

**Syntax:**

```
FS_TERMINATE = lower threshold, upper threshold
```

Specify space in bytes with the unit B. A metric prefix such as K, M, or G may precede the B. The valid range for the lower threshold is -1B and all positive integers. If the value is set to -1, the transition across the threshold is not checked.

**Default value:** In bytes: -1B, -1B

For more information related to using this keyword, see "Setting up file system monitoring" on page 38.

**GLOBAL_HISTORY**
Identifies the directory that will contain the global history files produced by **llracctmgr** command when no directory is specified as a command argument.

**Syntax:**

```
GLOBAL_HISTORY = directory
```

**Default value:** The default value is **$(SPOOL)** (the local spool directory).

For more information related to using this keyword, see "Collecting the accounting information and storing it into files" on page 45.

**HISTORY**
Defines the path name where a file containing the history of local LoadLeveler jobs is kept.

**Syntax:**

```
HISTORY = directory
```

**Default value: $(SPOOL)/history**

For more information related to using this keyword, see "Collecting the accounting information and storing it into files" on page 45.

**HISTORY_PERMISSION**
Specifies the owner, group, and world permissions of the history file associated with a **LoadL_schedd** daemon.

**Syntax:**

```
HISTORY_PERMISSION = permissions | rw-rw----
```

*permissions* must be a string with a length of nine characters and consisting of the characters, **r**, **w**, **x**, or **-**.

**Default value:** The default settings are 660 (**rw-rw----**). **LoadL_schedd** will use the default setting if the specified permission are less than **rw-------**.

**Example:** A specification such as `HISTORY_PERMISSION = rw-rw-r--` will result in permission settings of 664.

**INODE_NOTIFY**
Defines the lower and upper amounts, in inodes, of free file-system inodes at which LoadLeveler is to notify the administrator:
- If the number of free inodes becomes less than the lower threshold value, LoadLeveler sends a mail message to the administrator indicating that logging problems may occur.
- When the number of free inodes becomes greater than the upper threshold value, LoadLeveler sends a mail message to the administrator indicating that problem has been resolved.

**Syntax:**

```
INODE_NOTIFY = lower threshold, upper threshold
```

The valid range for both the lower and upper thresholds are -1 and all positive integers. If the value is set to -1, the transition across the threshold is not checked.

**Default value:** In inodes: 1000, -1

For more information related to using this keyword, see "Setting up file system monitoring" on page 38.

**INODE_SUSPEND**

Defines the lower and upper amounts, in inodes, of free file system inodes at which LoadLeveler drains and resumes the Schedd and startd daemons running on a node.

- If the number of free inodes becomes less than the lower threshold value, then LoadLeveler drains the Schedd and the startd daemons if they are running on a node. When this happens, logging is turned off and mail notification is sent to the administrator.
- When the number of free inodes becomes greater than the upper threshold value, LoadLeveler signals the Schedd and the startd daemons to resume. When this happens, logging is turned on and mail notification is sent to the administrator.

**Syntax:**

```
INODE_SUSPEND = lower threshold, upper threshold
```

The valid range for both the lower and upper thresholds are -1 and all positive integers. If the value is set to -1, the transition across the threshold is not checked.

**Default value:** In inodes: -1, -1

For more information related to using this keyword, see "Setting up file system monitoring" on page 38.

**INODE_TERMINATE**

Defines the lower and upper amounts, in inodes, of free file system inodes at which LoadLeveler is terminated. This keyword sends the SIGTERM signal to the Master daemon which then terminates all LoadLeveler daemons running on the node.

- If the number of free inodes becomes less than the lower threshold value, all LoadLeveler daemons are terminated.
- An upper threshold value is required for this keyword. However, since LoadLeveler has been terminated at the lower threshold, no action occurs.

**Syntax:**

```
INODE_TERMINATE = lower threshold, upper threshold
```

The valid range for the lower threshold is -1 and all positive integers. If the value is set to -1, the transition across the threshold is not checked.

**Default value:** In inodes: -1, -1

For more information related to using this keyword, see "Setting up file system monitoring" on page 38.

**JOB_ACCT_Q_POLICY**

Specifies the amount of time, in seconds, that determines how often the startd daemon updates the Schedd daemon with accounting data of running jobs.

**Syntax:**

```
JOB_ACCT_Q_POLICY = number
```

**Default value:** The default is 300.

For more information related to using this keyword, see "Gathering job accounting data" on page 43.

**JOB_EPILOG**
Path name of the epilog program.

**Syntax:**

```
JOB_EPILOG = program name
```

**Default value:** No default value is set.

For more information related to using this keyword, see "Writing prolog and epilog programs" on page 52.

**JOB_LIMIT_POLICY**
Specifies the amount of time, in seconds, that LoadLeveler checks to see if **job_cpu_limit** has been exceeded. The smaller of **JOB_LIMIT_POLICY** and **JOB_ACCT_Q_POLICY** is used to control how often the **startd** daemon collects resource consumption data on running jobs, and how often the **job_cpu_limit** is checked.

**Syntax:**

```
JOB_LIMIT_POLICY = number
```

**Default value:** The default is 300.

**JOB_PROLOG**
Path name of the prolog program.

**Syntax:**

```
JOB_PROLOG = program name
```

**Default value:** No default value is set.

For more information related to using this keyword, see "Writing prolog and epilog programs" on page 52.

**JOB_USER_EPILOG**
Path name of the user epilog program.

**Syntax:**

```
JOB_USER_EPILOG = program name
```

**Default value:** No default value is set.

For more information related to using this keyword, see "Writing prolog and epilog programs" on page 52.

**JOB_USER_PROLOG**
Path name of the user prolog program.

**Syntax:**

```
JOB_USER_PROLOG = program name
```

**Default value:** No default value is set.

For more information related to using this keyword, see "Writing prolog and epilog programs" on page 52.

**KBDD**
Location of kbdd executable (**LoadL_kbdd**).

**Syntax:**

```
KBDD = directory
```

**Default value:** `$(BIN)/LoadL_kbdd`

**KBDD_COREDUMP_DIR**

Local directory for storing **LoadL_kbdd** daemon core dump files.

**Syntax:**

```
KBDD_COREDUMP_DIR = directory
```

**Default value:** The **/tmp** directory.

For more information related to using this keyword, see "Specifying file and directory locations" on page 30.

**KILL**

Determines whether or not vacated jobs should be sent the SIGKILL signal and replaced in the queue. It is used to remove a job that is taking too long to vacate.

**Syntax:**

```
KILL: expression that evaluates to T or F (true or false)
```

When **T**, vacated LoadLeveler jobs are removed from the machine with no attempt to take checkpoints.

For information about time-related variables that you may use for this keyword, see "Variables to use for setting times" on page 127.

**LL_RSH_COMMAND**

Specifies an administrator provided executable to be used by **llrctl start** when starting LoadLeveler on remote machines in the administration file. The **LL_RSH_COMMAND** keyword is any executable that can be used as a substitute for **/usr/bin/rsh**. The **llrctl start** command passes arguments to the executable specified by **LL_RSH_COMMAND** in the following format:

```
LL_RSH_COMMAND hostname -n llrctl start options
```

**Syntax:**

```
LL_RSH_COMMAND = full_path_to_executable
```

**Default value: /usr/bin/rsh**. This keyword must specify the full path name to the executable provided. If no value is specified, LoadLeveler will use **/usr/bin/rsh** as the default when issuing a start. If an error occurred while locating the executable specified, an error message will be displayed.

**Example:** This example shows that using the secure shell (**/usr/bin/ssh**) is the preferred method for the **llrctl start** command to communicate with remote nodes. Specify the following in the configuration file:

```
LL_RSH_COMMAND=/usr/bin/ssh
```

**LOADL_ADMIN**

Specifies a list of LoadLeveler administrators.

**Syntax:**

```
LOADL_ADMIN = list of user names
```

Where *list of user names* is a blank-delimited list of those individuals who will have administrative authority.

**Default value:** No default value is set, which means no one has administrator authority until this keyword is defined with one or more user names.

**Example:** To grant administrative authority to users bob and mary, enter the following in the configuration file:

```
LOADL_ADMIN = bob mary
```

For more information related to using this keyword, see "Defining LoadLeveler administrators" on page 29.

**LOCAL_CONFIG**

Specifies the path name of the optional local configuration file containing information specific to a node in the LoadLeveler network. This keyword is not used for the database configuration option.

**Syntax:**

```
LOCAL_CONFIG = directory
```

**Default value:** No default value is set.

**Examples:**

- If you are using a distributed file system like NFS, some examples are:

  ```
  LOCAL_CONFIG = $(tilde)/$(host).LoadL_config.local
  LOCAL_CONFIG = $(tilde)/LoadL_config.$(host).$(domain)
  LOCAL_CONFIG = $(tilde)/LoadL_config.local.$(hostname)
  ```

  See "LoadLeveler variables" on page 123 for information about the **tilde**, **host**, and **domain** variables.
- If you are using a local file system, an example is:

  ```
  LOCAL_CONFIG = /var/LoadL/LoadL_config.local
  ```

**LOG**

Defines the local directory to store log files. It is not necessary to keep all the log files created by the various LoadLeveler daemons and programs in one directory, but you will probably find it convenient to do so.

**Syntax:**

```
LOG = local directory/log
```

**Default value:** `$(tilde)/log`

**LOG_MESSAGE_THRESHOLD**

Specifies the maximum amount of memory, in bytes, for the message queue. Messages in the queue are waiting to be written to the log file. When the message logging thread cannot write messages to the log file as fast as they arrive, the memory consumed by the message queue can exceed the threshold. In this case, LoadLeveler will curtail logging by turning off all debug flags except **D_ALWAYS**, therefore, reducing the amount of logging that takes place. If the threshold is exceeded by the curtailed message queue, message logging is stopped. Special log messages are written to the log file, which indicate that some messages are missing. Mail is also sent to the administrator indicating that messages are missing. A value of -1 for this keyword will turn off the buffer threshold meaning that the threshold is unlimited.

**Syntax:**

```
LOG_MESSAGE_THRESHOLD = bytes
```

**Default value:** 20*1024*1024 (bytes)

**MACHINE_AUTHENTICATE**

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

When set to **true**, every communication between LoadLeveler processes will verify that the sending process is running on a machine which is identified via a machine stanza in the administration file. The validation is done by capturing the address of the sending machine when the **accept** function call is issued to accept a connection. The **gethostbyaddr** function is called to translate the address to a name, and the name is matched with the list derived from the administration file.

**Syntax:**

```
MACHINE_AUTHENTICATE = true | false
```

**Default value: false**

For more information related to using this keyword, see "Defining a LoadLeveler cluster" on page 29.

**MACHINE_UPDATE_INTERVAL**

Specifies the time, in seconds, during which machines must report to the central manager.

**Syntax:**

```
MACHINE_UPDATE_INTERVAL = number
```

Where *number* specifies the time period, in seconds, during which machines must report to the central manager. Machines that do not report in this number of seconds are considered *down*. *number* must be a numerical value and cannot be an arithmetic expression.

**Default value:** The default is 300 seconds.

**MAIL**

Name of a local mail program used to override default mail notification.

**Syntax:**

```
MAIL = program name
```

**Default value:** No default value is set.

For more information related to using this keyword, see "Using your own mail program" on page 57.

**MASTER**

Location of the master executable (**LoadL_master**).

**Syntax:**

```
MASTER = file
```

**Default value:** `$(BIN)/LoadL_master`

For more information related to using this keyword, see "How LoadLeveler daemons process jobs" on page 6.

**MASTER_COREDUMP_DIR**

Local directory for storing **LoadL_master** core dump files.

**Syntax:**

```
MASTER_COREDUMP_DIR = directory
```

**Default value:** The **/tmp** directory.

For more information related to using this keyword, see "Specifying file and directory locations" on page 30.

**MASTER_DGRAM_PORT**

The port number used when connecting to the daemon.

**Syntax:**

```
MASTER_DGRAM_PORT = port number
```

**Default value:** The default is 9617.

For more information related to using this keyword, see "Defining network characteristics" on page 30.

**MASTER_STREAM_PORT**

Specifies the port number to be used when connecting to the daemon.

**Syntax:**

```
MASTER_STREAM_PORT = port number
```

**Default value:** The default is 9616.

For more information related to using this keyword, see "Defining network characteristics" on page 30.

**MAX_CKPT_INTERVAL**

The maximum number of seconds between checkpoints for running jobs.

**Syntax:**

```
MAX_CKPT_INTERVAL = number
```

where: *number* specifies the maximum period in seconds between checkpoints taken for running jobs.

**Default value:** 7200 (2 hours)

For more information related to using this keyword, see "LoadLeveler support for checkpointing jobs" on page 78.

**MAX_STARTERS**

Specifies the maximum number of tasks that can run simultaneously on a machine. In this case, a task can be a serial job step or a parallel task. **MAX_STARTERS** defines the number of initiators on the machine (the number of tasks that can be initiated from a **startd**).

**Syntax:**

```
MAX_STARTERS = number
```

**Default value:** If this keyword is not specified, the default is the number of elements in the **Class** statement.

**Note:** This keyword is deprecated in the configuration file and has been moved to the administration file.

**MIN_CKPT_INTERVAL**

The minimum number of seconds between checkpoints for running jobs.

**Syntax:**

```
MIN_CKPT_INTERVAL = number
```

where: *number* specifies the initial period, in seconds, between checkpoints taken for running jobs.

**Default value:** 900 (15 minutes)

For more information related to using this keyword, see "LoadLeveler support for checkpointing jobs" on page 78.

**OBITUARY_LOG_LENGTH**

Specifies the number of lines from the end of the file that are appended to the mail message. The master daemon mails this log to the LoadLeveler administrators when one of the daemons dies.

**Syntax:**

```
OBITUARY_LOG_LENGTH = number
```

*number* must be a numerical value and cannot be an arithmetic expression.

**Default value:** The default is 25.

**POLLING_FREQUENCY**

Specifies the interval, in seconds, with which the startd daemon evaluates the load on the local machine and decides whether to suspend, resume, or abort jobs. This time is also the minimum interval at which the kbdd daemon reports keyboard or mouse activity to the startd daemon.

**Syntax:**

```
POLLING_FREQUENCY = number
```

*number* must be a numerical value and cannot be an arithmetic expression.

**Default value:** The default is 300.

**POLLS_PER_UPDATE**

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

**Syntax:**

```
POLLS_PER_UPDATE = number
```

*number* must be a numerical value and cannot be an arithmetic expression.

**Default value:** The default is 1.

**PRESTARTED_STARTERS**

Specifies how many prestarted starter processes LoadLeveler will maintain on an execution node to manage jobs when they arrive. The startd daemon starts the number of starter processes specified by this keyword. You may specify this keyword in either the global or local configuration file.

**Syntax:**

```
PRESTARTED_STARTERS = number
```

*number* must be less than or equal to the value specified through the MAX_STARTERS keyword. If the value of PRESTARTED_STARTERS specified is greater then MAX_STARTERS, LoadLeveler records a warning message in the startd log and assigns PRESTARTED_STARTERS the same value as MAX_STARTERS.

If the value PRESTARTED_STARTERS is zero, no starter processes will be started before jobs arrive on the execution node.

**Default value:** The default is 1.

**Note:** This keyword is deprecated in the configuration file and has been moved to the administration file.

**PROCESS_TRACKING**

Specifies whether or not LoadLeveler will cancel any processes (throughout the entire cluster), left behind when a job terminates.

**Syntax:**

```
PROCESS_TRACKING = TRUE | FALSE
```

When **TRUE** ensures that when a job is terminated, no processes created by the job will continue running.

**Note:** It is necessary to set this keyword to **true** to do preemption by the suspend method with the BACKFILL or API scheduler.

**Default value: FALSE**

**PROCESS_TRACKING_EXTENSION**

Specifies the directory containing the kernel module **LoadL_pt_ke** (AIX) or **proctrk.ko** (Linux).

**Syntax:**

```
PROCESS_TRACKING_EXTENSION = directory
```

**Default value:** The directory **$HOME/bin**

For more information related to using this keyword, see "Tracking job processes" on page 49.

**PUBLISH_OBITUARIES**

Specifies whether or not the master daemon sends mail to the administrator when any daemon it manages ends abnormally. When set to **true**, this keyword specifies that the master daemon sends mail to the administrators identified by **LOADL_ADMIN** keyword.

**Syntax:**

```
PUBLISH_OBITUARIES = true | false
```

**Default value: true**

**RAS_MSG_FILE_DIR**

Specifies the directory where the Reliability, Availability, and Serviceability (RAS) message files are stored. For a database cluster, this directory will be used as the temporary space when writing RAS messages into the database.

**Syntax:**

```
RAS_MSG_FILE_DIR = directory
```

**Default value: $LOG**

**REGION_MGR**

Specifies the location of the region manager executable (**LoadL_region_mgr**). This keyword is used by the resource manager component only.

**Syntax:**

```
REGION_MGR = file
```

**Default value:** `$(BIN)/LoadL_region_mgr`

**REGION_MGR_COREDUMP_DIR**

Local directory for storing **LoadL_region_mgr** core dump files. This keyword is used by the resource manager component only.

**Syntax:**

```
REGION_MGR_COREDUMP_DIR = directory
```

**Default value:** The **/tmp** directory.

For more information related to using this keyword, see "Specifying file and directory locations" on page 30.

**REGION_MGR_DGRAM_PORT**
Specifies the port number used when connecting to the daemon. This keyword is used by the resource manager component only.

**Syntax:**

```
REGION_MGR_DGRAM_PORT = port number
```

**Default value:** The default is 9682.

For more information related to using this keyword, see "Defining network characteristics" on page 30.

**REGION_MGR_STREAM_PORT**
Specifies the port number used when connecting to the daemon. This keyword is used by the resource manager component only.

**Syntax:**

```
REGION_MGR_STREAM_PORT = port number
```

**Default value:** The default is 9680.

For more information related to using this keyword, see "Defining network characteristics" on page 30.

**REJECT_ON_RESTRICTED_LOGIN**
Specifies whether the user's account status will be checked on every node where the job will be run by calling the AIX **loginrestrictions** function with the **S_DIST_CLNT** flag.

**Restriction:** Login restriction checking is ignored by LoadLeveler for Linux.

Login restriction checking includes:
- Does the account still exist?
- Is the account locked?
- Has the account expired?
- Do failed login attempts exceed the limit for this account?
- Is login disabled via **/etc/nologin**?

If the AIX **loginrestrictions** function indicates a failure then the user's job will be rejected and will be processed according to the LoadLeveler configuration parameters **MAX_JOB_REJECT** and **ACTION_ON_MAX_REJECT**.

**Syntax:**

```
REJECT_ON_RESTRICTED_LOGIN = true | false
```

**Default value: false**

**RELEASEDIR**
Defines the directory where all the LoadLeveler software resides.

**Syntax:**

```
RELEASEDIR = release directory
```

**Default value: $(RELEASEDIR)**

**RESOURCE_MGR**
Specifies the location of the resource manager daemon executable (LoadL_resource_mgr). This keyword is used by the resource manager component only.

**Syntax:**

```
RESOURCE_MGR = file
```

**Default value:** $(BIN)/LoadL_resource_mgr

**RESOURCE_MGR_COREDUMP_DIR**

Specifies the local directory for storing LoadL_resource_mgr core dump files. This keyword is used by the resource manager component only.

**Syntax:**

```
RESOURCE_MGR_COREDUMP_DIR = directory
```

**Default value:** The `/tmp` directory

**RESOURCE_MGR_DGRAM_PORT**

Specifies the port number used when connecting to the daemon. This keyword is used by the resource manager component only.

**Syntax:**

```
RESOURCE_MGR_DGRAM_PORT = port_number
```

**Default value:** The default is 9619

**RESOURCE_MGR_LIST**

Specifies the machines where the primary and alternate resource manager daemons run. If no machines are specified, the central manager list will be used for the resource manager list. This keyword is used by the resource manager component only.

**Syntax:**

```
RESOURCE_MGR_LIST = primary_resource_manager_machine \
                    [alternate_resource_manager_machine_list]
```

where:

*primary_resource_manager_machine* is the hostname of the machine on which the primary resource manager daemon will run.

*[alternate_resource_manager_machine_list]* is a blank-delimited list of hostnames for the alternate resource manager daemons.

**RESOURCE_MGR_STREAM_PORT**

Specifies the port number used when connecting to the daemon. This keyword is used by the resource manager component only.

**Syntax:**

```
RESOURCE_MGR_STREAM_PORT = port_number
```

**Default value:** The default value is 9618.

**RESTARTS_PER_HOUR**

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

**Syntax:**

```
RESTARTS_PER_HOUR = number
```

*number* must be a numerical value and cannot be an arithmetic expression.

**Default value:** The default is 12.

**RESUME_ON_SWITCH_TABLE_ERROR_CLEAR**

Specifies whether or not the **startd** that was drained when the switch table failed to unload will automatically resume once the unload errors are cleared. The unload error is considered cleared after LoadLeveler can successfully unload the switch table. For this keyword to work, the **DRAIN_ON_SWITCH_TABLE_ERROR** option in the configuration file must be turned on and not disabled. Flushing, suspending, or draining of a **startd** manually or automatically will disable this option until the **startd** is manually resumed.

**Syntax:**

```
RESUME_ON_SWITCH_TABLE_ERROR_CLEAR = true | false
```

**Default value: false**

| **RESUME_WAIT_TIME**

| Controls the wait time for a suspended node to resume before its power state
| is set to error.

| **Syntax:**

```
| RESUME_WAIT_TIME = the_number_of_seconds
```

| Where *the_number_of_seconds* is used to specify the maximum amount of time
| in seconds the resource manager will wait for the suspended node to resume.
| The suspended node's power state will be set to error if it cannot resume in
| *the_number_of_seconds*.

| **Default value:** The default is 180.

| **Example:**

| In the following example, if the suspended node does not resume within 300
| seconds, the resource manager will set its power state to error:

```
| RESUME_WAIT_TIME = 300
```

**RSET_SUPPORT**

Indicates the level of RSet support present on a machine.

**Syntax:**

```
RSET_SUPPORT = option
```

The available options are:

**RSET_MCM_AFFINITY**
: Indicates that the machine can run jobs requesting MCM (memory or adapter) and processor affinity.

**RSET_NONE**
: Indicates that LoadLeveler RSet support is not available on the machine.

**RSET_USER_DEFINED**
: Indicates that the machine can be used for jobs with a user-created RSet in their job command file.

**Default value: RSET_NONE**

**SAVELOGS**

Specifies the directory in which log files are archived.

**Syntax:**

```
SAVELOGS = directory
```

Where *directory* is the directory in which log files will be archived.

**Default value:** No default value is set.

For more information related to using this keyword, see "Configuring recording activity and log files" on page 32.

**SAVELOGS_COMPRESS_PROGRAM**

Compresses logs after they are copied to the **SAVELOGS** directory. If not specified, **SAVELOGS** are copied, but are not compressed.

**Syntax:**

```
SAVELOGS_COMPRESS_PROGRAM = program
```

Where *program* is a specific executable program. It can be a system-provided facility (such as, **/bin/gzip**) or an administrator-provided executable program. The value must be a full path name and can contain command-line arguments. LoadLeveler will call the program as: *program filename*.

**Default value:** If blank, the logs are not compressed.

**Example:** In this example, LoadLeveler will run the **gzip -f** command. The log file in **SAVELOGS** will be compressed after it is copied to **SAVELOGS**. If the *program* cannot be found or is not executable, LoadLeveler will log the error and *SAVELOGS* will remain uncompressed.

```
SAVELOGS_COMPRESS_PROGRAM = /bin/gzip -f
```

**SCHEDD**

Location of the Schedd executable (**LoadL_schedd**).

**Syntax:**

```
SCHEDD = file
```

**Default value:** `$(BIN)/LoadL_schedd`

For more information related to using this keyword, see "How LoadLeveler daemons process jobs" on page 6.

**SCHEDD_COREDUMP_DIR**

Specifies the local directory for storing **LoadL_schedd** core dump files.

**Syntax:**

```
SCHEDD_COREDUMP_DIR = directory
```

**Default value:** The **/tmp** directory.

For more information related to using this keyword, see "Specifying file and directory locations" on page 30.

**SCHEDD_INTERVAL**

Specifies the interval, in seconds, at which the Schedd daemon checks the local job queue and updates the negotiator daemon.

**Syntax:**

```
SCHEDD_INTERVAL = number
```

*number* must be a numerical value and cannot be an arithmetic expression.

**Default value:** The default is 60 seconds.

**SCHEDD_RUNS_HERE**

Specifies whether the Schedd daemon runs on the host. If you do not want to run the Schedd daemon, specify **false**.

This keyword does not designate a machine as a public scheduling machine. Unless configured as a public scheduling machine, a machine configured to run the Schedd daemon will only accept job submissions from the same machine running the Schedd daemon. A public scheduling machine accepts job submissions from other machines in the LoadLeveler cluster. To configure a machine as a public scheduling machine, see the **schedd_host** keyword description in "Administration keyword descriptions" on page 134.

**Syntax:**

```
SCHEDD_RUNS_HERE = true | false
```

**Default value: true**

**Note:** This keyword is deprecated in the configuration file and has been moved to the administration file.

**SCHEDD_SUBMIT_AFFINITY**

Specifies whether job submissions are directed to a locally running Schedd daemon. When the keyword is set to **true**, job submissions are directed to a Schedd daemon running on the same machine where the submission takes place, provided there is a Schedd daemon running on that machine. In this case the submission is said to have "affinity" for the local Schedd daemon. If there is no Schedd daemon running on the machine where the submission takes place, or if this keyword is set to **false**, the job submission will only be directed to a Schedd daemon serving as a public scheduling machine. In this case, if there are no public scheduling machines configured the job cannot be submitted. A public scheduling machine accepts job submissions from other machines in the LoadLeveler cluster. To configure a machine as a public scheduling machine, see the **schedd_host** keyword description in "Administration keyword descriptions" on page 134.

**Syntax:**

```
SCHEDD_SUBMIT_AFFINITY = true | false
```

**Default value: true**

**SCHEDD_STATUS_PORT**

Specifies the port number used when connecting to the daemon.

**Syntax:**

```
SCHEDD_STATUS_PORT = port number
```

**Default value:** The default is 9606.

For more information related to using this keyword, see "Defining network characteristics" on page 30.

**SCHEDD_STREAM_PORT**

Specifies the port number used when connecting to the daemon.

**Syntax:**

```
SCHEDD_STREAM_PORT = port number
```

**Default value:** The default is 9605.

For more information related to using this keyword, see "Defining network characteristics" on page 30.

**SEC_ADMIN_GROUP**

When security services are enabled, this keyword points to the name of the UNIX group that contains the local identities of the LoadLeveler administrators.

**Restriction:** CtSec security is not supported on LoadLeveler for Linux.

**Syntax:**

```
SEC_ADMIN_GROUP = name of lladmin group
```

**Default value:** No default value is set.

For more information related to using this keyword, see "Configuring LoadLeveler to use cluster security services" on page 40.

**SEC_ENABLEMENT**
Specifies the security mechanism to be used.

**Restriction:** Do not set this keyword to CtSec in the configuration file for a Linux machine. CtSec security is not supported on LoadLeveler for Linux.

**Syntax:**

```
SEC_ENABLEMENT = COMPAT | CTSEC
```

**Default value:** No default value is set.

**SEC_SERVICES_GROUP**
When security services are enabled, this keyword specifies the name of the LoadLeveler services group.

**Restriction:** CtSec security is not supported on LoadLeveler for Linux.

**Syntax:**

```
SEC_SERVICES_GROUP=group name
```

Where *group name* defines the identities of the LoadLeveler daemons.

**Default value:** No default value is set.

**SEC_IMPOSED_MECHS**
Specifies a blank-delimited list of LoadLeveler's permitted security mechanisms when Cluster Security (CtSec) services are enabled.

**Restriction:** CtSec security is not supported on LoadLeveler for Linux.

**Syntax:** Specify a blank delimited list containing combinations of the following values:

**none** If this is the only value specified, then users *will* run unauthenticated and, if authorization is necessary, the job will fail. If this is not the only value specified, then users *may* run unauthenticated and, if authorization is necessary, the job will fail.

**unix** If this is the only value specified, then UNIX host-based authentication will be used; otherwise, other mechanisms may be used.

**Default value:** No default value is set.

**Example:**

```
SEC_IMPOSED_MECHS = none unix
```

**SPOOL**
Defines the local directory where LoadLeveler keeps the local job queue and checkpoint files.

**Syntax:**

```
SPOOL = local directory/spool
```

**Default value:** `$(tilde)/spool`

**STALE_ENERGY_TAG_CLEANUP**

Removes the energy tag from the database automatically when an energy tag has not been referenced in the last *the_number_of_days*. LoadLeveler will not remove the energy tag by default.

**Syntax:**

```
STALE_ENERGY_TAG_CLEANUP = the_number_of_days
```

where:

*the_number_of_days*
: Indicates the days that the energy tag has not been used.

**Default value:** -1.

**Example:**

Remove energy tags that have not been used in 30 days from the database:

```
STALE_ENERGY_TAG_CLEANUP = 30
```

**START**

Determines whether a machine can run a LoadLeveler job.

**Syntax:**

```
START: expression that evaluates to T or F (true or false)
```

When the expression evaluates to **T**, LoadLeveler considers dispatching a job to the machine. When you use a START expression that is based on the CPU load average, the negotiator may evaluate the expression as **F** even though the load average indicates the machine is Idle. When the expression evaluates to **F**, the LoadLeveler resource manager will not start a job on the machine.

**Default value:** No default value is set, which means that no jobs will be started.

For information about time-related variables that you may use for this keyword, see "Variables to use for setting times" on page 127.

**START_DAEMONS**

Specifies whether to start the LoadLeveler daemons on the node.

**Syntax:**

```
START_DAEMONS = true | false
```

**Default value: true**

When **true**, the daemons are started. In most cases, you will probably want to set this keyword to **true**. The individual users would modify their local configuration files and set this keyword to **false**. Because the global configuration file has the keyword set to **true**, their individual machines would still be able to participate in the LoadLeveler cluster.

Also, to define the machine as strictly a submit-only machine, set this keyword to **false**.

**STARTD**

Location of the startd executable (**LoadL_startd**).

**Syntax:**

```
STARTD = file
```

**Default value:** `$(BIN)/LoadL_startd`

For more information related to using this keyword, see “How LoadLeveler daemons process jobs” on page 6.

**STARTD_COREDUMP_DIR**
Local directory for storing **LoadL_startd** core dump files.

**Syntax:**

```
STARTD_COREDUMP_DIR = directory
```

**Default value:** The **/tmp** directory.

For more information related to using this keyword, see “Specifying file and directory locations” on page 30.

**STARTD_DGRAM_PORT**
Specifies the port number used when connecting to the daemon.

**Syntax:**

```
STARTD_DGRAM_PORT = port number
```

**Default value:** The default is 9615.

For more information related to using this keyword, see “Defining network characteristics” on page 30.

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

**Syntax:**

```
STARTD_RUNS_HERE = true | false
```

**Default value: true**

**Note:** This keyword is deprecated in the configuration file and has been moved to the administration file.

**STARTD_STREAM_PORT**
Specifies the port number used when connecting to the daemon.

**Syntax:**

```
STARTD_STREAM_PORT = port number
```

**Default value:** The default is 9611.

For more information related to using this keyword, see “Defining network characteristics” on page 30.

**STARTER**
Location of the starter executable (**LoadL_starter**).

**Syntax:**

```
STARTER = directory
```

**Default value:** `$(BIN)/LoadL_starter`

For more information related to using this keyword, see “How LoadLeveler daemons process jobs” on page 6.

**STARTER_COREDUMP_DIR**
Local directory for storing **LoadL_starter** coredump files.

**Syntax:**

```
STARTER_COREDUMP_DIR = directory
```

**Default value:** The **/tmp** directory.

For more information related to using this keyword, see "Specifying file and directory locations" on page 30.

**SUSPEND**
Determines whether running jobs should be suspended.

**Syntax:**

```
SUSPEND: expression that evaluates to T or F (true or false)
```

When **T**, LoadLeveler temporarily suspends jobs currently running on the machine. Suspended LoadLeveler jobs will either be continued or vacated. This keyword is not supported for parallel jobs.

**Default value:** No default value is set.

For information about time-related variables that you may use for this keyword, see "Variables to use for setting times" on page 127.

**SUSPEND_CONTROL**
Specifies the action to take when LoadLeveler fails to suspend an idle machine.

```
SUSPEND_CONTROL = noact | reset | shutdown
```

The available actions are:

**shutdown**
Shuts the failed machine down.

**reset**
Reboots the failed machine.

**noact**
No action is taken. Keep the failed machine in the current state.

**Default value: noact**

**TRACE**
Specifies the debug control flags that are used to control the trace function of LoadLeveler.

**Syntax:**

```
TRACE = flags
```

The debug control flags that can be used are:

**D_JOB_LIFECYCLE**
Enables job lifecycle tracing if this flag was specified. If this debug flag is not used, job lifecycle information will not be logged (even if you request job tracing).

**D_DISPATCHING_CYCLE**
Turns on dispatching cycle tracing.

**Default value:** No default value is set.

**TRUNC_KBDD_LOG_ON_OPEN**
When **true**, specifies the log file is restarted with every invocation of the daemon.

**Syntax:**

```
TRUNC_KBDD_LOG_ON_OPEN = true | false
```

**Default value: false**

For more information related to using this keyword, see "Configuring recording activity and log files" on page 32.

**TRUNC_MASTER_LOG_ON_OPEN**
When **true**, specifies the log file is restarted with every invocation of the daemon.

**Syntax:**

```
TRUNC_MASTER_LOG_ON_OPEN = true | false
```

**Default value: false**

For more information related to using this keyword, see "Configuring recording activity and log files" on page 32.

**TRUNC_REGION_MGR_LOG_ON_OPEN**
When **true**, specifies the log file is restarted with every invocation of the daemon. This keyword is used by the resource manager component only.

**Syntax:**

```
TRUNC_REGION_MGR_LOG_ON_OPEN = true | false
```

**Default value: false**

For more information related to using this keyword, see "Configuring recording activity and log files" on page 32.

**TRUNC_RESOURCE_MGR_LOG_ON_OPEN**
When **true**, specifies the log file is restarted with every invocation of the daemon. This keyword is used by the resource manager component only.

**Syntax:**

```
TRUNC_RESOURCE_MGR_LOG_ON_OPEN = true | false
```

**Default value: false**

**TRUNC_SCHEDD_LOG_ON_OPEN**
When **true**, specifies the log file is restarted with every invocation of the daemon.

**Syntax:**

```
TRUNC_SCHEDD_LOG_ON_OPEN = true | false
```

**Default value: false**

For more information related to using this keyword, see "Configuring recording activity and log files" on page 32.

**TRUNC_STARTD_LOG_ON_OPEN**
When **true**, specifies the log file is restarted with every invocation of the daemon.

**Syntax:**

```
TRUNC_STARTD_LOG_ON_OPEN = true | false
```

**Default value: false**

For more information related to using this keyword, see "Configuring recording activity and log files" on page 32.

**TRUNC_STARTER_LOG_ON_OPEN**
When **true**, specifies the log file is restarted with every invocation of the daemon.

**Syntax:**

```
TRUNC_STARTER_LOG_ON_OPEN = true | false
```

**Default value:** false

For more information related to using this keyword, see "Configuring recording activity and log files" on page 32.

**UPDATE_ON_POLL_INTERVAL_ONLY**

Specifies whether or not the LoadLeveler **startd** daemons will send machine update transactions to the central manager. Normally the LoadLeveler startd daemons running on executing nodes will send transactions to the central manager to provide updates of machine information at various times. An update is sent every polling interval. The polling interval is calculated by multiplying the values for the two keywords, **POLLING_FREQUENCY** and **POLLS_PER_UPDATE**, specified in the LoadLeveler configuration file.

In addition, updates are sent at other times such as when new jobs are started and when jobs terminate on the executing node. If you have a large and highly active cluster (the workload consists of a large number of short running jobs), the normal method for updating the central manager can add excessive network traffic. **UPDATE_ON_POLL_INTERVAL_ONLY** can help reduce this source of network traffic.

When **true** is specified, the LoadLeveler startd daemon will only send machine updates to the central manager at every polling interval and not at other times.

**Syntax:**

```
UPDATE_ON_POLL_INTERVAL_ONLY = false | true
```

**Default value: false**

**VACATE**

Determines whether suspended jobs should be vacated.

**Syntax:**

```
VACATE: expression that evaluates to T or F (true or false)
```

When **T**, suspended LoadLeveler jobs are removed from the machine and placed back into the queue (provided you specify **restart=yes** in the job command file). If a checkpoint was taken, the job restarts from the checkpoint. Otherwise, the job restarts from the beginning.

**Default value:** No default value is set.

For information about time-related variables that you may use for this keyword, see "Variables to use for setting times" on page 127.

**X_RUNS_HERE**

Specifies whether the kbdd (keyboard) daemon runs on the host. If you want to run the kbdd daemon, specify **true**.

**Syntax:**

```
X_RUNS_HERE = true | false
```

**Default value: false**

## User-defined keywords

This type of variable, which is generally created and defined by the user, can be named using any combination of letters and numbers.

A user-defined variable is set equal to values, where the *value* defines conditions, names files, or sets numeric values. For example, you can create a variable named **MY_MACHINE** and set it equal to the name of your machine named *iron* as follows:

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

You can then identify the keyword using a dollar sign ($) and parenthesis. For example, the literal **$(MY_MACHINE)** following the definition in the previous example results in the automatic substitution of **iron.ore.met.com** in place of **$(MY_MACHINE)**.

User-defined definitions may contain references, enclosed in parenthesis, to previously defined keywords. Therefore:

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

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

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

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

User-defined keywords can be specified in certain LoadLeveler configuration statements. Typically, the statements that can specify user-defined keywords are LoadLeveler expressions and statements that define file paths. The following list contains the LoadLeveler statements that can contain user-defined keywords:

| | | |
|---|---|---|
| **afs_getnewtoken** | **arch** | **bin** |
| **ckpt_execute_dir** | **comm** | **continue** |
| **ext_energy_policy_program** | **global_history** | **history** |
| **job_epilog** | **job_prolog** | **job_user_epilog** |
| **job_user_prolog** | **kbdd** | **kbdd_coredump_dir** |
| **ll_rsh_command** | **log** | **master** |
| **master_coredump_dir** | **process_tracking_extension** | **region_mgr** |
| **region_mgr_coredump_dir** | **releasedir** | **resource_mgr** |
| **resource_mgr_coredump_dir** | **schedd** | **schedd_coredump_dir** |
| **spool** | **start** | **startd** |
| **startd_coredump_dir** | **starter** | **starter_coredump_dir** |
| **starter_log** | | |

The sample configuration file shipped with the product defines and uses the following "user-defined" variables.

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

**CPU_Busy**
Defines the variable **CPU_Busy** and reassigns to it at each evaluation the

Boolean value True or False, depending on whether the Berkeley one-minute load average is equal to or greater than the saturation level of 1.5.

`CPU_Idle`
Defines the variable **CPU_Idle** and reassigns to it at each evaluation the Boolean value True or False, depending on whether the Berkeley one-minute load average is equal or less than 0.7.

`HighLoad`
Is a keyword that the user can define to use as a saturation level at which no further jobs should be started.

`HOUR`
Defines the variable **HOUR** and assigns to it a constant integer value.

`JobLoad`
Defines the variable **JobLoad** which defines the load on the machine caused by running the job.

`KeyboardBusy`
Defines the variable **KeyboardBusy** and reassigns to it at each evaluation the Boolean value True or False, depending on whether the keyboard and mouse have been idle for fifteen minutes.

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

`mail`
Specifies a local program you want to use in place of the LoadLeveler default mail notification method.

`MINUTE`
Defines the variable **MINUTE** and assigns to it a constant integer value.

`StateTimer`
Defines the variable **StateTimer** and reassigns to it at each evaluation the number of seconds since the current state was entered.

## LoadLeveler variables

LoadLeveler provides variables that you can use in your configuration file statements. LoadLeveler variables are evaluated by the LoadLeveler daemons at various stages. They do not require you to use any special characters (such as a parenthesis or a dollar sign) to identify them.

`Arch`
Indicates the system architecture.

`ClassSysprio`
The priority for the class of the job step, defined in the class stanza in the administration file.

**Default:** 0

`Connectivity`
The ratio of the number of active switch adapters on a node to the total number of switch adapters on the node. The value ranges from 0.0 (all switch adapters are down) to 1.0 (all switch adapters are active). A node with no switch adapters has a connectivity of 0.0. Connectivity can be used in a

**MACHPRIO** expression to favor nodes that do not have any down switch adapters or in a job's **REQUIREMENTS** to require only nodes with a certain connectivity.

For additional information about using this variable, see the **MACHPRIO** keyword description.

**ConsumableCpus**
The number of **ConsumableCpus** currently available on the machine.

**ConsumableLargePageMemory**
The amount of **ConsumableLargePageMemory**, in megabytes, currently available on the machine.

**ConsumableMemory**
The amount of **ConsumableMemory**, in megabytes, currently available on the machine.

**ConsumableVirtualMemory**
The amount of **ConsumableVirtualMemory**, in megabytes, currently available on the machine.

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

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

**Disk**
The free disk space in kilobytes on the file system where the executables for the LoadLeveler jobs assigned to this machine are stored. This refers to the file system that is defined by the execute keyword.

For additional information about using this variable, see the **MACHPRIO** keyword description.

**domain or domainname**
Dynamically indicates the official name of the domain of the current host machine where the program is running. Whenever a machine name can be specified or one is assumed, a domain name is assigned if none is present.

**EnteredCurrentState**
The value of **CurrentTime** when the current state (START, SUSPEND, and so on) was entered.

**FreeRealMemory**
The amount of free real memory, in megabytes, on the machine. This value should track very closely with the "fre" value of the **vmstat** command and the "free" value of the **svmon -G** command (units are 4K blocks).

For additional information about using this variable, see the **MACHPRIO** keyword description.

**GroupQueuedJobs**
The number of job steps associated with a LoadLeveler group which are either running or queued. (That is, job steps which are in one of these states: Checkpointing, Preempted, Preempt Pending, Resume Pending, Running, Starting, Pending, or Idle.)

**GroupRunningJobs**
The number of job steps for the LoadLeveler group which are in one of these states: Checkpointing, Preempted, Preempt Pending, Resume Pending, Running, Starting, or Pending.

**GroupSysprio**
The priority for the group of the job step, defined in the group stanza in the administration file.

**Default:** 0

**GroupTotalJobs**
The total number of job steps associated with this LoadLeveler group.

**host or hostname**
Dynamically indicates the standard host name as returned by gethostname() for the machine where the program is running. **host** and **hostname** are equivalent, and contain the name of the machine without the domain name appended to it. If administrators need to specify the domain name in the configuration file, they may use **domain** or **domainname** along with **host** or **hostname**. For example:

```
$(host).$(domain)
```

**JobIsBlueGene**
Indicates whether the job whose priority is being calculated is a Blue Gene job.

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

**LoadAvg**
The Berkely one-minute load average, a measure of the CPU load on the system. The load average is the average of the number of processes ready to run or waiting for disk I/O to complete. The load average does not map to CPU time.

For additional information about using this variable, see the **MACHPRIO** keyword description.

**MasterMachPriority**
A value that is equal to 1 for nodes which are master nodes (those with **master_node_exclusive = true**); this value is equal to 0 for nodes which are not master nodes. Assigning a high priority to master nodes may help job scheduling performance for parallel jobs which require master node features.

For additional information about using this variable, see the **MACHPRIO** keyword description.

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

For additional information about using this variable, see the **MACHPRIO** keyword description.

**OpSys**
Indicates the operating system on the host where the program is running. This value is automatically determined and should not be defined in the configuration file.

**PagesFreed**
The number of pages freed per second by the page replacement algorithm of the virtual memory manager.

For additional information about using this variable, see the **MACHPRIO** keyword description.

`PagesScanned`
: The number of pages scanned per second by the page replacement algorithm of the virtual memory manager.

  For additional information about using this variable, see the **MACHPRIO** keyword description.

`QDate`
: The difference in seconds between the UNIX date when the job step enters the queue and the UNIX date when the negotiator daemon starts up.

`Speed`
: The relative speed of the machine, defined in a machine stanza in the administration file.

  **Default**: 1

  For additional information about using this variable, see the **MACHPRIO** keyword description.

`State`
: The state of the startd daemon.

`tilde`
: The home directory for the LoadLeveler user ID.

`UserHoldTime`
: The total time that a job is in User Hold state.

`UserPrio`
: The user-defined priority of the job step, specified in the job command file with the **user_priority** keyword. The priority ranges from 0 to 100, with higher numbers corresponding to greater priority.

  **Default:** 50

`UserQueuedJobs`
: The number of job steps either running or queued for the user. (That is, job steps that are in one of these states: Checkpointing, Preempted, Preempt Pending, Resume Pending, Running, Starting, Pending, or Idle.)

`UserRunningJobs`
: The number of job step steps for the user which are in one of these states: Checkpointing, Preempted, Preempt Pending, Resume Pending, Running, Starting, or Pending.

`UserSysprio`
: The priority of the user who submitted the job step, defined in the user stanza in the administration file.

  **Default:** 0

`UserTotalJobs`
: The total number of job steps associated with this user.

`VirtualMemory`
: The size of available swap space (free paging space) on the machine, in kilobytes, reported by the startd daemon.

  For additional information about using this variable, see the **MACHPRIO** keyword description.

## Variables to use for setting dates

You can use the following date variables:

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

**`tm_mon`**
Number of months since January (0-11).

**`tm_wday`**
Number of days since Sunday (0-6).

**`tm_yday`**
Number of days since January 1 (0-365).

**`tm_year`**
The number of years since 1900 (0-9999). For example:

```
tm_year == 100
```

Denotes the year 2000.

**`tm4_year`**
The integer representation of the current year. For example:

```
tm4_year == 2010
```

Denotes the year 2010.

## Variables to use for setting times

You can use the following time variables in the START, SUSPEND, CONTINUE, VACATE, and KILL expressions.

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

**`tm_hour`**
The number of hours since midnight (0-23).

**`tm_isdst`**
Daylight Savings Time flag: positive when in effect, zero when not in effect, negative when information is unavailable. For example, to start jobs between 5 PM and 8 AM during the month of October, factoring in an adjustment for Daylight Savings Time, you can issue:

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

**`tm_min`**
Number of minutes after the hour (0-59).

**`tm_sec`**
Number of seconds after the minute (0-59).

# Chapter 7. Administration keyword reference

For a file-based configuration, the administration file lists and defines the machines in the LoadLeveler cluster, as well as the characteristics of classes, users, groups, and clusters.

LoadLeveler does not prevent you from having multiple copies of administration files, but having only one administration file prevents confusion and avoids potential problems that might arise from having multiple files to update. To use only one administration file that is available to all machines in a cluster, you must place the file in a shared file system.

For the database configuration option, the definitions for the machines in the LoadLeveler cluster, as well as the characteristics of classes, users, groups, and clusters are kept in database tables.

Table 22 lists the administration file subtasks:

*Table 22. Administration file subtasks*

| Subtask | Associated information (see . . . ) |
|---|---|
| To find out what administrator tasks you can accomplish by using the administration file | Chapter 3, "Configuring the LoadLeveler resource manager environment," on page 23 |
| To learn how to correctly specify the contents of an administration file | • "Administration file structure and syntax"<br>• "Administration keyword descriptions" on page 134 |

## Administration file structure and syntax

The administration file is called **LoadL_admin** and it lists and defines the *machine*, *machine_group*, *user*, *class*, *group*, *cluster* and *region*. For the database configuration option, there are one or more tables that correspond to each of these stanzas.

**Machine and machine_group stanzas**
: Defines the roles that the machines in the LoadLeveler cluster play. See "Defining machines" on page 61 for more information.

**User stanza**
: Defines LoadLeveler users and their characteristics. See "Defining users" on page 70 for more information.

**Class stanza**
: Defines the characteristics of the job classes. To define characteristics that apply to specific users, user substanzas can be added within a class stanza. See "Defining classes" on page 66 for more information.

**Group stanza**
: Defines the characteristics of a collection of users that form a LoadLeveler group. See "Defining groups" on page 71 for more information.

**Region stanza**
: Defines the characteristics of a region in a LoadLeveler cluster. See "Defining regions" on page 72 for more information.

Stanzas have the following general format:

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

Substanzas have the following general format:

```
label: {
   type = type_of_stanza
   keyword1 = value1
   keyword2 = value2
   ...
   substanza_label: {
      type = type_of_substanza
      keyword3 = value3
   }
}
```

Keywords are *not* case sensitive. This means you can enter them in lower case, upper case, or mixed case.

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

```
machine_a: type = machine
      central_manager = true       # defines this machine as the central manager

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

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

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

adapter_a: type = adapter
      adapter_name = en0  #defines an adapter
```

The following is a simple example of an administration file illustrating a class stanza that contains user substanzas:

```
default:
     type = machine
     central_manager = false
     schedd_host = true

default:
     type = class
     wall_clocK-limit = 60:00, 30:00

parallel: {
     type = class

     # Allow at most 50 running jobs for class parallel
     maxjobs = 50

     # Allow at most 10 running jobs for any single
     # user of class parallel
     default: {
          type = user
          maxjobs = 10
     }

     # Allow user dept_head to run as many as 20 jobs
     # of class parallel
     dept_head: {
          type = user
          maxjobs = 20
     }
}

dept_head:
     type = user
     maxjobs = 30
```

## Stanza characteristics

There are a number of characteristics associated with stanzas. The characteristics of a stanza are:

- Every stanza has a label associated with it. The label specifies the name you give to the stanza.
- Every stanza has a **type** field that specifies it as a machine, machine_group, user, class, group, cluster, or region stanza.
- New line characters are ignored. This means that separate parts of a stanza can be included on the same line. However, it is not recommended to have parts of a stanza cross line boundaries.
- White space is ignored, other than to delimit keyword identifiers. This eliminates confusion between tabs and spaces at the beginning of lines.
- A crosshatch sign (#) identifies a comment and can appear anywhere on the line. All characters following this sign on that line are ignored.
- Multiple stanzas of the same label are allowed, but only the first label is used.
- Default stanzas specify the default values for any keywords which are not otherwise specified. Each stanza type can have an associated default stanza. A default stanza must appear in the administration file ahead of any specific stanza entries of the same type. For example, a default class stanza must appear ahead of any specific class stanzas you enter.
- Stanzas can be nested within other stanzas (these are known as *substanzas*).
- The use of opening and closing braces ({ and }) to mark the beginning and end of a stanza is optional for stanzas that *do not* contain substanzas. A stanza that contains substanzas *must* be specified using braces as delimiting characters. Only user substanzas within class stanzas are supported. No types of stanzas other than class support substanzas and no types of stanzas other than user can be provided as substanzas within a class.
- If a syntax error is encountered, the remainder of the stanza is ignored and processing resumes with the next stanza.

## Syntax for limit keywords

The syntax for setting a limit is:

```
limit_type = hardlimit,softlimit
```

For example:

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

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

```
core_limit = 120kb
```

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

```
core_limit = ,100kb
```

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

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

*integer[.fraction][units]*

The allowable units for these limits are:

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

If no units are specified for data limits, then bytes are assumed.

For limit keywords that refer to a number limit — such as **nproc_limit**, **locks_limit**, and **nofile_limit** — the hard limit and the soft limit are expressed as:

*integer[.fraction][units]*

The allowable units for these limits are:

| | | |
|---|---|---|
| K | Kilo | (2**10) |
| M | Mega | (2**20) |
| G | Giga | (2**30) |
| T | Tera | (2**40) |
| P | Peta | (2**50) |
| E | Exa | (2**60) |

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

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

Fractions are rounded to seconds.

You can use the following character strings with all limit keywords except the **copy** keyword for **wall_clock_limit**, **job_cpu_limit**, and **ckpt_time_limit**:

**rlim_infinity**
: Represents the largest positive number.

**unlimited**
: Has same effect as **rlim_infinity**.

**copy**
: Uses the limit currently active when the job is submitted.

## 64-bit support for administration file keywords

Administrators can assign 64-bit integer values to selected keywords in the administration file. System resource limits, with the exception of CPU limits, are treated by LoadLeveler daemons and commands as 64-bit limits.

Table 23 describes 64-bit support for specific administration file keywords.

*Table 23. Notes on 64-bit support for administration file keywords*

| Keyword | Stanza | Notes |
|---|---|---|
| **as_limit** | Class | See the notes for **core_limit** and **data_limit**. |
| **core_limit**<br>**data_limit** | Class | 64-bit integer values can be assigned to these limits. Fractional specifications are allowed and will be converted to 64-bit integer values. Unit specifications are accepted and can be one of the following: b, w, kb, kw, mb, mw, gb, gw, tb, tw, pb, pw, eb, ew.<br><br>**Example:**<br>`core_limit = 8gb,4.25gb` |
| **default_resources**<br>**default_node_resources** | Class | Consumable resources associated with the **default_resources** and **default_node_resources** keywords can be assigned 64-bit integer values. Fractional specifications are not allowed. Unit specifications are valid only when specifying the values of the predefined **ConsumableMemory**, **ConsumableVirtualMemory**, and **ConsumableLargePageMemory** resources.<br><br>**Example:**<br>`default_resources = ConsumableVirtualMemory(12 gb)db2_license(112)` |
| **file_limit** | Class | See the notes for **core_limit** and **data_limit**. |
| **locks_limit** | Class | See the notes for **nproc_limit**. |
| **memlock_limit** | Class | See the notes for **core_limit** and **data_limit**. |
| **nofile_limit** | Class | See the notes for **nproc_limit**. |
| **nproc_limit** | Class | 64-bit integer values can be assigned to these limits. Fractional specifications are allowed and will be converted to 64-bit integer values. Unit specifications is number.<br><br>Unit specification is a number that can be used in conjunction with the following abbreviations:<br>**K** kilo<br>**M** mega<br>**G** giga<br>**T** tera<br>**P** peta<br>**E** exa<br><br>**Examples:**<br>`nproc_limit = 1000,285` |
| **resources** | Machine | Consumable resources associated with the **resources** keyword can be assigned 64-bit integer values. Fractional specifications are not allowed. Unit specifications are valid only when specifying the values of the predefined **ConsumableMemory**, **ConsumableVirtualMemory**, and **ConsumableLargePageMemory** resources.<br><br>**Examples:**<br>`resources = spice2g6(9123456789012) ConsumableMemory(10 gw)`<br>`resources = ConsumableVirtualMemory(15 pb) db2_license(1234567890)` |
| **rss_limit**<br>**stack_limit** | Class | See the notes for **core_limit** and **data_limit**.<br><br>**Example:**<br>`rss_limit  = 1.25eb,3.33pw` |

### 64-bit limits on Linux systems

Applications managed by LoadLeveler for AIX can be 64-bit applications if the hardware architecture on which AIX is running is capable of supporting 64-bit processes.

Resource limits, such as data limits and stack limits, can be 64-bit limits. When a value of *unlimited* is specified for a process limit (**cpu_limit** excepted) in the LoadLeveler administration file or job command file, the AIX version of LoadLeveler stores this value internally as `INT64_MAX`. Before starting the user job, **LoadL_starter** sets the appropriate limit to this value. This behavior is correct because, on AIX, `RLIM64_INFINITY` is the same as `INT64_MAX (= 0x7FFFFFFFFFFFFFFFLL)`.

On Linux systems, `RLIM64_INFINITY` is equal to `UINT64_MAX (= 0xFFFFFFFFFFFFFFFFULL)`. To maintain compatibility with AIX, LoadLeveler for Linux also stores *unlimited* internally as `INT64_MAX`. However, **LoadL_starter** on Linux sets all process limits (**cpu_limit** excepted) that are in the range (`INT64_MAX`, `UINT64_MAX`) to `UINT64_MAX` before starting the jobs managed by LoadLeveler.

For historical reasons, LoadLeveler for AIX treats the hard and soft time limits, such as **cpu_limit**, **job_cpu_limit**, and **wall_clock_limit**, as 32-bit limits and *unlimited* means `INT32_MAX`. For consistency reasons, LoadLeveler for Linux assumes the same behavior.

## Administration keyword descriptions

This topic contains an alphabetical list of the LoadLeveler administration keywords.

**admin**
: Specifies a list of administrators for a group or class.

**Syntax:**

```
admin = list
```

Where *list* is a blank-delimited list of administrators for either this class or this group, depending on whether this keyword appears in a class or group stanza, respectively. These administrators can hold, release, and cancel jobs in this class or this group.

**adapter_list**
: Specifies a list of adapters to be used on the machine.

**Syntax:**

```
adapter_list = list1 list2 ...
```

or

```
adapter_list = none
```

Where the *list* of adapters consists of interface names in the order that will be used for scheduling. If the **adapter_list** keyword is not specified, all adapters that were found dynamically will be used for scheduling. If **adapter_list** is set to **none**, no adapters will be used by this machine. The **adapter_list** cannot be set to a blank value.

**as_limit**
: Specifies the hard limit, soft limit, or both for the address space to be used by each process of the submitted job.

**Syntax:**

```
as_limit = hardlimit,softlimit
```

**Examples:**

```
as_limit = 125621         # hardlimit = 125621 bytes
as_limit = 5621kb         # hardlimit = 5621 kilobytes
as_limit = 2mb            # hardlimit = 2 megabytes
as_limit = 2.5mw          # hardlimit = 2.5 megawords
as_limit = unlimited      # hardlimit = 9,223,372,036,854,775,807 \
                            bytes (X'7FFFFFFFFFFFFFFF')
as_limit = rlim_infinity  # hardlimit = 9,223,372,036,854,775,807 \
                            bytes (X'7FFFFFFFFFFFFFFF')
```

For additional information about limit keywords, see the following topics:
- "Syntax for limit keywords" on page 131
- "Using limit keywords" on page 66

**ckpt_dir**

Specifies the directory to be used for checkpoint files for jobs that did not specify this directory in the job command file.

The actual directory used to store the checkpoint files is a combination of the value of this keyword and the value of the **ckpt_subdir** keyword.

**Syntax:**

```
ckpt_dir = directory
```

Where *directory* is the directory location to be used for checkpoint files that did not have a directory name specified in the job command file. If the value specified does not have a fully qualified directory path (including the beginning forward slash), the initial working directory will be inserted before the specified value.

The value specified by the **ckpt_dir** keyword is only used when the **ckpt_subdir** keyword does not contain a full path name in the job command file or is not specified in the job command file.

**Default:** Initial working directory

**ckpt_time_limit**

Specifies the hard limit, soft limit, or both limits for the elapsed time that checkpointing a job can take.

**Syntax:**

```
ckpt_time_limit = hardlimit,softlimit
```

Where *hardlimit,softlimit* defines the maximum time that checkpointing a job can take. When LoadLeveler detects that the softlimit has been exceeded, it attempts to end the checkpoint and allow the job to continue. If this is not possible, and the hard limit is exceeded, LoadLeveler will terminate the job. The start time of the checkpoint is defined as the time when the Startd daemon receives status from the starter that a checkpoint has started.

**Default:** Unlimited

**Examples:**

```
ckpt_time_limit = 30:45         #hardlimit - 30 minutes 45 seconds
ckpt_time_limit = 30:45,25:00   #hardlimit - 30 minutes 45 seconds
                                #soflimit  - 25 minutes
```

For additional information about limit keywords, see the following topics:
- "Syntax for limit keywords" on page 131
- "Using limit keywords" on page 66

`class`

Determines whether a machine will accept jobs of a certain job class. For parallel jobs, you must define a class instance for each task you want to run on a node using the format, **class** = *class_name (count)*. This format defines the **class** names using a statement that names the classes and sets the number of tasks for each class in parenthesis.

With this format, the following rules apply:
- Each class can have only one entry
- If a class has more than one entry or there is a syntax error, the entire **class** statement will be ignored
- If the **class** statement has a blank value or is not specified, it will be defaulted to **No_Class (1)**
- The number of instances for a class specified inside the parenthesis **( )** must be an unsigned integer. If the number specified is 0, it is correct syntactically, but the class will not be defined in LoadLeveler
- If the number of instances for all classes in the **class** statement are 0, the default **No_Class(1)** will be used

**Note:** The class names list is blank delimited.

You can have a maximum of 1024 characters in the class statement. You cannot use **allclasses** or **data_stage** as a class name, since these are reserved LoadLeveler keywords.

You can assign multiple classes to the same machine by specifying the classes in the LoadLeveler administration file (called **LoadL_admin**). The classes, themselves, should also be defined in the administration file. See the topic, "Setting up a single machine to have multiple job classes" in *LoadLeveler: Using and Administering* and “Defining classes” on page 66 for more information on classes.

**Syntax:**

```
class = class_name (count) ...
```

**Default value: No_Class(1)**

`class_comment`

Text characterizing the class.

**Syntax:**

```
class_comment = "string"
```

Where *string* is text characterizing the class. The comment string associated with this keyword cannot contain an equal sign (**=**) or a colon (**:**) character. The length of the string cannot exceed 1024 characters.

**Default:** No default value is set.

`collective_groups`

Requests the Collective Acceleration Unit (CAU) groups for the specified protocol instances of the job.

**Syntax:**

```
collective_groups = number
```

The value of the collective groups must be greater than or equal to zero. The value specified for the **collective_groups** keyword in the job command file overwrites any value specified for the **collective_groups** keyword in the administration file. If the job is not sharing the nodes with other jobs, then the protocol instance of the job will be allocated at least *number* CAU groups.

Additional CAUs can be allocated to the job step if additional CAU groups are available on the node and the node is not shared with other jobs. If the job is sharing the node with other jobs, then exactly *number* CAU groups are allocated to each protocol instance of the job.

**Default value:** The default value varies depending on whether the job shares the nodes with other jobs or not. If the job is not sharing the nodes with other jobs, then all the CAU groups are allocated to all the protocol instances of the job proportionally. If the job is sharing the nodes with other jobs, then zero CAU groups are allocated to the protocol instances of the job.

**core_limit**

Specifies the hard limit, soft limit, or both limits for the size of a core file a job can create.

**Syntax:**

```
core_limit = hardlimit,softlimit
```

**Examples:**

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

For additional information about limit keywords, see the following topics:
- "Syntax for limit keywords" on page 131
- "Using limit keywords" on page 66

**cpu_limit**

Specifies hard limit, soft limit, or both limits for the CPU time to be used by each individual process of a job step.

**Syntax:**

```
cpu_limit = hardlimit,softlimit
```

For example, if you impose a **cpu_limit** of five hours and you have a job step composed of five processes, each process can consume five CPU hours; the entire job step can therefore consume 25 total hours of CPU.

**Examples:**

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

For additional information about limit keywords, see the following topics:
- "Syntax for limit keywords" on page 131
- "Using limit keywords" on page 66

**cpu_speed_scale**

Determines whether CPU time is normalized according to machine speed.

**Syntax:**

```
cpu_speed_scale = true | false
```

Where **true** specifies that CPU time (which is used, for example, in setting limits and in accounting information), is in normalized units for each machine. **false** specifies that CPU time is in native units for each machine. For an example of using this keyword to normalize accounting information, see "Example: Setting up job accounting files" on page 46.

**Default: false**

**data_limit**

Specifies hard limit, soft limit, or both for the data segment to be used by each process of the submitted job.

**Syntax:**

```
data_limit = hardlimit,softlimit
```

**Examples:**

```
data_limit = 125621         # hardlimit = 125621 bytes
data_limit = 5621kb         # hardlimit = 5621 kilobytes
data_limit = 2mb            # hardlimit = 2 megabytes
data_limit = 2.5mw          # hardlimit = 2.5 megawords
data_limit = unlimited      # hardlimit = 9,223,372,036,854,775,807 bytes
                            # (X'7FFFFFFFFFFFFFFF')
data_limit = rlim_infinity  # hardlimit = 9,223,372,036,854,775,807 bytes
                            # (X'7FFFFFFFFFFFFFFF')
data_limit = copy           # copy data hardlimit value from
                            # submitting machine.
```

For additional information about limit keywords, see the following topics:
- "Syntax for limit keywords" on page 131
- "Using limit keywords" on page 66

**default_node_resources**

Specifies quantities of the consumable resources consumed by each node of a job step provided that a **node_resources** keyword is not coded for the step in the job command file. If a **node_resources** keyword is coded for a job step, it overrides any default node resources associated with the associated job class. The resources must be machine resources; floating resources cannot be assigned with the **node_resources** keyword.

**Syntax:**

```
default_node_resources = name(count) name(count) ... name(count)
```

The administrator defines the name and count for **default_node_resources**. In addition, *name(count)* could be **ConsumableCpus**(*count*), **ConsumableMemory**(*count units*), **ConsumableLargePageMemory***(count units)*, or **ConsumableVirtualMemory**(*count units*).

The **ConsumableMemory**, **ConsumableVirtualMemory**, and **ConsumableLargePageMemory** can be specified with both a *count* and *units*. **ConsumableMemory** or **ConsumableVirtualMemory** specified resource *count* must be an integer greater than zero. **ConsumableLargePageMemory** specified resource *count* must be an integer greater than or equal to zero. The allowable *units* are those normally used with LoadLeveler data limits:

```
b bytes
w words
kb kilobytes (2**10 bytes)
kw kilowords (2**12 bytes)
mb megabytes (2**20 bytes)
mw megawords (2**22 bytes)
gb gigabytes (2**30 bytes)
gw gigawords (2**32 bytes)
tb terabytes (2**40 bytes)
tw terawords (2**42 bytes)
```

```
pb petabytes (2**50 bytes)
pw petawords (2**52 bytes)
eb exabytes  (2**60 bytes)
ew exawords  (2**62 bytes)
```

The **ConsumableMemory**, **ConsumableVirtualMemory**, and **ConsumableLargePageMemory** values are stored in MB (megabytes) and are rounded up. If no units are specified, then megabytes are assumed. For **ConsumableMemory** or **ConsumableVirtualMemory** the smallest amount that you can request is 1 MB.

**default_resources**

Specifies the default amount of resources consumed by a task of a job step, provided that the **resources** or **dstg_resources** keyword is not coded for the step in the job command file. If a resources keyword is coded for a job step, then it overrides any **default resources** associated with the associated job class.

**Syntax:**

```
default_resources = name(count) name(count)...name(count)
```

The administrator defines the name and count values for **default_resources**. In addition, *name(count)* could be **ConsumableCpus***(count)*, **ConsumableMemory***(count units)*, **ConsumableVirtualMemory***(count units)*, or **ConsumableLargePageMemory***(count units)*.

The **ConsumableMemory**, **ConsumableVirtualMemory**, and **ConsumableLargePageMemory** can be specified with both a *count* and *units*. **ConsumableMemory** or **ConsumableVirtualMemory** specified resource *count* must be an integer greater than zero. **ConsumableLargePageMemory** specified resource *count* must be an integer greater than or equal to zero. The allowable *units* are those normally used with LoadLeveler data limits:

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

The **ConsumableMemory** and **ConsumableVirtualMemory** values are stored in MB (megabytes) and are rounded up. Therefore, the smallest amount of **ConsumableMemory** or **ConsumableVirtualMemory** that you can request is 1 MB. If no units are specified, then megabytes are assumed.

**dstg_max_starters**

Specifies a machine-specific limit on the number of data staging initiators. Since each task of a data staging job step consumes one initiator from the **data_stage** class on the specified machine, **dstg_max_starters** provides the maximum number of data staging tasks that can run at the same time on the machine.

**Syntax:**

```
dstg_max_starters = number
```

**Notes:**

1. If you have not set the **dstg_max_starters** value in either the global or local configuration files, there will not be any data staging initiators on the specified machine. In this configuration, the executing machine will not be allowed to perform data staging tasks.
2. The value specified for **dstg_max_starters** will be the number of initiators available for the built-in **data_stage** class on that machine.
3. The value specified for **dstg_max_starters** will not limit the value specified for **dstg_max_starters**.

**Default value: 0**

**endpoints**

Specifies the number of endpoints that can be used by each task per protocol instance.

**Syntax:**

```
endpoints = 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128
```

where:

*number*

Must be a power of 2 and no greater than 128 (that is, from {1, 2, 4, 8, 16, 32, 64, 128}). If the value specified is not a power of 2, the next higher power of 2 is used and a warning message is issued. The value of the **endpoints** keyword is inherited to all the protocol instances of the jobs.

**Default value:** One endpoints value is assigned for each task per protocol instance.

**exclude_classes**

**exclude_classes** can be specified within a cluster stanza.

Specifies a blank-delimited list of one or more job classes that will not accept remote jobs within the cluster.

**Syntax:**

```
exclude_classes = class_name[(cluster_name)] ...
```

Where *class_name* specifies a class to be excluded and *cluster_name* can be used to specify that remote jobs from *cluster_name* submitted under *class_name* will be excluded but any other jobs submitted under *class_name* from other clusters will be allowed.

Do not specify a list of **exclude_classes** and **include_classes**. Only one of these keywords can be used within any cluster stanza. **exclude_classes** takes precedence over **include_classes** if both are specified.

**Default:** The default is that no classes are excluded.

**feature**

Specifies an optional characteristic to use to match jobs with machines. You can specify unique characteristics for any machine using this keyword. When evaluating job submissions, LoadLeveler compares any required features specified in the job command file to those specified using this keyword. You can have a maximum of 1024 characters in the feature statement.

**Syntax:**

```
feature = string ...
```

**Default value:** No default value is set.

**Example:** If a machine has licenses for installed products ABC and XYZ in the local configuration file, you can enter the following:

```
feature = abc xyz
```

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

```
requirements = (feature == abc) && (feature == xyz)
```

**Example:** When submitting a job that requires the SMT function, first specify **smt = yes** in the job command file (or select a class which had **smt = yes** defined). Next, specify **job_type = parallel** and **node_usage = not_shared** and last, enter the following in the job command file:

```
requirements = (Feature == smt)
```

**file_limit**

Specifies the hard limit, soft limit, or both limits for the size of a file that a job can create.

**Syntax:**

file_limit = *hardlimit*,*softlimit*

For additional information about limit keywords, see the following topics:
- "Syntax for limit keywords" on page 131
- "Using limit keywords" on page 66

**imm_send_buffers**

Requests a number of immediate send buffers for each window allocated for each protocol instance of the job.

**Syntax:**

imm_send_buffers=*number*

The value of the immediate send buffers must be greater than or equal to zero. The value of the immediate send buffers is inherited from all the protocol instances of the job step unless the individual protocol instances are specified with their own immediate send buffers. If the job is sharing nodes with other jobs, then exactly *number* immediate send buffers are allocated to each window assigned to each protocol instance of the job. If the job is not sharing the nodes with other jobs then at least *number* immediate send buffers are allocated to each window assigned to each protocol instance of the job. Additional immediate send buffers can be distributed evenly to the windows assigned to the job if the nodes are not shared with other jobs.

**Default value:** The default value varies depending on whether the job shares the nodes with other jobs or not. If the job is not sharing the nodes with other jobs, then all the immediate send buffers are allocated to all the protocol instances of the job proportionally. If the job is sharing the nodes with other jobs, then one immediate send buffer is assigned for each window assigned to the job.

**include_classes**

**include_classes** can be specified within a cluster stanza.

Specifies a blank-delimited list of one or more job classes that will accept remote jobs within the cluster.

**Syntax:**

include_classes = *class_name[(cluster_name)] ...*

Where *class_name* specifies a class to be included and *cluster_name* can be used to specify that remote jobs from *cluster_name* will be included but any other jobs submitted under *class_name* from other clusters will not be allowed.

Do not specify a list of **exclude_classes** and **include_classes**. Only one of these can be used within any cluster stanza. **exclude_classes** takes precedence over **include_classes** if both are specified.

**Default:** The default is that all classes are included.

**job_cpu_limit**
Specifies the hard limit, soft limit, or both limits for the total amount of CPU time that all tasks of an individual job step can use per machine.

**Syntax:**

```
job_cpu_limit = hardlimit,softlimit
```

**Example:**

```
job_cpu_limit = 10000
```

For more information on this keyword, see:
- JOB_LIMIT_POLICY keyword
- For additional information about limit keywords, see the following topics:
  - "Syntax for limit keywords" on page 131
  - "Using limit keywords" on page 66

**locks_limit**
Specifies the hard limit, soft limit, or both for the file locks to be used by each process of the submitted job.

**Syntax:**

```
locks_limit = hardlimit,softlimit
```

**Examples:**

```
locks_limit = 125621          # hardlimit = 125621
```

For additional information about limit keywords, see the following topics:
- "Syntax for limit keywords" on page 131
- "Using limit keywords" on page 66

**machine_list**
Specifies a list of machines that belong to the **machine_group**.

**Syntax:**

The syntax of machine_list, which consists of a list of expressions separated by comma, is modeled after the xCAT **noderange** concept. machine_list will support a subset of the xCAT syntaxes. The supported usages are:
- A single machine
  - x330n01, pccluster21, c197blade1b04
- '-' or ':' to represent a range of machines
  - Only appended numbers can be expanded, and paddings can be properly added
  - Only one range of numbers can be specified within each set of enclosing brackets
  - x330n01-x330n04 (x330n01, x330n02, x330n03, x330n04)
  - x330n01-x330n123 (x330n01, x330n02, ...x330n10... x330n99, x330n100, ... x330n123)
  - pccluster[1-1000] (pccluster1, pccluster2...pccluter1000)

  - c197blade1b[01:99] (c197blade1b01, c197blade1b02...c197blade1b99)
  - c250f08c[06-12]ap[01-08] (c250f08c06ap01, c250f08c06ap02, ..., c250f08c07ap01, c250f08c07ap02, ... c250f08c12ap08)
- '+' to represent an incremented range of machines
  - Only appended numbers can be expanded, and paddings can be properly added
  - x330n01+3 (x330n01, x330n02, x330n03, x330n04)
  - pccluster1+999 (pccluster1, pccluster2...pccluter1000)
  - c197blade1b01+98 (c197blade1b01, c197blade1b02...c197blade1b99)
- '-' serving as a prefixed exclusion operator to represent the range of machines to be excluded from the result
  - '-' as exclusion operator has precedence over other expressions
  - -x330n02, x330n01+3
  - pccluster1-1000, -pccluster21-34

The following usages are supported by xCAT, but are not supported in LoadLeveler:
- '@' because there is only one machine_list in a stanza
- '^' as file operator
- Multiple expansion
- Regular expression is not supported
- Suffix
- “node” as the default prefix

**machine_mode**

Specifies the type of jobs this machine can run.

**Syntax:**

```
machine_mode = batch | interactive | general
```

Where:

**batch** Specifies this machine can run only batch jobs.

**interactive**
Specifies this machine can run only interactive jobs. Only POE is currently enabled to run interactively.

**general**
Specifies this machine can run both batch jobs and interactive jobs.

**Default: general**

**max_starters**

Specifies the maximum number of tasks that can run simultaneously on a machine. In this case, a task can be a serial job step or a parallel task. **max_starters** defines the number of initiators on the machine (the number of tasks that can be initiated from a **startd**).

**Syntax:**

```
max_starters = number
```

**Default value:** If this keyword is not specified, the default is the number of elements in the **class** statement.

**memlock_limit**

Specifies the hard limit, soft limit, or both for the memory that can be locked by each process of the submitted job.

**Syntax:**

```
memlock_limit = hardlimit,softlimit
```

**Examples:**

```
memlock_limit = 125621         # hardlimit = 125621 bytes
memlock_limit = 5621kb         # hardlimit = 5621 kilobytes
memlock_limit = 2mb            # hardlimit = 2 megabytes
memlock_limit = 2.5mw          # hardlimit = 2.5 megawords
memlock_limit = unlimited      # hardlimit = 9,223,372,036,854,775,807 \
                                 bytes (X'7FFFFFFFFFFFFFFF')
memlock_limit = rlim_infinity  # hardlimit = 9,223,372,036,854,775,807 \
                                 bytes (X'7FFFFFFFFFFFFFFF')
```

For additional information about limit keywords, see the following topics:
- "Syntax for limit keywords" on page 131
- "Using limit keywords" on page 66

**name_server**

Specifies a list of name servers used for a machine.

**Syntax:**

```
name_server = list
```

Where *list* is a blank-delimited list of character strings that is used to specify which nameservers are used for the machine. Valid strings are DNS, NIS, and LOCAL. LoadLeveler uses the list to determine when to append a DNS domain name for machine names specified in LoadLeveler commands.

If DNS is specified alone, LoadLeveler will always append the DNS domain name to machine names specified in LoadLeveler commands. If NIS or LOCAL is specified, LoadLeveler will never append a DNS domain name to machine names specified in LoadLeveler commands.

**Note:** The **name_server** keyword is a cluster-wide keyword that can only be specified for the default machine or default machine group stanza.

**nice**

Increments the *nice* value of a job.

**Syntax:**

```
nice = value
```

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

The default value is 0. If the administrator has decided to enforce consumable resources, the *nice* value will only adjust priorities of processes within the same WLM class. Because LoadLeveler defines a single class for every job step, the *nice* value has no effect.

For more information, consult the appropriate UNIX documentation.

**nofile_limit**

Specifies the hard limit, soft limit, or both for the number of open file descriptors that can be used by each process of the submitted job.

**Syntax:**

```
nofile_limit = hardlimit,softlimit
```

**Examples:**

```
nofile_limit = 1000            # hardlimit = 1000
```

For additional information about limit keywords, see the following topics:
- "Syntax for limit keywords" on page 131
- "Using limit keywords" on page 66

**nproc_limit**

Specifies the hard limit, soft limit, or both for the number of processes that can be created for the real user ID of the submitted job.

**Syntax:**

```
nproc_limit = hardlimit,softlimit
```

**Examples:**

```
nproc_limit = 256              # hardlimit = 256
```

For additional information about limit keywords, see the following topics:
- "Syntax for limit keywords" on page 131
- "Using limit keywords" on page 66

**pool_list**

Specifies a list of pool numbers to which the machine belongs. Do not use negative numbers in a machine pool_list.

**Syntax:**

```
pool_list = pool_numbers
```

Where *pool_numbers* is a blank-delimited list of non-negative numbers identifying pools to which the machine belongs. These numbers can be any positive integers including zero.

**power_management_policy**

**Syntax:**

```
power_management_policy = start_time;duration | off
```

Where:

**off**
Disables the site energy policy.

*start_time*
The format of *start_time* is the same format used by crontab.

*duration*
Is the number of minutes the machine will be in standby state. Set the duration and the time between durations to a value more than 10 minutes.

**Example 1:**

In this example, the machine will go in standby state from 0 o'clock every day for 5 hours:

```
power_management_policy=00 00 * * *; 300
```

The machine will stay in standby state at least 5 hours if there is no job assigned to it.

**Example 2:**

In this example, the node will be in standby state when there is no job running. The node will wake up when a job is dispatched to run on it, or when the **llrchgmstat** command is issued to wake it up:

```
power_management_policy=* * * * *
```

**Example 3:**

In this example, node c197blade4b06 will use the site policy that is configured in machine group mgroup1 and node c197blade4b05 will use the site policy that is set in the machine stanza:

```
c197blade4b05.ppd.pok.ibm.com: {
        type = machine
        central_manager = true
     schedd_host = true
     class = No_Class(128)
     power_management_policy = 00 03 1 3 *;1800
}
mgroup1: {
        type = machine_group
     schedd_runs_here = false
     startd_runs_here = true
     max_starters = 128
     class = No_Class(128)
     machine_mode = general
     machine_list =  c197blade4b06.ppd.pok.ibm.com
     power_management_policy = 00 03 1 5 *;1200
}
```

**prestarted_starters**

Specifies how many prestarted starter processes LoadLeveler will be maintained on an execution node to manage jobs when they arrive. The startd daemon starts the number of starter processes specified by this keyword.

**Syntax:**

```
prestarted_starters = number
```

*number* must be less than or equal to the value specified through the **max_starters** keyword. If the value of **prestarted_starters** specified is greater then **max_starters**, LoadLeveler records a warning message in the startd log and assigns **prestarted_starters** the same value as **max_starters**.

If the value **prestarted_starters** is zero, no starter processes will be started before jobs arrive on the execution node.

**Default value:** The default is 1.

**region**

Defines the region to which the machine or **machine_group** belongs. The machine cannot be defined as the primary region manager or alternate region manager of another region. Substanzas cannot use this keyword. All machines within a **machine_group** can belong to only one region.

**Syntax:**

```
region = region_name
```

**Default:** No default value is set.

**resources**

Specifies quantities of the consumable resources initially available on the machine.

**Syntax:**

```
resources = name(count) name(count) ... name(count)
```

Where *name(count)* is an administrator-defined name and count, or could also be **ConsumableCpus***(count)*, **ConsumableMemory***(count units)*, **ConsumableVirtualMemory***(count units)*, or **ConsumableLargePageMemory***(count units)*.

The **ConsumableMemory**, **ConsumableVirtualMemory**, and **ConsumableLargePageMemory** can be specified with both a count and units. The **ConsumableMemory** or **ConsumableVirtualMemory** specified resource count must be an integer greater than zero. The **ConsumableLargePageMemory** specified resource count must be an integer greater than or equal to zero. The allowable units are those normally used with LoadLeveler data limits:

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

The **ConsumableMemory**, **ConsumableVirtualMemory**, and **ConsumableLargePageMemory** values are stored in MB (megabytes) and rounded up. For **ConsumableMemory** or **ConsumableVirtualMemory**, the smallest amount that you can request is one megabyte. If no units are specified, then megabytes are assumed. Resources defined here that are not in the **SCHEDULE_BY_RESOURCES** list in the global configuration file will not effect the scheduling of the job.

For the **ConsumableCpus** resource, a value of **all** can be specified instead of count. This indicates that the CPU resource value will be obtained from the Startd daemons. However, these resources will not be available for scheduling until the first **Startd** update.

A list within < > angle brackets indicates a list of CPU IDs. Only CPUs with logical IDs specified in the list will be considered available for LoadLeveler jobs. The following example specifies a list of CPUs:

```
resources = ConsumableCpus< 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 >
```

CPU IDs can also be specified using a list of ranges:

```
resources = ConsumableCpus< 0-6 10-16 >
```

Jobs requesting processor affinity with the **task_affinity** keyword in the job command file will only run on machines where the resource statement contains the **ConsumableCpus** keyword.

The logical IDs of the CPUs available on a machine can be found issuing the `bindprocessor -q` command.

**Default:** No default value is set.

**restart**

Specifies whether LoadLeveler considers a job to be restartable.

**Syntax:**

```
restart = yes | no
```

**Default: yes**

**`rss_limit`**

Specifies the hard limit, soft limit, or both limits for the resident set size for a job.

**Syntax:**

```
rss_limit = hardlimit,softlimit
```

For additional information about limit keywords, see the following topics:
- "Syntax for limit keywords" on page 131
- "Using limit keywords" on page 66

**`schedd_fenced`**

Specifies whether or not the central manager is to ignore connections from the Schedd daemon running on this machine.

**Syntax:**

```
schedd_fenced = true | false
```

Where **true** specifies that the central manager ignores connections from the Schedd daemon running on this machine. Use the **true** setting together with the **llrmovespool** command when you want to attempt to recover resources lost when a node running the Schedd daemon fails. A **true** setting prevents conflicts when you use the **llrmovespool** command to move jobs from one Schedd machine to another. For more information, see the topic, "How do I recover resources allocated by a Schedd machine?" in *LoadLeveler: Using and Administering*.

**Default: false**

**`schedd_host`**

Specifies whether or not this machine is used to help submit-only machines access LoadLeveler hosts that run LoadLeveler jobs.

**Syntax:**

```
schedd_host = true | false
```

When **true** this keyword specifies that if a Schedd is running on a machine that it will serve as a public scheduling machine. A public scheduling machine accepts job submissions from other machines in the LoadLeveler cluster. Jobs are submitted to a public scheduling machine if:

- The submission occurs on a machine which does not run the Schedd daemon. These include submit-only machines and machines which are configured to run other LoadLeveler daemons but not the Schedd daemon.
- The submission occurs on a machine which runs the Schedd daemon but is configured to submit jobs to a public scheduling machine by having the **SCHEDD_SUBMIT_AFFINITY** keyword set to **false** in the global or local configuration file.

This keyword does not configure LoadLeveler to run the Schedd daemon on a node. Use the administration keyword **schedd_runs_here** to run the Schedd daemon on a node.

**Default: false**

**`schedd_runs_here`**

Specifies whether the Schedd daemon runs on the host. If you do not want to run the Schedd daemon, specify **false**.

This keyword does not designate a machine as a public scheduling machine. Unless configured as a public scheduling machine, a machine configured to run the Schedd daemon will only accept job submissions from the same machine running the Schedd daemon. A public scheduling machine accepts job submissions from other machines in the LoadLeveler cluster. To configure a machine as a public scheduling machine, see the **schedd_host** keyword description in "Administration keyword descriptions" on page 134.

**Syntax:**

```
schedd_runs_here = true | false
```

**Default value: true**

**smt**

Indicates the required simultaneous multithreading (SMT) state for all job steps assigned to the class.

**Syntax:**

```
smt = yes | no | as_is
```

Where:

| | |
|---|---|
| **yes** | The job step requires SMT to be enabled. |
| **no** | The job step requires SMT to be disabled. |
| **as_is** | The SMT state will not be changed. |

**Note:** You cannot use the **smt** and **rset** keywords together if **smt** is set to either **yes** or **no**. Using these keywords together will cause your job to fail.

**Default value: as_is**

**Examples:**

```
smt = yes
```

**stack_limit**

Specifies the hard limit, soft limit, or both limits for the size of a stack.

**Syntax:**

```
stack_limit = hardlimit,softlimit
```

For additional information about limit keywords, see the following topics:
- "Syntax for limit keywords" on page 131
- "Using limit keywords" on page 66

**startd_runs_here = true | false**

Specifies whether the startd daemon runs on the host. If you do not want to run the startd daemon, specify **false**.

**Syntax:**

```
startd_runs_here  = true | false
```

**Default value: true**

**type**

Identifies the type of stanza in the administration file.

**Syntax:**

```
type = stanza_type
```

Where *stanza_type* is one of the following:
- Adapter

- Class
- Machine

**Default:** No default value is set.

**wall_clock_limit**

Specifies the hard limit, soft limit, or both limits for the amount of elapsed time for which a job can run.

**Syntax:**

```
wall_clock_limit = hardlimit,softlimit
```

Note that LoadLeveler uses the time the negotiator daemon dispatches the job as the start time of the job. When a job is checkpointed, vacated, and then restarted, the **wall_clock_limit** is not adjusted to account for the amount of time that elapsed before the checkpoint occurred.

If you are running the BACKFILL scheduler, you must set a wall clock limit either in the job command file or in a class stanza (for the class associated with the job you submit). LoadLeveler administrators should consider setting a default wall clock limit in a default class stanza.

For additional information about limit keywords, see the following topics:
- "Syntax for limit keywords" on page 131
- "Using limit keywords" on page 66

# Chapter 8. Commands

LoadLeveler resource manager contains commands that are available to all users of LoadLeveler as well as commands that are reserved only for LoadLeveler administrators.

If security services are not configured, then administrators are identified by the **LOADL_ADMIN** keyword in the configuration file. If security services are configured, the configuration file must identify the administrator's group. For additional information on defining security mechanisms, see the "Configuring the LoadLeveler environment" topic in *LoadLeveler: Using and Administering*.

The administrator commands can operate on the entire LoadLeveler job queue and all machines configured. The user commands mainly affect those jobs submitted by that user.

Table 24 lists:
- All of the LoadLeveler resource manager commands
- The intended users
- The supported operating systems
- A reference to the full description of each command

*Table 24. LoadLeveler resource manager command summary*

| Command name | Intended users | Supported operating systems | For more information, see... |
|---|---|---|---|
| **llracctmrg** | Administrators only | AIX and Linux | "llracctmrg - Collect machine history files" on page 153 |
| **llrchgmstat** | Administrators only | Linux | "llrchgmstat - Switch the compute node state" on page 155 |
| **llrclusterauth** | Administrators only | AIX and Linux | "llrclusterauth - Generates public and private keys" on page 157 |
| **llrconfig** | Administrators only | AIX and Linux | "llrconfig - Manage the LoadLeveler configuration" on page 158 |
| **llrctl** | Administrators only | AIX and Linux | "llrctl - Control LoadLeveler daemons" on page 166 |
| **llrdupdate** | Administrators only | AIX and Linux | "llrdbupdate - Update the LoadLeveler database" on page 171 |
| **llreinit** | The **root** user | Linux | "llreinit - Prepare energy coefficients" on page 173 |
| **llrinit** | Administrators only | AIX and Linux | "llrinit - Initialize machines in the LoadLeveler cluster" on page 175 |
| **llrmovespool** | Administrators only | AIX and Linux | "llrmovespool - Move job records" on page 177 |
| **llrq** | Both administrators and general users | AIX and Linux | "llrq - Query job information" on page 179 |
| **llrqetag** | Both administrators and general users | Linux | "llrqetag - Query information about energy policy tags" on page 195 |
| **llrrmetag** | Both administrators and general users | Linux | "llrrmetag - Remove the energy policy" on page 198 |

Table 24. LoadLeveler resource manager command summary (continued)

| Command name | Intended users | Supported operating systems | For more information, see... |
|---|---|---|---|
| **llrstatus** | Both administrators and general users | AIX and Linux | "llrstatus - Query machine information" on page 200 |

## llracctmrg - Collect machine history files

Use the **llracctmrg** command to collect individual machine history files together into a single file.

### Syntax

**llracctmrg** [**-?**] [**-H**] [**-v**] [**-R**] [**-h** *hostlist*] [**-d** *directory*]

### Flags

**-?** Provides a short usage message.

**-H** Provides extended help information.

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

**-R** Merges individual machine reservation history files into a single history file.

**-h** *hostlist*
Specifies a blank-delimited list of machines from which to collect data. The default is all machines in the LoadLeveler cluster.

**-d** *directory*
Specifies the directory to hold the new global history file. If not specified, the directory specified in the **GLOBAL_HISTORY** keyword in the configuration file is used.

### Description

This command by default collects data from all the machines identified in the administration file. To override the default, specify a machine or a list of machines using the **-h** flag.

When the **llracctmrg** command ends, accounting information is stored in a file called **globalhist.***YYYYMMDDHHmm*.

where:

| | |
|---|---|
| **YYYY** | Indicates the year |
| **MM** | Indicates the month |
| **DD** | Indicates the day |
| **HH** | Indicates the hour |
| **mm** | Indicates the minute. |

Information such as the amount of resources consumed by the job and other job-related data is stored in this file.

Note that when the collection of accounting information to the global history file is complete, the accounting information is cleared in the history file.

For job data, you can use this file as input to the **llr_get_history** subroutine. For example, if you created the file **globalhist.199808301050**, you can pass **globalhist.199808301050** as the *filename* argument to the **llr_get_history** subroutine to process the accounting information stored in this file.

When the **-R** flag is used to merge reservation history files instead of job history files, a file named **reservation_globalhist.***YYYYMMDDHHmm* is created in the specified directory. You can view reservation data with any text editor. For more information on the format of the reservation history file, see the accounting information in the “Configuring the LoadLeveler environment” topic in *LoadLeveler: Using and Administering*.

Data on processes which fork child processes will be included in the file only if the parent process waits for the child process to end. Therefore, complete data may not be collected for jobs which are not composed of simple parent/child processes. For example, if a LoadLeveler job invokes an **rsh** command to execute some function on another machine, the resources consumed on the other machine will not be collected as part of the accounting data.

## Examples

1. The following example collects data from machines named `mars` and `pluto`:

   ```
   llracctmrg -h mars pluto
   ```
2. The following example collects data from the machine named `mars` and places the data in an existing directory called **merge**:

   ```
   llracctmrg -h mars -d merge
   ```
3. The following example collects reservation history data from all machines in the LoadLeveler cluster:

   ```
   llracctmrg -R
   ```

   You should receive a response similar to the following:

   ```
   llracctmrg: History transferred successfully from
               c94n04.ppd.pok.ibm.com (98 bytes).
   ```

   A file named **reservation_globlhist.200902130915** is generated in the global history file location, assuming **llracctmrg** is issued at the time of 09:15 02/13/2009.

## Results

The following shows a sample system response from the **llracctmrg -h mars -d merge** command:

```
llracctmrg: History transferred successfully from mars (10080 bytes)
```

## Security

LoadLeveler administrators can issue this command.

## llrchgmstat - Switch the compute node state

Use the **llrchgmstat** command to switch the compute node state between working and standby state.

### Syntax

**llrchgmstat -?** | **-H** | **-v** | **-h** *hostlist* **-m** {**suspend** | **resume**}

### Flags

**-?** Provides a short usage message.

**-H** Provides extended help information.

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

**-h** *hostlist*
Is a blank-delimited list of compute nodes.

**-m {suspend | resume}**
Switches the compute node to standby or working state.

where:

**suspend**
Changes the power state of the compute node to standby state.

**resume**
Changes the power state of the compute node to working state.

### Description

The **llrchgmstat** command switches the compute node between standby and working state. The compute node cannot be switched to standby state when a job is still running.

### Examples

1. To switch idle nodes to standby state, issue:

   ```
   llrchgmstat —h node01 node02 node03 node04 node05 —m suspend
   ```

   You should receive output similar to the following:

   ```
   The following nodes were successfully suspended:
   node01.ppd.pok.ibm.com
   node02.ppd.pok.ibm.com

   The following nodes failed to suspend:
   node03.ppd.pok.ibm.com : has running jobs
   node04.ppd.pok.ibm.com : in the standby state (by policy)
   node05.ppd.pok.ibm.com : in the standby state (by command)
   ```
2. To resume node01 from standby state, issue:

   ```
   llrchgmstat —h node01 —m resume
   ```

   You should receive output similar to the following:

   ```
   The following nodes were successfully resumed:
   node01.ppd.pok.ibm.com

   Or if failure occurs:
   ```

```
The following nodes failed to resume:
node01.ppd.pok.ibm.com : has running jobs
```

## Security

LoadLeveler administrators can issue this command.

## llrclusterauth - Generates public and private keys

Use the **llrclusterauth** command to generate public and private keys that are used to provide secure intercluster communications.

### Syntax

**llrclusterauth** [**-?**] | [**-H**] | [**-v**] | [**-k**]

### Flags

| | |
|---|---|
| **-?** | Provides a short usage message. |
| **-H** | Provides extended help information. |
| **-v** | Displays the name of the command, release number, service level, service level date, and lowest level of the operating system to run this release. |
| **-k** | Creates a public key, a private key, a security certificate, and a directory for authorized keys. The keys and certificate are created in the **/var/LoadL/ssl** directory for AIX and in the **/var/opt/LoadL/ssl** directory for Linux. • The private key is stored in **id_rsa** • The public key is stored in **id_rsa.pub** • The security certificate is stored in **id_rsa.cert** • The authorized keys are stored in **authorized_keys** This command must run with **root** authority when using the **-k** flag to create key files. |

If any directory in the path for the security files does not exist, the command will create the directory and set the owner to **root** and set the permissions to '0700'. The key and certificate files will be owned by **root** with permissions of '0600'.

### Description

The **llrclusterauth** command generates public and private keys that are used to provide secure intercluster communications. When multicluster security is configured to use Secure Sockets Layer (SSL), a connection on a secure port will be accepted only if the public key for the node requesting the connection is stored in a file in the authorized keys directory on the node being connected to.

### Standard Error

An error message is issued and the command exits for the following error cases:
- The command process does not have **root** authority
- A required directory for the security files cannot be created
- A security file cannot be created

### Security

LoadLeveler administrators can issue this command.

## llrconfig - Manage the LoadLeveler configuration

Use the **llrconfig** command to manage LoadLeveler's configuration database.

### Syntax

To initialize the LoadLeveler database configuration:

**llrconfig**
**-?** | **-H** | **-v** | **-i** [**-q**] [**-f** *config_file_list*] [**-t** *cluster_configuration_file*]

To initialize the LoadLeveler database configuration:

**llrconfig**
[**-h** *hostlist*] { **-d** *keywords* ... | [**-N**] **-c** *keyword=value* ... }

To work with administration stanzas and keywords in the database configuration:

**llrconfig**
**-s** { **-d** *stanza*[= *value*[*:substanza*] ] | [**-N**] { {**-a** | **-r**} *stanza=value*[*:substanza*] | **-a** **-f** *stanza_file* | **-c** *stanza=value*[*:substanza*] *keyword1=value1*... } }

### Flags

**-?** Provides a short usage message.

**-H** Provides extended help information.

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

**-i** Initializes the configuration database using default LoadLeveler configuration files or the LoadLeveler configuration files specified by the **-f** flag. This flag can be used with the **-f** or **-t** flag.

**-q** Specifies no prompting with the **-i** option.

**-f** *config_file_list*
Is a blank-delimited list of full path names of configuration files that are used to initialize the database when used in conjunction with the **-i** flag. You can specify a list of configuration files with the **-f** flag when used in conjunction with the **-i** flag; however, if a keyword is specified in more than one file, the last value is used. Administration files cannot be specified in the list, but the last value for the **ADMIN_FILE** specified in a file in the *config_file_list* will be used to initialize the database.

**-f** *stanza_file*
Is the name of a file containing administration stanzas to be added to the database when used in conjunction with the **-s** flag. You can only specify one stanza file with the **-f** flag when used in conjunction with the **-s** flag.

The *stanza_file* has the same format as the administration file and it can contain multiple different stanzas.

**-t** *cluster_configuration_file*
Specifies the full path name of an xCAT cluster configuration file used to supply LoadLeveler cluster information to initialize the database. This flag can only be used with the **-i** flag.

**-h** *hostlist*
Is a blank-delimited list of machines. You can only specify the **-h** flag if you

are using the **-c** and **-d** flags to change or display the keywords of these machines. The **-h** flag is not valid when **-s** is used.

**-d** *keywords ...*
Displays the values of the keywords in a blank-delimited list for the global or default machine, or for machines specified with the **-h** flag. If the value of a keyword cannot be displayed, an error message will be printed and the command continues to display the value of the next keyword.

**-d** *stanza***[=** *value***[***:substanza***]]**
When used with **-s**, displays the values of the stanza names or stanza attributes. In stanza mode, all instances of a specified stanza type, for instance all class stanza names, are displayed or a specific stanza or substanza, such as **class=large**, is displayed. Values that come from defaults will not be listed. Only values that have been defined for this stanza will be displayed.

**-N** Specifies do not reconfigure LoadLeveler after the update is made to the database.

**-c** *keyword=value ...*
Changes a keyword *value* to the new *value* provided. One or more strings in a format similar to *keyword=value* must be specified with the **-c** flag. The change applies to machines specified by the **-h** flag or if a machine is not specified, the change applies to the global or default machine.

**-c** *stanza***=***value***[***:substanza***]** *keyword1***=***value1***... }**
When used with **-s**, changes the value of a keyword in a stanza or substanza to the specified value. When specified with the **-s** flag, a stanza type and stanza name is specified in addition to the keyword-value pairs being changed. To change values in a substanza, specify the substanza name as well.

**-s** Specifies that stanza mode is used. The flags that follow it apply to administration stanzas and keywords. The **-s** flag can be used with the **-a**, **-c**, **-d**, or **-r** flag.

In stanza mode, the stanza is specified using this format:

*stanza*= *value*[*:substanza*]

where:

- *stanza* is one of the LoadLeveler stanza types:
  - Class
  - Cluster
  - Group
  - Machine
  - Machine_group
  - Region
  - User
- *value* is the name of a specified stanza instance.
- *substanza* is the name of a specified user or machine substanza instance.

**-a** Adds a new stanza to the configuration without setting values. If the stanza already exists in the database, an error message will be printed out. This flag is only valid in stanza mode (when using the **-s** flag).

**-r** Removes the specified stanza from the database. If the stanza does not exist in database, an error message will be printed out. This flag is only valid in stanza mode (when using the **-s** flag).

## Description

The **llrconfig** command is used to manage LoadLeveler's configuration database. You can only specify one **-i**, **-c**, or **-d** flag at a time. In stanza mode, you can specify only one of the **-a**, **-c**, **-d**, or **-r** flags with the **-s** flag.

To initialize the database configuration of LoadLeveler, **llrconfig -i** provides several ways to put the LoadLeveler configuration and administration information into the database.

Table 25 lists some of your configuration and database information options:

*Table 25. LoadLeveler configuration and database information options*

| Initialize command option: | When to use: | Which files are used: |
|---|---|---|
| **llrconfig -i** | If LoadLeveler is already configured using a file-based configuration | The first configuration file found by searching in these locations:<br>• The configuration file pointed to by the **LoadLConfig** keyword in the **/etc/LoadL.cfg** master configuration file<br>• A default configuration file, **LoadL_config**, found in the home directory of the LoadLeveler user ID<br>• The sample configuration file **LoadL_config.l** from the samples directory |
| **llrconfig -i -f** *config_file_list* | If you want to use one or more configuration files to load the database | The specified files are used, in the order that they were specified. |
| **llrconfig -i -t** *cluster_configuration_file* **-f** *config_file_list* | If you have set up a cluster configuration file for use with xCAT | Any files specified by the **-f** option followed by the LoadLeveler section of the file specified by the **-t** option. |

You can use the **-f** option to list one or more configuration files. If duplicate keywords are specified in the configuration files, the value of the last keyword will be used. If a **LoadL_config.local** file is specified in the list, the keywords found in that file will be set for the default or global configuration, rather than for any specific machine. To specify machine-specific information, you need to set those keywords in the machine or machine_group stanzas in the administration file associated with the configuration file used to load the configuration database, or update the configuration database for specific machines after the initialize command has run. You cannot specify an administration file in the list of configuration files. However, you can supply one administration file to use for initializing the database by assigning it to the **ADMIN_FILE** keyword in a configuration file that is in the list. If **ADMIN_FILE** is specified in more than one configuration file, only the last one will be used to initialize the configuration. The **LOCAL_CONFIG** keyword is ignored during database initialization.

When **llrconfig -i** is run, for any value that the user specifies that is the same as the LoadLeveler code default value, no value will be set in the configuration database.

Running **llrconfig -i** will create some scripts (**llserver.sh** and **llcompute.sh**) in xCAT's postscript directory that can be used to assist in creating the LoadLeveler

**/etc/LoadL.cfg** file, **LOG**, **SPOOL**, and **EXECUTE** directories on the nodes. Add these postscripts to the corresponding xCAT nodegroups' postscript attributes to have them automatically invoked when provisioning the nodes. For example, add an entry for the xCAT postscripts table using the xCAT command:

```
chdef llcompute -p postscripts="llcompute.sh"
```

If any updates are made to change the **LOG**, **SPOOL**, or **EXECUTE** directory using **llrconfig -c**, make sure the **/install/postscripts/LoadL/llr_config_directories** file is also updated to reflect the new directories.

It is suggested you do not run the initialization more than once. If you run the initialize option again after the database has been initialized, the database will be populated again from the configuration files and any updates that were made to configuration parameters will be lost. If you reinitialize with changes made to the defined machines, machine groups, or cluster names, unexpected results may be seen when running **llrconfig -s**.

The initialize option must be run by the **root** user. It must be run on a node where the database or database client is installed.

To update the database after initialization, you can issue the **llrconfig -c** command on any LoadLeveler node that has been set up to access the database using unixODBC. To specify machine-specific information, use the **-h** flag to specify one or more machines. Changes made without the **-h** flag apply to the default or global machine. To make changes to the configuration stanzas, for class, cluster, group, machine, machine_group, region, or user, use **llrconfig -s** and the **-a**, **-c**, or **-r** option.

It is not necessary to reconfigure LoadLeveler to pick up changes made using the **llrconfig -c** command. A reconfiguration will be propagated to the nodes needing reconfiguration. If you want to prevent the reconfiguration when **llrconfig** is issued, use the **-N** option.

When several keyword changes are specified on the command line, if the value of a keyword cannot be changed, an error message will be printed and the command will continue to change the value of the next keyword.

To remove the setting of a keyword, set it using the format **-c** *keyword*= or **-c** *keyword*="". If set without a value, the keyword's value will be removed. To remove an entire stanza, use **llrconfig -s -r** *stanza*.

If *keyword="value"* is specified with the **-c** flag, the quotes will not be part of the assigned value. You must use quotes whenever the value includes shell special characters. For example, `llrconfig -c start = 4 > 5` is **not** correct and should be written as `llrconfig -c start = "4 > 5"`. If a LoadLeveler variable is part of the assignment, use single quotes to prevent resolution by the shell. For example, issue: `llrconfig -c log = '/var/loadl/$(host)'`.

When using the **-d** option, deprecated keywords are not displayed even if the keyword was added using the deprecated format. For example, if **central_manager=true** was defined in the machine stanza, you should issue:

```
llrconfig -d central_manager_list
```

to display the value because the **central_manager** keyword in the machine stanza is deprecated in favor of **central_manager_list**.

## Examples

1. To migrate your existing file-based configuration to the configuration database, issue:

   ```
   llrconfig -i
   ```

2. To initialize a configuration using a cluster configuration file and one of the provided sample files, issue:

   ```
   llrconfig –i –t /tmp/cluster_file  –f \
   /usr/lpp/LoadL/resmgr/full/samples/LoadL_config.l
   ```

   where: **/tmp/cluster_file** contains:

   ```
   # LoadLeveler cluster manager daemons and administrator
   ll-config:
        central_manager_list = hmc01 hmc02 hmc03
        resource_mgr_list = hmc01 hmc02 hmc03
        loadl_admin = loadl root loadladmin
        schedd_list = bbxn01 bbxn02 bbyn01 bbyn02
   ```

3. To initialize a configuration with keywords from the specified files, issue:

   ```
   llrconfig -i -f /usr/lpp/LoadL/resmgr/full/samples/LoadL_config \
   /usr/lpp/LoadL/resmgr/full/samples/LoadL_config.local
   ```

   **Note:** The local file keywords are applied to the global or default machine rather than a particular machine.

4. To show the value of the **polling_frequency**, issue:

   ```
   llrconfig -d polling_frequency
   ```

5. To change the configured default **polling_frequency** to 120, issue:

   ```
   llrconfig -c polling_frequency = 120
   ```

6. To set debug flags for the Schedd daemon on **c197blade3b01**, issue:

   ```
   llrconfig -h c197blade3b01 -c SCHEDD_DEBUG=D_FULLDEBUG
   ```

7. To reset debug flags to use the default value for the Schedd daemon on **c197blade3b01**, issue:

   ```
   llrconfig -h c197blade3b01 -c SCHEDD_DEBUG=
   ```

   or

   ```
   llrconfig -h c197blade3b01 -c SCHEDD_DEBUG=""
   ```

8. To change class initiators on machine **c94n01**, issue:

   ```
   llrconfig -h c94n01 -c class="small(4) large(3)" max_starters = 6
   ```

9. To change **schedd_debug** to **"D_ALWAYS D_FULLDEBUG D_SCHEDD"**, and **master_debug** to **"D_ALWAYS D_FULLDEBUG"** for machine **c94n01**, issue:

   ```
   llrconfig -h c94n01 -c schedd_debug = D_ALWAYS D_FULLDEBUG D_SCHEDD \
   master_debug = D_ALWAYS D_FULLDEBUG
   ```

10. To make several changes and only run reconfiguration after the last, issue:

    ```
    llrconfig -N -c max_starters = 3
    llrconfig -N -c class = "small(5) large(1)"
    llrconfig -c resources = "ConsumableCpus(4) licenseA(1)"
    ```

11. To display all the class stanzas in the configuration database, issue:

    ```
    llrconfig -s -d class
    ```

    You should receive a response similar to the following:

    ```
    small
    inter_class
    large
    ```

12. To display the values set for the class stanza small, issue:

    ```
    llrconfig -s -d class=small
    ```

    You should receive a response similar to the following:

```
small: type=class
  vhu: type = user {
  maxidle = 6
  }
```

13. To display the machines defined in the cluster, issue:

    ```
    llrconfig -s -d machine
    ```

    You should receive a response similar to the following:

    ```
    default
    c917f3jp03.ppd.pok.ibm.com
    c917f3jp04.ppd.pok.ibm.com
    c917f3jp05.ppd.pok.ibm.com
    c917f3jp06.ppd.pok.ibm.com
    c917f3jp07.ppd.pok.ibm.com
    c917f3jp08.ppd.pok.ibm.com
    ```

14. To display the machine groups defined in the database, issue:

    ```
    llrconfig -s -d machine_group
    ```

    You should receive a response similar to the following:

    ```
    mg1
    mg2
    ```

15. To display the values set for machine group mg1, issue:

    ```
    llrconfig -s -d machine_group=mg1
    ```

    You should receive a response similar to the following:

    ```
    mg1: type=machine_group
      machine_list = c1f1b01,c1f1b02,c1f1b03
      c197blade1b01: type = machine {
        schedd_runs_here = true
      }
    ```

16. To display the values set for the user substanza vhu in the class stanza small, issue:

    ```
    llrconfig -s -d class=small:vhu
    ```

17. To add a new group stanza to the database, issue:

    ```
    llrconfig -s -a group=hpc
    ```

    **Note:** The group is added with no attributes.

18. To change the attributes of the class stanza small in the database, issue:

    ```
    llrconfig -s -c class=small max_queue=10 cpu_limit=30
    ```

19. To add new class stanzas and machine stanzas that are contained in a stanza file, create the file containing the stanzas to add. This shows the contents of an example file named **patch_admin.file**:

    ```
     #  Add two classes
          interactive:       type = class
                    wall_clock_limit = 20:30,20:00
                    priority = 100
          POE:            type = class
                    wall_clock_limit = 24:00:00,24:00:00
                    priority = 55
          # 4 new nodes using the default attributes
          n12:       type = machine
          n23:       type = machine
          n34:       type = machine
          n45:       type = machine
    ```

    Then issue:

    ```
    llrconfig -s -a -f  patch_admin.file
    ```

20. To remove an existing user stanza "joe" from the database, issue:

```
llrconfig -s -r user=joe
```

21. To change the value of **schedd_runs_here** for machine i08n008, which is part of machine_group mg4, first view the machine group to determine if there already is a machine substanza for the machine you want to modify in the machine group, by issuing:

    ```
    llrconfig -s -d machine_group=mg4
    ```

    If there is no machine substanza, you should receive a response similar to the following:

    ```
    mg4: {
            type=machine_group
            machine_list=i08n001-i08n008,
            max_starters=32
            island=island1
            class=No_Class(32) interactive(32) large(32)
            resources=ConsumableCpus(all) ConsumableMemory(65536)
    }
    ```

    If there is no substanza for the machine i08n008, add one by issuing:

    ```
    llrconfig -s -a machine_group=mg4:i08n008
    ```

    This adds an empty machine substanza to the machine_group.

    Next, set the value of **schedd_runs_here=true** for machine i08n008, by issuing:

    ```
    llrconfig -s -c machine_group=mg4:i08n008 schedd_runs_here=true
    ```

    Finally, view the change in the machine group by issuing:

    ```
    llrconfig -s -d machine_group=mg4
    ```

    You should receive a response similar to the following:

    ```
    mg4: {
            type=machine_group
            machine_list=i08n001-i08n008,
            max_starters=32
            island=island1
            class=No_Class(32) interactive(32) large(32)
            resources=ConsumableCpus(all) ConsumableMemory(65536)
            i08n008.ppd.pok.ibm.com: {
                    type=machine
                    schedd_runs_here=true
            }
    }
    ```

## Results

- When no errors are found:

  ```
  [c197blade8b14][/]> llrconfig -i
  LoadLeveler tables are being created in the xcatdb database.
  Initializing configuration data in the LoadLeveler tables from the
  following file or files:

  /u/loadl/LoadL_config.c197blade8b14

  Continue? (y/n): y
  llrconfig: The database initialization has completed.
  ```

- When information messages or errors are found during configuration check, and the configuration database is still initialized:

  ```
  [c197blade8b14][/]> llrconfig -i
  LoadLeveler tables are being created in the xcatdb database.
  Initializing configuration data in the LoadLeveler tables from the
  following file or files:
  /u/loadl/LoadL_config
  Continue? (y/n): y
  negotiator_debug: The value, d_loadl, is not valid.
  ```

```
starter_debug: The keyword is specified in the configuration but has no
value set.
trunc_gsmonitor_log_on_open: The keyword is no longer supported.
releasedir: can't read "/opt/ibmll/LoadL/fulla".
bin: can't read "$(RELEASEDIR)/bin".
lib: The keyword is no longer supported.
acct_validation: can't execute "$(BIN)/llacctval".
kbdd: can't execute "$(BIN)/LoadL_kbdd".
startd: can't execute "$(BIN)/LoadL_startd".
process_tracking_extension: can't read "$(BIN)".
region_mgr: can't execute "$(BIN)/LoadL_region_mgr".
resource_mgr: can't execute "$(BIN)/LoadL_resource_mgr".
schedd: can't execute "$(BIN)/LoadL_schedd".
negotiator: can't execute "$(BIN)/LoadL_negotiator".
starter: can't execute "$(BIN)/LoadL_starter".
master: can't execute "$(BIN)/LoadL_master".
llrconfig: The database initialization has completed.
```

## Security

Only the **root** user can issue this command with the **-i** flag on a node running the database software.

LoadLeveler administrators must login to a node with database access through unixODBC and have ODBC set up for their IDs providing permission to issue this command with the **-c**, **-d**, **-a**, or **-r** flag.

## llrctl - Control LoadLeveler daemons

Use the **llrctl** command to control LoadLeveler daemons on all members of the LoadLeveler cluster.

### Syntax

**llrctl** { **-?** | **-H** | **-v** | [**-q**] [**-g** | **-h** *host*] *keyword* }

### Flags

**-?** Provides a short usage message.

**-H** Provides extended help information.

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

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

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

**-h** *host*
Indicates that the command applies to only the *host* machine in the LoadLeveler cluster. If **-h** is not specified, the default is the machine on which the **llrctl** command is issued.

*keyword*
Must be specified after all flags and can be the following:

**capture** *eventname*
Captures accounting data for all jobs running on the designated machines. *eventname* is a user-defined name associated with the data when you run this command. It must be a character string containing no blanks. The data is captured in the accounting records in the history file. For more information, see the discussion of collecting job resource data based on events in *LoadLeveler: Using and Administering*.

**ckconfig**
Checks the defined configuration. **ckconfig** reads either the configuration database or the configuration files (including the global, local, and the administration files), and reports all error and informational messages. The errors and informational messages produced by **ckconfig** will be sent to standard error and will also be logged to the **ckconfigLog** file located in the directory specified in **$LOG** in the configuration file.

**drain [schedd | startd [***classlist* **| allclasses]]**

- When you issue **drain** with no options, the following occurs:
  1. No more LoadLeveler jobs can begin running on this machine.
  2. No more LoadLeveler jobs can be submitted through this machine.
- When you issue **drain schedd**, the following occurs:
  1. The Schedd machine accepts no more LoadLeveler jobs for submission.
  2. Job steps in the Starting or Running state in the Schedd queue are allowed to continue running.
  3. Job steps in the Idle state in the Schedd queue are drained, that is, they will not get dispatched.

- When you issue **drain startd**, the following occurs:
  1. The startd machine accepts no more LoadLeveler jobs to be run.
  2. Job steps already running on the startd machine are allowed to complete.
- When you issue **drain startd** ***classlist***, the classes you specify that are available on the startd machine are drained (made unavailable).
- When you issue **drain startd allclasses**, all available classes on the startd machine are drained.

**`dumplogs [buffer | locks | flush]`**
: When the logging buffer or lock recording is enabled, **llrctl dumplogs** will cause the log messages in the logging buffer or locking records to be written to the related log file.

  where:

  **`buffer`**
  : Dumps out logs in the circular logging buffer. This is the default type. For more information, see "Controlling the logging buffer" on page 34.

  **`locks`**
  : Dumps out locking records. For more information, see "Controlling locking records" on page 35.

  **`flush`**
  : Terminates running jobs on this machine and sends them back, in the Idle state, to the negotiator to await redispatch (provided **restart=yes** is in the job command file). No new jobs are sent to this machine until resume is issued. This keyword forces a checkpoint if jobs are enabled for checkpointing. However, the checkpoint is canceled if it does not complete within the time period specified in the **ckpt_time_limit** keyword in the job command file.

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

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

**`resume [schedd | startd [`*`classlist`*` | allclasses]]`**
- When you issue **resume schedd**, the following occurs:
  1. The Schedd machine accepts LoadLeveler jobs for submission.
  2. Job steps in the Idle state in the Schedd queue are resumed, that is, they will be considered for dispatch.
- When you issue **resume startd**, the startd machine accepts LoadLeveler jobs to be run.
- When you issue **resume** with no options, the actions described for both **resume schedd** and **resume startd** are taken.
- When you issue **resume startd** ***classlist***, the startd machine resumes the execution of those job classes you specify that are also configured (defined on the machine).
- When you issue **resume startd allclasses**, the startd machine resumes the execution of all configured classes.

**`rmshm`**
: Allows **root** to remove all LoadLeveler shared memory segments and associated semaphores.

**start [drained]**

- When you issue **start** with no options, it starts the LoadLeveler daemons on the machine or machines designated, either explicitly or implicitly.
- When you issue **start** without the **-g** or **-h** flag, the LoadLeveler daemons are started on the same machine that issued the command.
- When you issue **llrctl start** with either the **-g** or **-h** flag, the command specified by the **LL_RSH_COMMAND** configuration file keyword is used to run the command on all machines specified in the administration file. If **LL_RSH_COMMAND** is not specified, remote shell (**rsh**) is used and you must have **rsh** privileges in order to use **llrctl start** with either the **-g** or **-h** flag.
- When you issue **start** with the **drained** option, the LoadLeveler daemons are started, but the startd daemon is started in the drained state.

**stop**
Stops the LoadLeveler daemons on the specified machine.

**suspend**
Suspends all jobs on this machine. This is not supported for parallel jobs.

**version**
Displays release number, service level, service level date, and operating system information for every LoadLeveler executable.

When you issue **llrctl start** or **llrctl version** with either the **-g** or **-h** flag, the command specified by the **LL_RSH_COMMAND** configuration file keyword is used to run the command on all machines specified in the administration file. If **LL_RSH_COMMAND** is not specified, remote shell (**rsh**) is used and you must have **rsh** privileges in order to use **llrctl start** or **llrctl version** with either the **-g** or **-h** flag.

LoadLeveler commands that run **rsh** include **llrctl version** and **llrctl start**.

## Description

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

- To perform the control operations of the **llrctl** command, you must be a LoadLeveler administrator. The only exceptions to this rule are the **start** and **rmshm** operations.
- LoadLeveler will fail to start if any value has been set for the **MALLOCTYPE** environment variable.
- For a file-based configuration, after you make changes to the administration and configuration files for a running cluster, be sure to issue **llrctl reconfig**. This command causes the LoadLeveler daemons to reread these files, and prevents problems that can occur when the LoadLeveler commands are using a new configuration while the daemons are using an old configuration.

  For the database configuration option, you should reconfigure after you update the configuration database using the configuration editor.

  **Note:** Changes to **SCHEDULER_TYPE** will not take effect at reconfiguration. The administrator must stop and restart or recycle LoadLeveler when changing **SCHEDULER_TYPE**.
- The **llrctl drain startd** *classlist* command drains classes on the startd machine, and the startd daemon remains operational. If you reconfigure the daemon, the

draining of classes remains in effect. However, if the startd machine goes down and is brought up again (either by the master daemon or by a LoadLeveler administrator), the startd daemon is configured according to the global or local configuration file in effect, and therefore the draining of classes is lost.

Draining all the classes on a startd machine is *not* equivalent to draining the startd machine. When you drain all the classes, the startd enters the Idle state. When you drain the startd, the startd enters the Drained state. Similarly, resuming all the classes on a startd machine is *not* equivalent to resuming the startd machine.

- If a job step is running on a machine that receives the **llrctl recycle** command, or the **llrctl stop** and **llrctl start** commands, the running job step is terminated. If the **restart** option in the job command file was set to **yes**, then the job step will be restarted when LoadLeveler is restarted. If the job step is checkpointable, it will be restarted from the last valid checkpoint file when LoadLeveler is restarted.
- If you find that the **llrctl -g** command (even if it is specified with additional options) is taking a long time to complete, you should consider using the xCAT command **xdsh** to send **llrctl** commands (omitting the **-g** flag) to multiple nodes in a parallel fashion.

## Examples

1. This example stops LoadLeveler on the machine named *iron*:
   ```
   llrctl -h iron stop
   ```
2. This example starts the LoadLeveler daemons on all members of the LoadLeveler cluster (with the exception of the submit-only machines), starting with the central manager, as defined in the machine stanzas of the administration file:
   ```
   llrctl -g start
   ```
3. This example causes the LoadLeveler daemons on machine *iron* to re-read the administration and configuration files, which may contain new configuration information for the *iron* machine:
   ```
   llrctl -h iron reconfig
   ```
4. This example drains the classes *medium* and *large* on the machine named *iron*:
   ```
   llrctl -h iron drain startd medium large
   ```
5. This example drains the classes *medium* and *large* on all machines:
   ```
   llrctl -g drain startd medium large
   ```
6. This example stops all the jobs on the system, then allows only jobs of a certain class (*medium*) to run:
   ```
   llrctl -g drain startd allclasses
   llrctl -g flush
   llrctl -g resume
   llrctl -g resume startd medium
   ```
7. This example resumes the classes *medium* and *large* on the machine named *iron*:
   ```
   llrctl -h iron resume startd medium large
   ```
8. This example illustrates how to capture accounting information on a work shift called *day* on the machine *iron*:
   ```
   llrctl -h iron capture day
   ```
   You can capture accounting information on all the machines in the LoadLeveler cluster by using the **-g** option, or you can collect accounting information on the local machine by simply issuing the following:
   ```
   llrctl capture day
   ```

Capturing information on the local machine is the default. For more information, see the "Collecting job resource data based on events" topic in *LoadLeveler: Using and Administering*.

9. This example saves the locking records for LoadLeveler daemons configured with the **D_LOCK_TRACE** debug flag to the daemon's corresponding existing log files in the **LOG** directory of node c197blade7b02:

   ```
   llrctl -h c197blade7b02 dumplogs locks
   ```

## Security

LoadLeveler administrators can issue this command.

Only the **root** user can issue this command with the **rmshm** keyword.

## llrdbupdate - Update the LoadLeveler database

Use the **llrdbupdate** command to update the LoadLeveler database when service is applied.

### Syntax

**perl llrdbupdate -? | -H | -v | [-f] [-t]**

### Flags

**-?** Provides a short usage message.

**-H** Provides extended help information.

**-v** Displays the name of the command, release number, and service level date.

**-f** Forces a database update. Database version checking will be ignored. Normally, this command checks to see that the version is a valid release number, service level, service level date, and lowest level of the operating system to run this release.

**-t** Displays trace output for the SQL commands and execution log.

### Description

For clusters using the database option for configuration, run the **llrdbupdate** command on the node running the database when service is applied. **llrdbupdate** ensures that the configuration database is synchronized with the version of LoadLeveler in the service update. It needs to be run once regardless of how many nodes are in the cluster and only on a node that is configured to access the database.

You cannot use this command to back out database changes by running an earlier version of **llrdbupdate** against a later version of the database. It is suggested that you back up the database before running this command.

**llrdbupdate** is a Perl script and must be invoked with the **perl** command.

### Examples

1. To update the database after an update is installed, issue:
   ```
   [root@c197blade4b06 ]# perl /usr/lpp/LoadL/resmgr/full/bin/llrdbupdate
   ```
   You should receive a response similar to the following:
   ```
   llrdbupdate: The LoadLeveler database has been updated to version new_version
                successfully.
   ```
2. To update the database to the latest version and put trace information into a file called **/tmp/llrdbupdate.log**, issue:
   ```
   [root@c197blade4b06 ]# perl /usr/lpp/LoadL/resmgr/full/bin/llrdbupdate -t \
   >/tmp/llrdbupdate.log 2>&1
   ```
   The contents of the file **/tmp/llrdbupdate.log** will be similar to the following:
   ```
   llrdbupdate: Successfully executed: SELECT schema()
   llrdbupdate: Successfully executed: select lldbVersion as version from
                TLL_Cluster where clusterID=(select DISTINCT clusterID from
                TLL_Nodelist)
   llrdbupdate: Successfully executed: SELECT TABLE_NAME as name, TABLE_NAME
                as base_name FROM information_schema.TABLES WHERE
                TABLE_SCHEMA='lldb' AND TABLE_NAME LIKE 'TLL%'
   llrdbupdate: Successfully executed: SELECT TABLE_NAME as name FROM
   ```

```
             information_schema.VIEWS WHERE TABLE_SCHEMA='lldb' AND
             TABLE_NAME LIKE 'VLL%'
...
llrdbupdate: The LoadLeveler database has been updated to version new_version
             successfully.
```

3. To force a database update, issue:

   ```
   [root@c197blade4b06 ]# perl /usr/lpp/LoadL/resmgr/full/bin/llrdbupdate -f
   ```

   You should receive a response similar to the following:

   ```
   llrdbupdate: The LoadLeveler database has been updated to version new_version
                successfully.
   ```

## Security

Only the **root** user can issue this command.

## llreinit - Prepare energy coefficients

Use the **llreinit** command to prepare energy coefficients for the LoadLeveler energy function.

### Syntax

**llreinit -?** | **-H** | **-v** | **-l** | [**-s** {**rsh** | **ssh** | **xdsh**}] [**-h** *node_list* | **-a** *new_node_list*] **-d** *benchmark_dir*

### Flags

**-?** Provides a short usage message.

**-H** Provides extended help information.

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

**-l** Loads the energy coefficients into the database. It does not run the benchmarks on the compute nodes to calculate the coefficients.

**-s rsh | ssh | xdsh**
Specifies which remote execution command will be used to run the energy initialization command on the remote node. The default command is **rsh**.

**-h** *node_list*
Specifies the compute nodes that need to rerun the benchmarks. This option is used when there was a failure to initialize the energy function on some compute nodes.

**-a** *new_node_list*
Specifies the new nodes that need to be added in the cluster.

**-d** *benchmark_dir*
Specifies the location of the energy benchmarks.

### Description

The **llreinit** command initializes the environment for the energy function. This command will run benchmarks on each compute node, collect hardware counters for benchmarks, and load energy coefficients into the database. This command is run after the system administrator runs the **llrconfig -i** command to initialize the LoadLeveler database.

This command is available only when the database-based configuration is used by LoadLeveler.

The **LL_CENTRAL_NODE** environment variable can be set to a specific host name. If the **llreinit** command is running on the specified host to prepare energy coefficients, make sure all the compute nodes can access the host without a password. If **LL_CENTRAL_NODE** is not set to a specific host name, the host name that *hostname* **-s** displays will be used.

### Examples

1. To initialize energy coefficients for LoadLeveler energy functions, issue:
   ```
   llreinit -s ssh -d /benchmarks
   ```

   You should receive a response similar to the following:

```
Load coefficients into database.
Energy aware management initialization has successfully completed.
```

This command will run benchmarks on every compute node in the cluster. The coefficients will be saved into the database if no error occurs.

You might receive a response similar to the following:

```
Connections cannot be made to all compute nodes from c596n17,
check /etc/energy/failed_node_list for details.
```

2. To rerun benchmarks on any nodes where the command failed to get a coefficients file because of network problems, correct the network error and then rerun the initialization command for these nodes by issuing:

   ```
   llreinit -d /benchmarks -h c596n19 c596n20
   ```

   You should receive a response similar to the following:

   ```
   Load coefficients into database.
   Energy aware management initialization has successfully completed.
   ```

3. To initialize the energy function on nodes that have been added to the cluster after the energy function was initialized, issue:

   ```
   llreinit –d /benchmarks –a c596n19 c596n20
   ```

   You should receive a response similar to the following:

   ```
   The added nodes are ready to be used for energy aware management.
   ```

   In this example, nodes c596n19 and c596n20 are added to the cluster with the energy function enabled.

## Security

Only **root** users can issue this command on a node where the UNIX ODBC library was installed to access the database.

## llrinit - Initialize machines in the LoadLeveler cluster

Use the **llrinit** command to initialize a new machine as a member of the LoadLeveler cluster.

### Syntax

**llrinit** [**-?**] [**-H**] [**-q**] [**-prompt**] [**-local** *pathname*] [**-release** *pathname*] [**-debug**]

### Flags

**-?** Provides a short usage message.

**-H** Provides extended help information.

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

**-prompt**
Prompts or leads you through a set of questions that help you to complete the **llrinit** command.

**-local** *pathname*
*pathname* is the local directory in which the spool, execute, and log subdirectories will be created. The default, if this flag is not used, is the home directory.

There must be a unique local directory for each LoadLeveler cluster member.

**-release** *pathname*
*pathname* is the release directory, where the LoadLeveler bin, man, include, and samples subdirectories are located. The default, if this flag is not used, is the **/usr/lpp/LoadL/full** directory on AIX or the **/opt/ibmll/LoadL/full** directory on Linux.

**-debug**
Displays debug messages during the execution of **llrinit**.

### Description

This command runs once on each machine during the installation process. It must be run by the user ID you have defined as the LoadLeveler user ID. The **log**, **spool**, and **execute** directories are created with the correct modes and ownerships. The LoadLeveler configuration and administration files, **LoadL_config** and **LoadL_admin**, respectively, are copied from LoadLeveler's release directory to LoadLeveler's home directory. The local configuration file, **LoadL_config.local**, is copied from LoadLeveler's release directory to LoadLeveler's local directory.

**llrinit** initializes a new machine as a member of the LoadLeveler cluster by doing the following:

- Creates the following LoadLeveler subdirectories with the given permissions:
  - **spool** subdirectory, with permissions set to 700.
  - **execute** subdirectory, with permissions set to 1777.
  - **log** subdirectory, with permissions set to 775.
- Copies the **LoadL_config** and **LoadL_admin** files from the release directory samples subdirectory into the home directory of the LoadLeveler user ID.
- Copies the **LoadL_config.local** file from the release directory samples subdirectory into the local directory.
- Creates symbolic links from the loadl home directory to the spool, execute, and log subdirectories and the **LoadL_config.local** file in the local directory (if home and local directories are not identical).

- Creates symbolic links from the home directory to the **bin**, **man**, **samples**, and **include** subdirectories in the release directory.
- Updates the **LoadL_config** with the release directory name.

Before running **llrinit** ensure that your HOME environment variable is set to LoadLeveler's home directory. To run **llrinit**, you must have:
- Write privileges in the LoadLeveler home directory
- Write privileges in the LoadLeveler release directory
- Write privileges in the LoadLeveler local directory.

## Examples

The following example initializes a machine, assigning **/var/loadl** as the local directory and **/usr/lpp/LoadL/resmgr/full** as the release directory.

```
llrinit -local /var/loadl -release /usr/lpp/LoadL/resmgr/full
```

Ensure that the local directory exists before running the preceding command.

## Results

The command:

```
llrinit -local /home/ll_admin -release /usr/lpp/LoadL/resmgr/full
```

will yield the following output:

```
llrinit: creating directory "/home/ll_admin/spool"
llrinit: creating directory "/home/ll_admin/log"
llrinit: creating directory "/home/ll_admin/execute"
llrinit: set permission "700" on "/home/ll_admin/spool"
llrinit: set permission "775" on "/home/ll_admin/log"
llrinit: set permission "1777" on "/home/ll_admin/execute"
llrinit: creating file "/home/ll_admin/LoadL_admin"
llrinit: creating file "/home/ll_admin/LoadL_config"
llrinit: creating file "/home/ll_admin/LoadL_config.local"
llrinit: editing file /home/ll_admin/LoadL_config
llrinit: editing file /home/ll_admin/LoadL_admin
llrinit: creating symbolic link "/home/ll_admin/bin ->
        /usr/lpp/LoadL/full/bin"
llrinit: creating symbolic link "/home/ll_admin/man ->
        /usr/lpp/LoadL/full/man"
llrinit: creating symbolic link "/home/ll_admin/samples ->
        /usr/lpp/LoadL/full/samples"
llrinit: creating symbolic link "/home/ll_admin/include ->
        /usr/lpp/LoadL/full/include"
llrinit: program complete.
```

## Security

LoadLeveler administrators can issue this command.

## llrmovespool - Move job records

Use the **llrmovespool** command to move the job records from the spool of one job manager to another job manager in the local cluster. The **llrmovespool** command is intended for recovery purposes only.

### Syntax

**llrmovespool** { **-?** | **-H** | **-v** | [**-d** *directory*] **-h** *hostname* }

### Flags

**-?** Provides a short usage message.

**-H** Provides extended help information.

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

**-d** *directory*
Indicates the directory containing the job queue whose job records are to be moved. If not specified, the job queue in the current working directory will be moved. The specified directory must be accessible from the machine upon which the command is issued.

**-h** *hostname*
Indicates the host name of the machine running the job manager daemon (**LoadL_schedd** daemon), which manages the job queue database that will receive the job records being moved. This flag has no default and is required.

### Description

The **llrmovespool** command must be run from a machine that has read and write access to the specified **spool** directory containing the job records being moved. This machine must also have network connectivity to the machine running the job manager daemon that will receive the job records. Jobs to be moved can be in any state.

The job manager daemon daemon that created the job records to be moved must not be running during the move operation. Jobs within the job queue database will be unrecoverable if the job queue is updated during the move by any process other than the **llrmovespool** command. The job manager daemon that created the job records to be moved must have the **schedd_fenced** machine stanza keyword set to **true** prior to the **llrmovespool** command being issued.

All moved jobs retain their original job identifiers. The **llrmovespool** command reports the status of each job as it is processed. When the job records for a job are successfully transferred, the **schedd_host** of the job is updated to represent the new job manager daemon and the job records in the specified **spool** directory are deleted. The successful status is reported to standard output. If the transfer for any step within a job fails, the job records for that step remain in the specified **spool** directory and the error status is reported to standard error. If for some reason a job fails, the **llrmovespool** command should be reissued against the specified **spool** directory to reprocess the job.

The command can be issued by administrators only.

## Standard Error

An error message is issued and the command exits for the following error cases:
- The command was not issued with the required **-h** flag.
- The machine stanza for the machine running the job manager daemon, which manages the job queue database that is receiving the job records, has the **schedd_fenced** keyword set to **true**.
- The machine stanza for the machine running the job manager daemon, which manages the job queue database being moved, does not have the **schedd_fenced** keyword set to **true**.
- The specified host name is not a valid machine.
- The command cannot make a connection to the specified host name.
- The specified directory does not exist.
- The job records within the specified directory cannot be accessed.
- There are no job records within the specified **spool** directory.
- The job manager daemon on the specified host name cannot accept the transferred job because a job with the same job identifier already exists.

## Examples

This example moves the job records found in **/tmp/tmp_spool** to the job manager daemon running on the **c188f2n08** machine:

```
llrmovespool -d /tmp/tmp_spool -h c188f2n08
```

You should receive output similar to the following:

```
The job spool records in /tmp/tmp_spool are being moved to c188f2n08.
The records for job c188f2n02.ppd.pok.ibm.com.1@c188f2n02.ppd.pok.ibm.com
were successfully transferred.

The transfer is complete.
```

## Security

LoadLeveler administrators can issue this command.

## llrq - Query job information

Use the **llrq** command to query information about jobs in the resource manager job database.

### Syntax

**llrq** [**-?**] [**-H**] [**-v**] [**-W**] [**-l**] [**-m**] [**-w**] [**-j** *joblist* | *joblist*] [**-u** *userlist*]
[**-h** *hostlist*] [**-f** *category_list*] [**-r** *category_list*]
[**-S** {**total** | **user** | **group** | **class**}]

### Flags

**-?** Provides a short usage message.

**-H** Provides extended help information.

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

**-W** Specifies that the width of columns in tabular output will be increased to fit the widest entry.

**-l** Specifies that a long listing be generated for each job for which status is requested.

**-m** Specifies that a medium listing be generated for each job for which status is requested. The difference between the medium listing and the long listing is that the medium listing does not display any Node or Task information for running job steps, while the long listing does.

**-w** Provides Workload Manager (WLM) CPU, real memory, virtual memory, and large page statistics for jobs in the running state. This flag can be used with a joblist, steplist, or a single stepid. All other flags except **-h** will result in an error message.

**-j** *joblist* |*joblist*
Is a blank-delimited list of resource manager job and step IDs. When a job identifier is specified, the command action is taken for all steps of the job. At least one job or step identifier must be specified.

**-u** *userlist*
Is a blank-delimited list of users. Only job steps belonging to users in this list are queried.

**-h** *hostlist*
Is a blank-delimited list of machines. Only job steps managed by the job manager on machines in this list are queried. When the **-h** flag is used with the **-w** flag, only a single machine name can be specified to obtain the WLM statistics for that machine.

**-f** *category_list*
Is a blank-delimited list of categories to query. Each category specified must be preceded by a percent sign. The *category_list* cannot contain duplicate entries. This flag is used to create a customized version of the standard **llrq** listing. You cannot use this flag with the **-l** or **-m** flags. The output fields produced by this flag all have a fixed length. The output is displayed in the order that the categories are specified. *category_list* can be one or more of the following:

| | |
|---|---|
| **%a** | Account number |
| **%cc** | Completion code |
| **%dc** | Completion date |
| **%dd** | Dispatch Date |

| | |
|---|---|
| **%dh** | Hold date |
| **%dq** | Queue date ("Submitted" date of "standard" **llrq** output) |
| **%gl** | LoadLeveler group |
| **%gu** | UNIX group |
| **%h** | Host name (first host name if more than one machine is allocated to the job step) |
| **%id** | Step ID |
| **%is** | Virtual image size |
| **%jn** | Job name |
| **%jt** | Job type |
| **%nh** | Number of hosts allocated to the job step |
| **%o** | Job owner |
| **%sn** | Step name |
| **%st** | Status |

`-r` *`category_list`*
Is a blank-delimited list of categories to query. Each category you specify must be preceded by a percent sign. You cannot use this flag with the **-l** or **-m** flags. The output produced by this flag is considered raw, in that the fields can be variable in length. Output fields are separated by an exclamation point (!). *category_list* can be one or more of the formats listed under the **-f** flag.

**`-S {total | user | group | class}`**
Displays a high-level summary of the number of jobs in each of the Waiting, Pending, Running, Held, or Preempted states. If total is specified, only one line containing totals for the entire job queue is displayed. If user, group, or class is specified, then one line for each user, group or class, respectively, is displayed. If the **-S** flag is used with any other flag, an error message is generated.

## Description

The **llrq** command queries information about jobs in the LoadLeveler queues.

**Note the following when using this command:**

- If a job step is not specified and if **-u** or **-h** is not specified, all jobs are queried.
- If a job step is specified, you cannot specify **-u** or **-h**, except in the case of **-w**, for which the **-h** flag has special meaning.
- When **-u** or **-h** are used in combination, the result is the intersection of the job steps selected by each flag.
- The **-r** and **-f** flags are mutually exclusive. If specified together, the following error message is displayed:
  ```
  2512-038 The -r option and the -f option are not compatible.
  ```
- You cannot specify **-S** in combination with any other flag.
- The **-m** flag cannot be used in combination with the **-l** or **-x** flags.
- The **-h**, **-j**, **-u**, **-c**, and **-R** options can be used in combination with **-m** to filter the request based on managing host; job or step ID; user; class; or reservation, respectively.

The **long listing** includes the following fields. See Appendix A, "llrq -l command output listing," on page 321 for sample output.

**Account**
The account number specified in the job command file.

**Adapter Requirement**
Reflects the settings of the **network** keyword in the job command file.

For more information on the **network** keyword statement, see the "Job command file keyword descriptions" topic in *LoadLeveler: Using and Administering*.

**Allocated Hosts**
The machines that have been allocated for this job step.

**Args** Arguments that were passed to the executable.

**Blocking**
Reflects the settings for the **blocking** keyword in the job command file.

**Bulk Transfer**
Indicates that the value will be Yes or No depending on whether the application requested that the communication subsystem use bulk transfer by specifying `bulkxfer=yes` in the job command file.

**Checkpoint Directory**
Value of the **ckpt_dir** keyword.

**Checkpoint SubDir**
Value of the**ckpt_subdir** keyword.

**Checkpointable**
Indicates if LoadLeveler considers the job step checkpointable (yes, no, or interval).

**Ckpt Execute Dir**
The directory where the job step's executable will be saved for checkpointable jobs.

**Ckpt Hard Limit**
Checkpoint hard limit as specified at job step submission.

**Ckpt Soft Limit**
Checkpoint soft limit as specified at job step submission.

**Class** The class of the job step as specified at job submission.

**Cmd** The name of the executable associated with the **executable** keyword (if specified) or the name of the job command file.

**Comment**
The comment specified by the **comment** keyword in the job command file.

**Completion Code**
The status returned by the wait3 UNIX system call.

**Completion Date**
Date and time job completed or exited.

**Core Hard Limit**
Core hard limit as specified at job submission.

**Core Soft Limit**
Core soft limit as specified at job submission.

**Coschedule**
Indicates whether the job step is required to be coscheduled (**yes** or **no**).

**CPU Frequency**
The CPU frequency.

**Cpu Hard Limit**
CPU hard limit as specified at job submission.

**Cpu Soft Limit**
CPU soft limit as specified at job submission.

**Cpus Per Core**
The CPUs per processor core requested by the job.

**Data Hard Limit**
Data hard limit as specified at job submission.

**Data Soft Limit**
Data soft limit as specified at job submission.

**Data Staging Script**
The script that will be run for data staging in this job step.

**Data Stg Dependency**
An expression specifying how this step is dependent on other data staging steps in the job.

**Dependency**
Job step dependencies as specified at job submission.

**Dispatch Time**
The time that the job was dispatched.

**Eligibility Time**
The last time the job became eligible for dispatch.

**Energy policy tag**
The name of the energy policy tag associated with the job.

**Env** Environment variables to be set before executable runs. Appears only when the **-x** option is specified.

**Err** The file to be used for stderr.

**Error Text**
The error text in the Blue Gene job record from the Blue Gene DB2 database. This field is displayed for Blue Gene jobs only.

**Est. Execution Time**
The estimated execution time.

**Est. Power Consumed**
The estimated power consumed.

**File Hard Limit**
File hard limits as specified at job submission.

**File Soft Limit**
File soft limit as specified at job submission.

**Hold Job Until**
Job step is deferred until this date and time.

**In** The file to be used for stdin.

**Initial Working Dir**
The directory from which the job step is run. The relative directory from which the stdio files are accessed, if appropriate.

**Job Accounting Key**
The Job Accounting Key is a unique identifier for a LoadLeveler job step. The accounting key is stored in the AIX accounting record for each process associated with a LoadLeveler job step. This field can be used to correlate

AIX accounting records with LoadLeveler accounting records. The Job Accounting Key is stored in the history file and can be accessed using the Accounting API.

This keyword is not applicable on LoadLeveler for Linux platforms.

For more information on the Job Accounting Key, see “Correlating AIX and LoadLeveler accounting records” on page 46

**Job Name**
The name of the job.

**Job Step ID**
The job step identifier.

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

**LoadLeveler Group**
The LoadLeveler group associated with the job step.

**Machine Speed**
For a serial job step, the value associated with the **speed** keyword of the machine that is running this job step. For a parallel job step, the value associated with the **speed** keyword of the first machine that has been allocated for this job step.

**Max Processors**
The maximum number of processors that can be used for this job step.

**McmAffinityOptions**
The MCM affinity options for the job.

**MetaCluster JobId**
The ID assigned by LoadLeveler to the job step for use in all MetaCluster operations on that job step. This ID corresponds to the **MDCR_JOBID** environment variable used by MetaCluster.

**Min Processors**
The minimum number of processors needed for this job step.

**Monitor Program**
Specifies the monitor program used for the job.

**Network Usage**

- Network usages information has the format:
  ```
  Network Usages: Network Usage, ... , Network Usage
  ```
- Network usage has the format:
  ```
  Network Id (Instances, protocol, mode, Number of Windows, \
  adapter window memory, imm send buffers,collective groups)
  ```
  where:

  **Network Id = 0**
  Specifies that all switch networks are used.

  **Network Id = -1**
  Specifies that the Ethernet network is used.

  **Network Id > 0**
  Specifies the network ID that is used.

**(Node) Allocated Hosts**

- The machines of this Node type that have been allocated for this job step. The format is:

```
hostname:task status: cpu usage, ... ,cpu usage + ... +
```

- The CPU usage information has one of the following formats:

  ```
  CPU <cpulist>;
  MCMnumber:CPU <cpulist>;
  ```

  The *cpulist* is a blank-delimited list of individual CPU IDs or CPU ranges, or a combination of both CPU IDs and CPU ranges. The CPU range is specified as the starting CPU ID and the ending CPU ID separated by a hyphen (-).

**(Node) Name**
Blank value. Reserved for future use.

**(Node) Node actual**
Actual number of machines of this Node type that are used in the running of this job step.

**(Node) Node maximum**
Maximum number of machines of this Node type that can be used to run this job step.

**(Node) Node minimum**
Minimum number of machines of this Node type required to run this job step.

**(Node) Preferences**
Job step preferences as specified at job submission.

**(Node) Requirements**
Job step requirements as specified at job submission.

**(Node/Master Task) Exec Args**
The arguments passed to the master task executable.

**(Node/Master Task) Executable**
The executable associated with the master task.

**(Node/Master Task) Num Task Inst**
The number of task instances of the master task.

**(Node/Master Task) Task Instance**
Task instance information has the format:

```
hostname: task ID
```

**(Node/Task) Num Task Inst**
The number of task instances.

**(Node/Task) Task Instance**

- Task instance information has the format:

  ```
  hostname: task ID: cpu usage
  ```

- The CPU usage information has one of the following formats:

  ```
  CPU <cpulist>;
  MCMnumber:CPU <cpulist>;
  ```

  The *cpulist* is a blank-delimited list of individual CPU IDs or CPU ranges, or a combination of both CPU IDs and CPU ranges. The CPU range is specified as the starting CPU ID and the ending CPU ID separated by a hyphen (-).

**Node Resources**
: Reflects the settings for the **node_resources** keyword in the job command file.

**Node Usage**
: A request that a node be shared or not shared or that a time-slice is not shared. The user specifies this request while submitting the job.

**Notifications**
: The notification status for the job step, where:

  **`always`**
  : Indicates notification is sent through the mail for the complete, error, never, and start notification categories.

  **`complete`**
  : Indicates notification is sent through the mail only when the job step completes.

  **`error`**
  : Indicates notification is sent through the mail only when the job step terminates abnormally.

  **`never`**
  : Indicates notification is never sent.

  **`start`**
  : Indicates notification is sent through the mail only when starting or restarting the job step.

**Notify User**
: The user to be notified by mail of a job's status.

**Out**
: The file to be used for stdout.

**Owner**
: The user ID that the job will be run under.

**parallel_threads**
: Indicates whether the job requests thread binding and the number of parallel threads of an OpenMP job.

**Perf. degradation**
: The performance degradation in percent.

**Port Number**
: The port number for InfiniBand resources used by the running job.

**Preempt Wait Count**
: Specifies the number of job steps that an idle job step must preempt before it can be started.

**Preemptable**
: Indicates whether a job step is preemptable (yes or no).

**Preferences**
: Job step preferences as specified at job submission.

**Queue Date**
: The date and time that LoadLeveler received the job.

**Req. Energy Savings**
: The requested energy saving percentage in the job command file.

**Requirements**
Job step requirements as specified at job submission.

**Resource**
The resource being enforced by WLM. This is either **CPU**, **Real Memory**, **Virtual Memory**, or **Large Page Memory**.

**Resources**
Reflects the settings for the **resources** keyword in the job command file.

**Restart**
Restart status (**yes** or **no**).

**Restart From Ckpt**
Indicates if a job has been restarted from an existing checkpoint (**yes** or **no**).

**Restart Same Nodes**
Indicates if a job step should be restarted on the same nodes after vacate (**yes** or **no**).

**RSet** The RSet requirement of the job.

**Rss Hard Limit**
RSS hard limit as specified at job step submission.

**Rss Soft Limit**
RSS soft limit as specified at job step submission.

**Running Host**
For a serial job step, the machine that is running this job step. For a parallel job step, the first machine that has been allocated for this job step.

**Scheduler ID**
LoadLeveler scheduler ID.

**Sending Cluster**
The cluster name that the job was sent from when moved. This field only displays multicluster-specific information.

**Shell** The shell to be used when the job step runs.

**SMT requested**
Indicates the required simultaneous multithreading (SMT) state, which is defined in the job command file, if the job step requires SMT to be turned on or off. Valid values are **yes** or **no**.

**SMT required**
Indicates the required simultaneous multithreading (SMT) state, if the job step requires SMT to be enabled, disabled, or kept as is. Valid values are **yes**, **no**, or **as_is**.

**Stack Hard Limit**
Stack hard limit as specified at job submission.

**Stack Soft Limit**
Stack soft limit as specified at job submission.

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

**Starter inblock/Step inblock**
Number of times file system performed input. Cumulative total.

**Starter isrss/Step isrss**
Depending on the Operating System, this field may contain the integral value of unshared stack size.

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

**Starter majflt/Step majflt**
Number of page faults (I/O required). Cumulative total.

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

**Starter minflt/Step minflt**
Number of page faults (reclaimed). Cumulative total.

**Starter msgrcv/Step msgrcv**
Number of IPC messages received. Cumulative total.

**Starter msgsnd/Step msgsnd**
Number of IPC messages sent. Cumulative total.

**Starter nivcsw/Step nivcsw**
Number of involuntary context switches. Cumulative total.

**Starter nsignals/Step nsignals**
Number of signals delivered. Cumulative total.

**Starter nswap/Step nswap**
Number of times swapped out. Cumulative total.

**Starter nvcsw/Step nvcsw**
Number of context switches due to voluntarily giving up processor. Cumulative total.

**Starter oublock/Step oublock**
Number of times file system performed output. Cumulative total.

**Starter System Time/Step System Time**
CPU system time of Starter/Step processes. Cumulative total.

**Starter Total Time/Step Total Time**
CPU total time of Starter/Step processes. Cumulative total.

**Starter User Time/Step User Time**
CPU user time of Starter/Step processes. Cumulative total.

**Status** The status (state) of the job. For more information, see "LoadLeveler job states" on page 15.

**Step Adapter Memory**
The total adapter pinned memory for the job step.

**Step Cpu Hard Limit**
Job step CPU hard limit as specified at job submission.

**Step Cpu Soft Limit**
Job step CPU soft limit as specified at job submission.

**Step Cpus**
The total ConsumableCpus for the job step.

**Step Large Page Memory**
The total ConsumableLargePageMemory for the job step.

**Step Name**
The name of the job step.

**Step rCxt Blocks**
The number of rCxt blocks for switch adapters.

**Step Real Memory**
The total ConsumableMemory for the job step.

**Step Type**
Type of job step:
- Serial
- General parallel
- Blue Gene
- MPICH parallel

**Step Virtual Memory**
The total ConsumableVirtualMemory for the job step.

**Structure Version**
An internal version identifier.

**Submitting Host**
The name of the machine to which the job is submitted.

**Task Affinity**
The task affinity requirements of the job.

**Task_geometry**
Reflects the settings for the **task_geometry** keyword in the job command file.

**Topology Requirement**
The island topology requested in the **node_topology** keyword.

**total** Total CPU time consumed in milliseconds. CPU resource only.

**Trace** Indicates if the job life cycle is being traced (yes or no).

**Unix Group**
The effective UNIX group name.

**User Space Windows**
The number of switch adapter windows assigned to the job step.

**Virtual Image Size**
The value of the **image_size** keyword (if specified) or the size of the executable associated with the **executable** keyword (if specified) or the size of the job command file.

**Wall Clk Hard Limit**
Wall clock hard limit as specified at job submission.

**Wall Clk Soft Limit**
Wall clock soft limit as specified at job submission.

## Examples

1. This example generates the standard listing where the cluster has two jobs running and one job waiting:

```
Id                       Owner     Submitted   ST Class      Running On
--------------------     --------- ----------- -- ---------  ----------
mars12.304.0@mars12      rose      11/7  11:41 R  No_Class   mars09
mars11.26.0@mars11       bill      11/8  02:34 R  high       mars01
```

```
mars11.27.0@mars04       zach      11/8  07:12 I  low

3 job step(s) in queue, 1 waiting, 0 pending, 2 running, 0 held,
0 preempted
```

The standard listing includes the following fields:

**Class** Job class.

**Id** The resource manager job step ID. Due to space limitations, the job step ID may be truncated. If the job step ID is truncated, a dash (-) will appear at the end to indicate that characters have been left out. To see the full resource manager job step ID, run **llrq** with the **-l** flag.

**Owner**
User ID that the job will be run under.

**Running On**
If running, the name of the machine the job step is running on. This is blank when the job is not running. For a parallel job step, only the first machine is shown.

**ST** Current state of the job step. For more information, see the "LoadLeveler job states" topic in *LoadLeveler: Using and Administering*.

**Submitted**
Date and time of job submission.

2. This example generates the long listing. The long listing is generated when you specify the **-l** flag with the **llrq** command:

   ```
   llrq -l
   ```

3. Using the abbreviated form of *jobid*, this example generates a standard listing for all job steps with `jobid` 12 assigned by the local machine:

   ```
   llrq 12
   ```

4. This example generates a standard listing for all job steps owned by either rich or nathan and bound to reservation 6:

   ```
   llrq -u rich nathan -R 6
   ```

5. The following example generates a customized listing for all job steps:

   ```
   llrq -f %id %o %R
   ```

   You should receive a response similar to the following:

   ```
   Step Id                  Owner       Reservation ID
   ------------------------ ----------- ------------------------
   c94n04.5.0               zhong       c94n04.2.r
   c94n04.4.0               zhong

   2 job step(s) in queue, 1 waiting, 0 pending, 0 running, 1 held,
                           0 preempted
   ```

6. This example generates a customized and formatted standard listing:

   ```
   llrq -f %id %dq %dd %gl %h
   ```

   You should receive output similar to the following:

   ```
   Step Id        Queue Date  Disp. Date  LL Group   Running On
   -------------- ----------- ----------- ---------- ---------------
   l16.2.0        04/08 09:19 04/08 09:21 No_Group   l16.pok.ibm.com
   l16.1.0        04/08 09:19 04/08 09:21 No_Group   l16.pok.ibm.com
   ```

```
ll6.3.0        04/08 09:19 04/08 09:21 No_Group   ll5.pok.ibm.com

3 job step(s) in queue, 0 waiting, 0 pending, 3 running, 0 held, 0 preempted
```

7. This example generates a customized, unformatted (raw) standard listing. Output fields are separated by an exclamation point (!).

   ```
   llrq -r %id %dq %dd %gl %h
   ```

   You should receive output similar to the following:

   ```
   ll6.pok.ibm.com.2.0!01/16/2009 09:19!01/16/2009 09:21!
                       No_Group!ll6.pok.ibm.com
   ll6.pok.ibm.com.1.0!01/16/2009 09:19!01/16/2009 09:21!
                       No_Group!ll6.pok.ibm.com
   ll6.pok.ibm.com.3.0!10/16/2009 09:19!01/16/2009 09:21!
                       No_Group!ll5.pok.ibm.com
   ```

8. This example generates a WLM CPU and memory statistics listing where c704f5sq06.55.0 is a parallel job step currently running on two nodes c704f5sq05 and c704f5sq06. If consumable resource enforcement is enabled, the **-w** option can be used to obtain CPU and memory statistics of job steps in the running state.

   ```
   llrq -w c704f5sq06.55.0
   ```

   You should receive output similar to the following:

   ```
   ====== c704f5sq06.ppd.pok.ibm.com.55.0@c704f5sq06.ppd.pok.ibm.com ======
         c704f5sq05.ppd.pok.ibm.com.55.0@c704f5sq06.ppd.pok.ibm.com:
                 Resource: CPU
                         snapshot: 13
                         total: 14996
                 Resource: Real Memory
                         snapshot: 0
                         high water: 718
                 Resource: Virtual Memory
                         snapshot: 98
                         high water: 25284
                 Resource: Large Page Memory
                         snapshot: 96

         c704f5sq06.ppd.pok.ibm.com.55.0@c704f5sq06.ppd.pok.ibm.com:
                 Resource: CPU
                         snapshot: 13
                         total: 14549
                 Resource: Real Memory
                         snapshot: 0
                         high water: 382
                 Resource: Virtual Memory
                         snapshot: 81
                         high water: 29109
                 Resource: Large Page Memory
                         snapshot: 80
   ```

   The standard listing includes the following fields:

   **high water**
   : The highest number of memory pages used. **Real Memory** and **Virtual Memory** resources only.

   **snapshot**
   : For **CPU** and **Real Memory** consumption, it is the current percentage of the total resources available. For **Virtual Memory** and **Large Page Memory**, it is the current usage in megabytes.

   The following statistics are displayed for every node the job is running on:

- The current CPU resource consumption as a percentage of the total resources available
- The total CPU time consumed in milliseconds
- The current real memory consumption as a percentage of the total resources available
- The highest number of real memory pages used
- The current virtual memory usage in megabytes
- The highest number of virtual memory pages used
- The current large page memory usage in megabytes

9. The following is sample **llrq -l** output for task instances and allocated hosts if the job requested MCM affinity:

```
Allocated Hosts: e189f4rp04.ppd.pok.ibm.com::
                      MCM0:CPU< 0-5 >;,MCM0:CPU< 0-5 >;
                 + e189f4rp03.ppd.pok.ibm.com::
                      MCM1:CPU< 4-5 >;,MCM1:CPU< 4-5 >;

Num Task Inst: 4
      Task Instance: e189f4rp04:0:MCM0:CPU< 0-5 >;
      Task Instance: e189f4rp04:1:MCM0:CPU< 0-5 >;
      Task Instance: e189f4rp03:2:MCM1:CPU< 4-5 >
      Task Instance: e189f4rp03:3:MCM1:CPU< 4-5 >
```

   Note that the < > notation will be used to list individual CPU IDs instead of the CPU count ( ) notation when the **RSET_SUPPORT** configuration file keyword is set to **RSET_MCM_AFFINITY**.

10. The following example shows the Resource Set information in the **llrq -l** listing when the consumable CPUs Resource Set requirement is requested:

```
Allocated Hosts : e189f4rp01.ppd.pok.ibm.com::
                      CPU< 0-5 >,CPU< 0-5>
                  + e189f4rp02.ppd.pok.ibm.com::
                      CPU< 0-5 >,CPU< 0-5 >

Num Task Inst: 4
      Task Instance: e189f4rp01:0:CPU< 0-5 >
      Task Instance: e189f4rp01:1:CPU< 0-5 >
      Task Instance: e189f4rp02:2:CPU< 0-5 >
      Task Instance: e189f4rp02:3:CPU< 0-5 >
```

11. The following example shows output for the **llrq -l** command when SMT is requested:

```
===== Job Step blablahome.clusters.com.22.0@blablahome.clusters.com ====
   Job Step Id: blablahome.clusters.com.22.0@blablahome.clusters.com
      Job Name: blablahome.clusters.com.22
.
.
.
        Status: Running
.
.
.
    Large Page: N
    Coschedule: no
 SMT requested: yes
Checkpointable: no
.
.
.
```

12. This following example lists the network usages assigned to the job step in addition to other resources of the job:

```
llrq -l
```

You should receive a response similar to the following:

```
==== Job Step c890f1ec05.ppd.pok.ibm.com.18.0@blablahome.clusters.com ===
Job Step Id: c890f1ec05.ppd.pok.ibm.com.18.0@blablahome.clusters.com
..................
Network Usages: 0(1, LAPI, US, 1, 0 rCxt Blks),
                18338657682652659715(1, MPI, US, 1, 0 rCxt Blks),
-----------------------------------------------------------------------

Network Usages: 18446744073709551615(1,mpi,IP,0,0,1,0),

Scheduler ID: _LoadLevler_scheduler_ID_
Monitor Program: /home/loadl/jobs/my_notification_program
------------------------------------------------------------------------
Node
----

Name            :
Requirements    :
Preferences     :
Node minimum    : 1
Node maximum    : 1
Node actual     : 1
Allocated Hosts : c890f1ec06.ppd.pok.ibm.com:RUNNING:MCM1:CPU<0>, \
                  MCM1:CPU<1>
Master Task
-----------

Executable   : /usr/bin/poe
Exec Args    : /tmp/llbld/bin/btat -t 1 -d 90 -m 100 -v -ilevel 6 \
               -labelio yes -pmdlog yes
Num Task Inst: 1
Task Instance: c890f1ec06:-1

Task
----

Num Task Inst: 2
Task Instance: c890f1ec06:0:MCM1:CPU< 0 >
Task Instance: c890f1ec06:1:MCM1:CPU< 1 >

Node
----

Name           :
Requirements   :
Preferences    :
Node minimum   : 1
Node maximum   : 1
Node actual    : 1
Allocated Hosts: c890f1ec07.ppd.pok.ibm.com:RUNNING:MCM0:CPU< 0 >

Task
----

Num Task Inst: 1
Task Instance: c890f1ec07:2:MCM0:CPU< 0 >
..................

-----------------------------------------------------------------------

Job Step Status on Allocated Hosts
----------------------------------
```

```
c890f1ec06.ppd.pok.ibm.com:RUNNING
c890f1ec07.ppd.pok.ibm.com:RUNNING
c890f1ec08.ppd.pok.ibm.com:RUNNING

1 job step(s) in query, 0 waiting, 0 pending, 1 running, 0 held,
0 preempted
```

13. The following example shows a high-level summary of the state of all jobs in the queue:

```
llrq -S total
```

You should receive a response similar to the following:

```
13 job step(s) in queue, 11 waiting, 0 pending, 1 running, 1 held, 0 preempted
```

14. The following example shows a high-level summary of the jobs of each user:

```
llrq -S user
```

You should receive a response similar to the following:

```
loadl: 6 job steps in queue, 5 waiting, 0 pending, 1 running, 0 held, 0 preempted
llbld: 7 job steps in queue, 6 waiting, 0 pending, 0 running, 1 held, 0 preempted
Total: 13 job steps in queue, 11 waiting, 0 pending, 1 running, 1 held, 0 preempted
```

15. The following example shows a medium listing for a single job step:

```
llrq -m -j c890f14ec03.ppd.pok.ibm.com.147
```

You should receive a response similar to the following:

```
===== Job Step c890f14ec03.ppd.pok.ibm.com.147.0 =====
        Job Step Id: c890f14ec03.ppd.pok.ibm.com.147.0
           Job Name: c890f14ec03.ppd.pok.ibm.com.147
          Step Name: 0
  Structure Version: 10
              Owner: nathan
         Queue Date: Mon Dec 13 11:39:13 2010
             Status: Running
     Reservation ID:
  Requested Res. ID:
   Flexible Res. ID:
          Recurring: False
 Scheduling Cluster:
 Submitting Cluster:
    Sending Cluster:
  Requested Cluster:
     Schedd History:
   Outbound Schedds:
    Submitting User:
   Eligibility Time: Mon Dec 13 11:39:14 2010
      Dispatch Time: Mon Dec 13 11:39:14 2010
    Completion Date:
     Completion Code:
        Favored Job: No
      User Priority: 50
       user_sysprio: 0
      class_sysprio: 0
      group_sysprio: 0
    System Priority: -21
          q_sysprio: -21
 Previous q_sysprio: 0
      Notifications: Never
 Virtual Image Size: 2.032 mb
         Large Page: N
              Trace: no
         Coschedule: no
       SMT required: as_is
    MetaCluster Job: no
     Checkpointable: no
    Ckpt Start Time:
Good Ckpt Time/Date:
   Ckpt Elapse Time: 0 seconds
Fail Ckpt Time/Date:
    Ckpt Accum Time: 0 seconds
    Checkpoint File:
   Ckpt Execute Dir:
  Restart From Ckpt: no
 Restart Same Nodes: no
            Restart: yes
        Preemptable: yes
 Preempt Wait Count: 0
     Hold Job Until:
     User Hold Time: 00:00:00 (0 seconds)
               RSet: RSET_MCM_AFFINITY
Mcm Affinity Option: MCM_ACCUMULATE MCM_MEM_PREF MCM_SNI_NONE
      Task Affinity: core(1)
      Cpus Per Core:  0
   Parallel Threads:  0
                Env:
                 In: /dev/null
                Out: /dev/null
                Err: /dev/null
Initial Working Dir: /u/nathan/test/5u1perf
         Dependency:
Data Stg Dependency:
          Step Type: General Parallel
         Node Usage: not_shared
    Submitting Host: c890f14ec03.ppd.pok.ibm.com
        Schedd Host: c890f14ec03.ppd.pok.ibm.com
      Job Queue Key:
        Notify User: nathan@c890f14ec03.ppd.pok.ibm.com
              Shell: /bin/ksh
  LoadLeveler Group: No_Group
              Class: No_Class
    Ckpt Hard Limit: undefined
    Ckpt Soft Limit: undefined
     Cpu Hard Limit: undefined
     Cpu Soft Limit: undefined
    Data Hard Limit: undefined
    Data Soft Limit: undefined
      As Hard Limit: undefined
      As Soft Limit: undefined
   Nproc Hard Limit: undefined
   Nproc Soft Limit: undefined
 Memlock Hard Limit: undefined
 Memlock Soft Limit: undefined
   Locks Hard Limit: undefined
   Locks Soft Limit: undefined
  Nofile Hard Limit: undefined
  Nofile Soft Limit: undefined
    Core Hard Limit: undefined
    Core Soft Limit: undefined
    File Hard Limit: undefined
    File Soft Limit: undefined
   Stack Hard Limit: undefined
   Stack Soft Limit: undefined
     Rss Hard Limit: undefined
     Rss Soft Limit: undefined
Step Cpu Hard Limit: undefined
Step Cpu Soft Limit: undefined
Wall Clk Hard Limit: 01:00:00 (3600 seconds)
Wall Clk Soft Limit: 00:00:30 (30 seconds)
            Comment:
            Account:
         Unix Group: usr
Negotiator Messages:
      Bulk Transfer: No
Adapter Requirement: (eth1,mpi,IP,shared,AVERAGE,instances=1,imm_send_buffers=0,collective_groups=0)
          Step Cpus: 0
Step Virtual Memory: 0.000 mb
   Step Real Memory: 0.000 mb
Step Large Page Mem: 0.000 mb
Topology Requirement: island max=all, min=all
      Network Usages: 18446744073709551615(1,mpi,IP,0,0,0,0,1),

        Scheduler ID: _LoadLevler_scheduler_ID_
     Monitor Program:

1 job step(s) in queue, 0 waiting, 0 pending, 1 running, 0 held, 0 preempted
```

## Security

LoadLeveler administrators and users can issue this command.

## llrqetag - Query information about energy policy tags

Use the **llrqetag** command to display energy policy tags and the information associated with them.

### Syntax

**llrqetag** [**-?** | **-H** | **-v** | [**-u** *user*] [**-j** *job*] [**-e** *energy_tag*]]

### Flags

**-?** Provides a short usage message.

**-H** Provides extended help information.

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

**-u** *user*
Displays energy policies that belong to this LoadLeveler user.

**-j** *job*
Is a job or step identifier. When a job identifier is specified, the command action is taken for all steps of the job.

The format of a job identifier is *host.jobid*. The format of a step identifier is *host.jobid.stepid*.

where:

- *host* is the name of the machine that assigned the job and step identifiers.
- *jobid* is the job number assigned to the job when it was submitted.
- *stepid* is the job step number assigned to the job step when it was submitted. The job or step identifier can be specified in an abbreviated form, *jobid* or *jobid.stepid*, when the command is invoked on the same machine that assigned the job and step identifiers. In this case, LoadLeveler will use the local machine's hostname to construct the full job or step identifier.

**-e** *energy_tag*
Is an energy tag name of the form [*user*.]*tag_name*

where:

- *user* is the user name who generated the energy tag.
- *tag_name* is the user-defined identifier in the job command file.

### Description

The **llrqetag** command queries information about the energy policies that is saved in the database. When you issue this command, it displays all the energy policies in the database that have been generated by the user. For the **root** user, the command will display all the tag names in the database.

The policy tag is a user-defined identifier of the job energy policy, which is used to identify the estimated power consumed, the estimated execution time of the job, and the job performance degradation. The energy tag name is specified in the job command file and when you submit a job, the energy tag name must be unique for each of your jobs.

This command is available only when the database-based configuration is used by LoadLeveler.

## Examples

1. To display the energy policy information for energy policy tag **long_running_job**, issue:

   ```
   llrqetag –e long_running_job
   ```

   You should receive a response similar to the following:

   ```
             Tag Name: long_running_job
         Generated by: c111bc4n13.ppd.pok.ibm.com.91.0
       Last used Time: Tue May 10 05:00:01 EDT 2011
                 User: userA
    Nominal Frequency: 3.3 GHZ
    Node's Energy Use: 0.8 Kwh
       Execution Time: 950 Seconds
   Frequency(GHZ) EstEnergyConsum(Kwh) EngSaving(%) EstTime(Seconds) PerfDeg(%)
             3.10                 0.75          6.2             1000        5.2
             2.80                 0.65         18.8             1050       10.5
             2.60                 0.6          25.0             1100       15.8
             2.20                 0.5          37.5             1150       21.1
   ```

   where:

   **Tag Name**
   Is the energy tag name.

   **Generated by**
   Is the job step identifier that generated the tag.

   **Last used Time**
   Is the time when the tag was last used.

   **User:** Is the user who generated the tag.

   **Nominal Frequency**
   Is the CPU nominal frequency.

   **Node's Energy Use**
   Is the energy consumption per node of the job step at the nominal frequency.

   **Execution Time**
   Is the execution time of the job step at the nominal frequency.

   **Frequency (GHZ)**
   Is the CPU frequency.

   **EstEnergyConsum (KWH)**
   Is the estimated energy consumption of the job step at this frequency.

   **EngSaving (%)**
   Is the percentage of energy savings at this frequency.

   **EstTime (Seconds)**
   Is the estimated run time of the job step at this frequency.

   **PerfDeg (%)**
   Is the percentage of the performance degradation at this frequency.

2. To display all energy policy information generated by userA, issue:

   ```
   llrqetag -u userA
   ```

   You should receive a response similar to the following:

   ```
             Tag Name: long_running_job
         Generated by: c111bc4n13.ppd.pok.ibm.com.91.0
       Last used Time: Tue May 10 05:00:01 EDT 2011
                 User: userA
   ```

```
 Nominal Frequency: 3.3 GHZ
 Node's Energy Use: 0.8 Kwh
    Execution Time: 950 Seconds
Frequency(GHZ) EstEnergyConsum(KWH) EngSaving(%) EstTime(Seconds) PerfDeg(%)
          3.10                 0.75          6.0             1000        5.0
          2.80                 0.65         19.0             1050       11.0
```

**Note:** The **llrqetag** command will display the energy policies generated by the user who issues the command if the **-u** option is not specified.

## Security

LoadLeveler administrators and users can issue this command.

## llrrmetag - Remove the energy policy

Use the **llrrmetag** command to remove energy policies from the database.

### Syntax

| **llrrmetag** [ **-?** | **-H** | **-v** | **-a** [**-u** *user*] [**-j** *job*] [**-t** *MM*/*DD*/[*YY*]*YY*] [**-e** *energy_tag*]]

### Flags

**-?** Provides a short usage message.

**-H** Provides extended help information.

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

| **-a** Removes all energy policies that were created by the current user.

**-u** *user*
Removes all energy policies that were created by the specified LoadLeveler user.

**-j** *job*
Removes energy policies that were created by the specified LoadLeveler job or job step.

The format of a job identifier is *host.jobid*. The format of a step identifier is *host.jobid.stepid*.

where:

- *host* is the name of the machine that assigned the job and step identifiers.
- *jobid* is the job number assigned to the job when it was submitted.
- *stepid* is the job step number assigned to the job step when it was submitted. The job or step identifier can be specified in an abbreviated form, *jobid* or *jobid.stepid*, when the command is invoked on the same machine that assigned the job and step identifiers. In this case, LoadLeveler will use the local machine's hostname to construct the full job or step identifier.

**-t** *MM*/*DD*/[*YY*]*YY*
Removes the energy tag if it has not been used since the time specified in this option.

where:

*MM* Is the month.

*DD* Is the day.

*YYYY*
Is the year.

**Note:** A 2-digit year can also be used with this option (*MM/DD/YY*).

**-e** *energy_tag*
Is an energy tag name of the form [*user*.]*tag_name*.

where:

- *user* is the user name who generated the energy tag.
- *tag_name* is the user-defined identifier in the job command file.

## Description

The **llrrmetag** command removes energy policies from the database. You can only remove your own energy policies, but the system administrator can run this command for any energy policy.

This command is available only when the database-based configuration is used by LoadLeveler.

## Examples

1. To remove all energy policy tags generated by the user ID tom, issue:

   ```
   llrrmetag –u tom
   ```

   You should receive a response similar to the following:

   ```
   llrrmetag: User loadl removed 3 energy policy tags.
   ```

   When this command is issued by user ID tom, `llrrmetag` is all that needs to be issued.
2. For the administrator to remove the energy policy with the tag long_running_job submitted by user ID tom, issue:

   ```
   llrrmetag -e tom.long_running_job
   ```

   You should receive a response similar to the following:

   ```
   llrrmetag: User loadl removed 1 energy policy tags.
   ```

## Security

LoadLeveler administrators and users can issue this command.

## llrstatus - Query machine information

Use the **llrstatus** command to return status information about machines in the LoadLeveler cluster.

### Syntax

**llrstatus**

[**-?**] [**-H**] [**-v**] [**-W**] [**-R**] [**-M**] [**-l**] [**-a**] [**-d**] [**-e**]
[**-f** *category_list*] [**-r** *category_list*]
[**-L** {**cluster** | **machine_group** | **machine**}]
[**-h** *hostlist* | *hostlist*]

### Flags

**-?** Provides a short usage message.

**-H** Provides extended help information.

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

**-W** Specifies that the width of columns in tabular output will be increased to fit the widest entry.

**-R** Lists the machine consumable resources associated with each machine for which status is requested. This option should not be used with any other option.

**-M** Lists the Multiple Chip Modules (MCMs) available on a machine, where:

**Available CPUs**
Are the CPU IDs of the CPUs available for LoadLeveler on this MCM.

**Adapters**
Are the switch adapters connected to this MCM of the form:

```
[available windows, available rCxtBlocks]
```

where:

**available windows**
Is the total number of windows on the adapter.

**available rCxt blocks**
Is the total number of rCxt blocks on the adapter.

Note that the < > notation will be used to list individual CPU IDs instead of the CPU count ( ) notation when the **RSET_SUPPORT** configuration file keyword is set to **RSET_MCM_AFFINITY**.

**-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.

**-a** Displays information for each virtual adapter followed by information for each physical adapter it manages. This flag also displays the port number on each InfiniBand adapter port.

**-d** Displays a list of regions and its corresponding active region manager.

**-e** Displays the energy coefficients information.

**-f** *category_list*
Is a blank-delimited list of categories you want to query. Each specified category must be preceded by a percent sign. The *category_list* cannot contain duplicate entries. This flag is used to create a customized version of the standard **llrstatus** listing. The output fields produced by this flag all have a

fixed length. The output is displayed in the order in which the categories are specified. *category_list* can be one or more of the following:

| | |
|---|---|
| **%a** | Hardware architecture |
| **%act** | Number of job steps dispatched by the resource manager daemon on this machine |
| **%cm** | Custom Metric value |
| **%cpu** | Number of CPUs on this machine |
| **%d** | Available disk space in the LoadLeveler execute directory |
| **%i** | Number of seconds since last keyboard or mouse activity |
| **%inq** | Number of job steps in the job queue of this job manager machine |
| **%l** | Berkeley one-minute load average |
| **%m** | Physical memory on this machine |
| **%n** | Machine name |
| **%o** | Operating system on this machine |
| **%sca** | Availability of the job manager daemon |
| **%scs** | State of the job manager daemon |
| **%sta** | Availability of the startd daemon |
| **%sts** | State of the startd daemon |
| **%v** | Available swap space (free paging space) of this machine |

**-r** *category_list*
Is a blank-delimited list of categories to query. Each specified category must be preceded by a percent sign. The *category_list* cannot contain duplicate entries. This flag is used to create a customized version of the standard **llrstatus** listing. The output produced by this flag is considered raw, in that the fields can be variable in length. Output fields are separated by an exclamation point (!). *category_list* can be one or more of the categories listed under the **-f** flag.

**-L {cluster | machine_group | machine}**
Selects the level of detail to be displayed, where:

**cluster**
Displays a brief, high-level summary of the state of the machines in the cluster. When combined with the **-l** option, the listing is expanded to show which machines are currently in an unavailable state. For machines configured to run Startd, available is defined as Idle, Running, or Busy, and unavailable is defined as Down, Draining, Drained, Flush or Suspended. For machines configured to run Schedd, available is defined as Avail and unavailable is defined as Down or Drained.

**machine_group**
Displays information on a machine group basis. The standard listing displays one line for each machine group containing a high-level summary of the state of the machines within that group. When the **-l** option is used, a long listing is displayed, showing a list of the machines belonging to the group along with detailed configuration and status information.

**machine**
Displays information on a machine basis. The standard listing displays one line for each machine. When the **-l** option is used, a long listing is displayed.

**-h** *hostlist*
*hostlist*
Is a blank-delimited list of machines and machine groups for which status is requested.

**Note:** If a configuration has a machine group and a machine sharing the same name, the **llrstatus** command will always assume that the name refers to the machine group.

If the **-X** flag is specified in combination with a *hostlist*, the **-h** flag must be specified. For example:

```
llrstatus -X my_cluster1 my_cluster2 -h c94n13 c94n14
```

## Description

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 resource manager with a snapshot of its situation on an interval calculated by **POLLING_FREQUENCY * POLLS_PER_UPDATE**. Since the information returned by using **llrstatus** is a collection of such snapshots, all taken at varying times, the total picture may not be completely consistent.

Certain resources such as remote direct-memory access (RDMA) have their available value always calculated by startd. Available **ConsumableCpus** resources are calculated by startd if the value is specified as **all** in the administration file. When the value of a resource is calculated by startd, the **llrstatus** command appends a plus (+) sign to the resource name in the output reports. Resources that are automatically created, such as RDMA, have a less than (<) sign appended to them.

**Notes on using the -L option:**

1. The **-L** option only applies to the standard listing (no other option specified) and the long listing (**-l**).
2. The **-f** and **-r** options are only supported with **-L machine**. Using these options with **-L cluster** or **-L machine_group** will result in an error.
3. The **-L** option cannot be used in combination with **-a**, **-e**, **-M**, or **-R**, or an error will result.

**Notes on using the hostlist option:**

1. For a machine-level option (**-L machine** or any existing **llrstatus** options such as **-a**, **-M**, or **-R**), a machine group in the *hostlist* will expand to include all of the machines defined in that machine group.
2. For the **machine_group-level** option (**-L machine_group**), the *hostlist* will be used to select which machine groups will be displayed. Specifying a machine group name in the *hostlist* will result in that machine group being displayed. Specifying a machine name in the host list will result in the machine group to which that machine belongs being displayed. If a machine name is specified, and that machine does not belong to a machine group, then the row OTHERS will be displayed. The OTHERS row contains a collection of those machines that are not defined as part of a machine group, and in the case where a *hostlist* is specified, only the requested machines are included.

3. For the **cluster-level** option (**-L cluster**), the *hostlist* is ignored.

**Default behavior of the llrstatus command**

The default output of the **llrstatus** command can be changed using the **LOADL_STATUS_LEVEL** environment variable. Its value can be one of the following:

**CLUSTER**
When **LOADL_STATUS_LEVEL=CLUSTER**, specifying the **llrstatus** command without the **-L** option is equivalent to specifying **llrstatus -L** cluster. Cluster level output is the default for the **llrstatus** command.

**MACHINE_GROUP**
When **LOADL_STATUS_LEVEL=MACHINE_GROUP**, specifying the **llrstatus** command without the **-L** option is equivalent to specifying **llrstatus -L machine_group**.

**MACHINE**
When **LOADL_STATUS_LEVEL=MACHINE**, specifying the **llrstatus** command without the **-L** option is equivalent to specifying **llrstatus -L machine.**

**Notes:**
1. If the **LOADL_STATUS_LEVEL** environment variable is not set to **CLUSTER**, **MACHINE_GROUP**, or **MACHINE**, the **llrstatus** command will display the cluster level information by default.
2. The **llrstatus** command with the **-R**, **-M**, **-a**, or **-e** options will ignore the value set for the **LOADL_STATUS_LEVEL** environment variable.

## Examples

1. This example generates the cluster-level standard listing. The standard listing is generated when you do *not* specify the **-l** option with the **llrstatus** command. Alternatively, set the environment variable **LOADL_STATUS_LEVEL=cluster** and issue:

   ```
   llrstatus
   ```

   Or, issue:

   ```
   llrstatus -L cluster
   ```

   You should receive output similar to the following:

   ```
   Active           1/2
   Schedd           1/1                 0 job steps
   Startd           1/2                 0 running tasks
   RegionMgr        1/1

   Total Computing nodes in Standby (S3) state    0 machines

   The Resource Manager is defined on c596n11.ppd.pok.ibm.com


   Absent:            1
   Startd:         Down    Suspend
                     0          0
   Schedd:         Down    Drained   Draining
                     0          0          0
   RegionMgr:      Down  Alternate
                     0          0
   ```

This output indicates that there is 1 startd down, no startd drained, no startd draining, no startd flush, 3 startds suspend, no Schedd drained, no Schedd draining, 1 Schedd down and 90 machines absent. The last four lines show a brief summary, which indicates there are totally 16,300 machines defined in the administration file or database, and 16,210 machines have reported their status to the Resource Manager daemon. Out of 96 machines known by the Central Manager to be Schedd machines, 95 are up and able to accept jobs. Among these Schedd machines there 2,487 queued job steps in various states. Out of 16,200 machines known to be Startd machines, 16,196 are up and either already running jobs or able to run jobs. Among these 16,196 Startd machines there are a total of 517,832 running tasks belonging to various jobs. And finally, the Central Manager is identified.

**Note:** The totals for RegionMgr indicate the number of Region Managers serving out of the total number of regions defined. This count may include alternate Region Managers, which have taken over after a primary Region Manager has become unavailable. This is broken out below, where "Down" is a count of the primary Region Managers which are down and "Alternate" is a count of the alternate Region Managers serving. If these two numbers are equal, then all regions are being served, but if Down is greater than alternate, then some regions are currently without a Region Manager.

2. This example shows **machine_group** status for the cluster.

   ```
   llrstatus -L machine_group
   ```

   Or alternatively, set the environment variable **LOADL_STATUS_LEVEL=machine_group** and issue:

   ```
   llrstatus
   ```

   You should receive output similar to the following:

   ```
   GROUP_NAME   Schedd InQ    Act   Startd      Run Nmachs
   x330          8/8    90     11   99/100     1936 100
   pccluster    20/21   88     12  597/600    97227 600
   cnblade       9/9    21     21 9998/10000  10416 10000
   c197blade    46/46  769    159 5495/5498   13252 5498
   OTHERS        2/2     1      1    2/2          1 2
   ```

   **Note:** The OTHERS row contains a collection of machines that are not defined as part of any machine group.

3. This example shows that for machine level, **llrstatus** will print all the machines.

   ```
   llrstatus -L machine
   ```

   Or alternatively, set the environment variable **LOADL_STATUS_LEVEL=machine** and issue:

   ```
   llrstatus
   ```

   You should receive output similar to the following:

   ```
   Name                    Schedd InQ Act Startd Run LdAvg Idle Arch  OpSys
   k10n09.ppd.pok.ibm.com  Avail   3   1  Run      1 2.72     0 R6000 AIX71
   k10n12.ppd.pok.ibm.com  Avail   0   0  Idle     0 0.00   365 R6000 AIX71

   R6000/AIX71           2 machines   3 jobs   1 running tasks
   Total Machines        2 machines   3 jobs   1 running tasks

   The Resource Manager is defined on k10n09.ppd.pok.ibm.com

   All machines on the machine_list are present.
   ```

   The **standard listing** includes the following fields:

**Act** The number of job steps dispatched by the Schedd daemon on this machine.

**Arch** The hardware architecture of the machine as listed in the configuration file.

**Idle** The number of seconds since keyboard or mouse activity in a login session was detected. Highest number displayed is 9999.

**InQ** Number of job steps in the job queue of this Schedd machine.

**LdAvg** Berkeley one-minute load average on this machine.

**Name** Host name of the machine.

**OpSys**
The operating system on this machine.

**Run** The number of initiators used by the startd daemon to run LoadLeveler jobs on this machine. One initiator is used for each serial job step and one initiator is used for each task of a parallel job step.

**Schedd**
State of the Schedd daemon, which can be one of the following:
- Down
- Drned (Drained)
- Drning (Draining)
- Avail (Available)

For more information, see "The Schedd daemon" on page 8.

**Startd** State of the startd daemon, which can be:
- Busy
- Down
- Drned (Drained)
- Drning (Draining)
- Flush
- Idle
- None
- Run (Running)
- Suspnd (Suspend)

For more information, see "The startd daemon" on page 9.

**Total Machines**
The standard listing includes the following summary fields:

**jobs** The number of job steps in LoadLeveler job queues.

**machines**
The number of machines in the cluster that have made a status report to the central manager.

**running tasks**
The number of initiators used by all the startd daemons in the LoadLeveler cluster. One initiator is used for each serial job step. One initiator is used for each task of a parallel job step.

The standard listing also contains summary information for the cluster such as the type of scheduler and the cluster name.

4. This example generates the long listing. The long listing is generated when you specify the **-l** flag with the **llrstatus -L machine** command. See Appendix C, "llrstatus -l command output listing," on page 325 for sample output of long listings.

```
llrstatus –L machine –l c271f2rp02
```

Alternatively, set the environment variable **LOADL_STATUS_LEVEL=machine** and issue:

```
llrstatus -l c271f2rp02
```

The **long listing** includes the following fields:

**Adapter**
Network adapter information associated with this machine.

- For a switch adapter, the information format is:

  ```
  adapter_name(network_type, interface_name, interface_address,
  multilink_address, switch_node_number or adapter_logical_id,
  total_adapter_windows, total rCxt blocks, total imm send buffers,
  adapter_fabric_connectivity)
  ```

  For example:

  ```
  Adapter = network1833865768265265971s(striped,ml0,10.0.0.53,,-1,0,
     0 rCxt Blks,0111,)
  network18338657682652659713(aggregate ,,,10.0.0.53,-1,0,
     0 rCxt Blks,0,)
  network18338657682652659714(aggregate,,,10.0.0.53,-1,64,
     0 rCxt Blks,1,)
  network18338657682652659715(aggregate,,,10.0.0.53,-1,64,
     0 rCxt Blks,1,)
  network18338657682652659716(aggregate,,,10.0.0.53,-1,64,
     0 rCxt Blks,1,)
  ```

  Possible values for `adapter_state` are:

  **AdapterHBInterval**
  Specifies the amount of time, in seconds, that defines the heartbeat interval between the region manager and **startd**.

  **AdapterHBRetries**
  Specifies the number of heartbeat intervals that the region manager will wait before declaring the adapter on **startd** as down.

  **ErrAdapter**
  The adapter information is incorrect. More information might be available in the network table API log.

  **ErrDown**
  The adapter has been reported as down. This field is displayed as part of the command output when the adapter is found to be down by either an adapter heartbeat, or by the input/output control (**ioctl**) or protocol network services daemon (**PNSD**) status on the machine.

  **ErrInternal**
  An error occurred while accessing the adapter. More information might be available in the network table API log and if the **D_FULLDEBUG** and **D_ADAPTER** flags are specified for **STARTD_DEBUG**.

  **ErrNotConfigured**
  The adapter is not configured. This field is displayed as part of the command output only during start up and reconfiguration as a default value until it is changed by either an **ioctl** or **PNSD**, or an adapter heartbeat.

  **ErrNTBL**
  An error occurred while accessing the network table API.

More information might be available if the **D_FULLDEBUG** and **D_ADAPTER** flags are specified for **STARTD_DEBUG**.

**ErrPerm**
The network table API reported that LoadLeveler does not have permission to access the adapter. More information might be available in the network table API log.

**ErrPNSD**
An error occurred in the network table API. More information might be available in the network table API log.

**ErrType**
The network table API reported that the adapter is not a switch adapter. Check the adapter configuration.

**HB_DOWN**
Heartbeat is down between the **startd** and region manager.

**HB_REGION_DOWN**
The region manager is down.

**HB_UNKNOWN**
The initial heartbeat state.

**HB_UP**
Heartbeat is up between the **startd** and region manager.

**MachineDown**
The machine to which the adapter is attached could not be reached by LoadLeveler to query status.

**NOT READY**
An unspecified problem occurred when accessing the adapter state. More information might be available if the **D_FULLDEBUG** and **D_ADAPTER** flags are specified for **STARTD_DEBUG**.

**READY**
The adapter can be used for communication.

- For non-switch adapters, the format is:

```
adapter_name(network_type, interface_name, interface_address,
interface_netmask, multilink_address, config_state, heartbeat_state,
adapter_state)
```

For example:

```
en0(ethernet,c890f4ec05.ppd.pok.ibm.com,9.114.80.53,,READY,HB_UP,
   READY)
ml0(multilink,10.0.0.53,10.0.0.53,,READY,HB_UP,READY)
```

**Arch** The hardware architecture of this machine.

**Completed**
The job manager has this number of completed jobs.

**Config Time Stamp**
Date and time of last configuration or reconfiguration.

**ConfiguredClasses**
A list of configured classes and the associated number of configured initiators on this machine.

**ConsumableResources**
A list of consumable resources associated with this machine.

Each element of this list has the format: `resource_name(available, total)`.

**Note:** The individual CPU ID < > notation will be used to list individual CPU IDs instead of the CPU count ( ) notation for machines where the **RSET_SUPPORT** configuration file keyword is set to **RSET_MCM_AFFINITY**. The CPU count ( ) notation will be used when the **RSET_SUPPORT** configuration file keyword is set to **RSET_USER_DEFINED** or **RSET_NONE**, or if this keyword is not specified in the configuration file.

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

**Cpus** The number of CPUs on this machine.

**Current Frequency**
The current power frequency on this machine.

**Disk** The available space, in kilobytes (less 512 KB) in LoadLeveler's execute directory on this machine.

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

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

If a machine does not have switch adapters, the **FabricConnectivity** field has no meaning and should be ignored by the user.

**Feature**
The set of all features on this machine.

**FreeLargePageMemory**
Free Large Page memory.

In LoadLeveler for Linux, the **FreeLargePageMemory** field has no meaning and should be ignored by the user.

**FreeRealMemory**
Free real memory, in megabytes, on this machine. This value should track closely with the "fre" value of the **vmstat** command and the "free" value of the **svmon -G** command whose units are 4 KB blocks.

**Held** The job manager has this number of held jobs.

**Idle** The job manager has this number of idle jobs.

**Island** The name of the island to which this machine belongs.

**Keyboard Idle**
The number of seconds since last keyboard or mouse activity.

**KILL** The expression, defined following C conventions in the configuration file, that evaluates to true or false (T or F). This determines whether jobs running on this machine should be sent the SIGKILL signal.

**LargePageMemory**
Configured Large Page physical memory.

In LoadLeveler for Linux, the **LargePageMemory** field has no meaning and should be ignored by the user.

**LargePageSize**
The size of a Large Page memory block.

In LoadLeveler for Linux, the **LargePageSize** field has no meaning and should be ignored by the user.

**LoadAvg**
The Berkely one-minute load average on machine.

**Machine**
The fully qualified name of the machine.

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

**MasterMachPriority**
The machine priority for the parallel master node.

**Mcms** The MCMs information associated with this machine has the format:

```
mcm_info ... mcm_info
```

The format of *mcm_info* is:

```
MCMnumber Available Cpus: < cpulist > (Total Cpus)
Adapters: adapter_info .... adapter_info
```

where:

**Available Cpus**
Are the CPUs available for LoadLeveler on this MCM.

**Adapters**
Are the switch adapters connected to this MCM.

where:

*number*
Is the MCM sequence number:

*cpulist* Is a blank-delimited list of individual CPU IDs or CPU ranges, or a combination of both CPU IDs and CPU ranges associated with the MCM.

*adapter_info*
Has the format:

```
[available windows, available memory]
```

where:

**available windows**
Is the total number of available windows.

**available memory**
Is the total available memory. It is the number of rCxt blocks for switch adapters.

**Memory**
Is the regular physical memory, in megabytes, on this machine.

**Name** Is the host name of the machine.

**OpSys**
Is the operating system on this machine.

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

In LoadLeveler for Linux, the **PagesFreed** field has no meaning and should be ignored by the user.

**PagesPagedIn**
Pages paged in from paging space per second. This value corresponds to the "pi" value of the **vmstat** command output.

In LoadLeveler for Linux, the **PagesPagedIn** field has no meaning and should be ignored by the user.

**PagesPagedOut**
Pages paged out to paging space per second. This value corresponds to the "po" value of the **vmstat** command output.

In LoadLeveler for Linux, the **PagesPagedOut** field has no meaning and should be ignored by the user.

**PagesScanned**
Pages scanned by the page-replacement algorithm per second. This value corresponds to the "sr" value of the **vmstat** command output.

In LoadLeveler for Linux, the **PagesScanned** field has no meaning and should be ignored by the user.

**Pending**
The job manager has this number of pending jobs.

**Pool** The identifiers of the pools to which the startd machines of this machine group belong.

**Power Policy**
The site level energy policy. The crontab entry and duration will be displayed.

**Power State**
The current machine power state.

where:

**Working**
: Is when the node is working normally.

**Standby**
: Is when the node is suspended.

**Error**
: Is when the node failed to suspend or resume.

**Pending**
: Is when the node is transforming between Working and Standby.

**Poweroff**
: The node is powered off.

**Unknown**
: Is when the S3 function is disabled on the node.

**Region**
: Specifies the name of the region to which this machine belongs.

**Remove Pending**
: The job manager has this number of pending jobs to be removed.

**RSetSupportType**
: Indicates the type of RSet support set up on a machine. Possible values are:

  **RSET_MCM_AFFINITY**
  : Creates and attaches RSets for tasks to satisfy memory and affinity requests.

  **RSET_NONE**
  : Indicates that LoadLeveler RSet support is not available.

  **RSET_USER_DEFINED**
  : Attaches user-created RSets to Tasks.

**Running steps**
: The list of job steps currently running on this machine.

**ScheddAvail**
: The flag indicating if the machine is running a job manager daemon (**LoadL_schedd**), (0=no, 1=yes).

**ScheddRunning**
: The number of job steps submitted to this machine that are running somewhere in the LoadLeveler cluster.

**ScheddState**
: The state of the Schedd daemon on this machine, which can be one of the following:
  - Down
  - Drned (Drained)
  - Drning (Draining)
  - Avail (Available)

**Speed**
: The speed associated with the machine.

**SMT**
: Indicates whether the simultaneous multithreading (SMT) function is turned on, off, or is not supported in the running machine. Valid values are **Enabled**, **Disabled**, or **Not Supported**.

**START**
The expression, defined following C conventions in the configuration file, that evaluates to true or false (T or F). This determines whether jobs can be started on this machine.

**StartdAvail**
The flag indicating if machine is running a startd daemon (0=no, 1=yes).

**Starting**
The job manager has this number of jobs starting.

**State** State of the startd daemon, which can be:
- Busy
- Down
- Idle
- None
- Running
- Suspend

For more information, see "The startd daemon" on page 9.

**Subnet**
The TCP/IP subnet that this machine resides on.

**Supported Freq. List**
The supported power frequencies on this machine.

**SUSPEND**
The expression, defined following C conventions in the configuration file, that evaluates to true or false (T or F). This determines whether running jobs should be suspended on this machine.

**TimeStamp**
The date and time the resource manager last received a status update from the job manager on this machine.

**Tmp** Available space, in kilobytes (less than 512 KB) in the **/tmp** directory on this machine.

**Total Jobs**
The number of total job steps submitted to the job manager on this machine.

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

**Unexpanded**
The job manager has this number of unexpanded jobs.

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

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

5. This example generates the machine group level long listing. The long listing is generated when you specify the **-l** flag with the **llrstatus -L machine_group** command. See Appendix C, "llrstatus -l command output listing," on page 325 for sample output of long listings.

   ```
   llrstatus –L machine_group –l
   ```

   Alternatively, set the environment variable **LOADL_STATUS_LEVEL=machine_group** and issue:

   ```
   llrstatus -l
   ```

   The **long listing** includes the following fields:

   **Completed**
   The number of job steps in this state on all the Schedd machines of this machine group.

   **ConfiguredClasses**
   The list of configured classes and the associated number of configured initiators on each machine of this machine group.

   **ConsumableResources**
   The list of consumable resources associated with each machine of this machine group. Each element of this list has the format: `resource_name(available, total)`.

   **Note:** The individual CPU ID < > notation will be used to list individual CPU IDs instead of the CPU count ( ) notation for machines where the **RSET_SUPPORT** configuration file keyword is set to **RSET_MCM_AFFINITY**. The CPU count ( ) notation will be used when the **RSET_SUPPORT** configuration file keyword is set to **RSET_USER_DEFINED** or **RSET_NONE**, or if this keyword is not specified in the configuration file.

   **Explicitly Defined**
   All the explicitly defined submachines within this machine group.

   **Feature**
   The set of all features on each machine of this machine group.

   **Held**
   The number the number of job steps in this state on all the Schedd machines of this machine group.

   **Idle**
   The number of job steps in this state on all the Schedd machines of this machine group.

   **Island**
   The name of the island to which each machine in this machine group belongs.

   **Machine List**
   The **machine_list** keyword defined in the administration file followed by the machines in substanzas. It is possible that a machine in a substanza is also included in the **machine_list** keyword.

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

   **Max_Dstg_Starters**
   A machine-specific limit on the number of data staging initiators. This is also the number of initiators available on that machine for the **data_stage** class.

**Max_Starters**
The maximum number of initiators that can be used simultaneously on each machine of this machine group.

**Name** The name of the machine group stanza.

**Number of Machines**
Displays the number of machines in this machine group.

**Pending**
The number of job steps in this state on Schedd machines of this machine group.

**Pool** The identifier of the pool where the startd machines of this machine group are located.

**Prestarted_Starters**
The maximum number of Prestarted Starters that can be started on this machine or machine group at any time.

**Region**
Specifies the name of the region to which this machine group belongs.

**Removed**
The number of job steps in this state on Schedd machines of this machine group.

**Remove Pending**
The number of job steps in this state on Schedd machines of this machine group.

**ReservationPermitted**
Indicates whether or not the node can be reserved. It is displayed as T or F (true or false).

**Running steps**
The number of job steps currently running on all startd machines of this machine group.

**Running Tasks**
The number of initiators used by the startd daemon to run LoadLeveler jobs. One initiator is used for each serial job step. One initiator is used for each task of a parallel job step. For the machine group's long listing, it represents the number of initiators used by all the startd daemons in the machine group.

**ScheddAvail**
A number indicating the number of Schedd machines in this machine group.

**ScheddRunning**
The number of job steps submitted to the Schedd machines of this machine group that are running somewhere in the LoadLeveler cluster.

**ScheddState**
The state of the Schedd daemon on this machine, which can be one of the following:
- Down (*number*)
- Drned (Drained) (*number*)
- Drning (Draining) (*number*)
- Avail (Available) (*number*)

where: *number* is the number of Schedds in corresponding states within the machine group.

**Speed** The speed associated with the machines in this machine group.

**StartdAvail**
A number indicating the number of startds in this machine group.

**Starting**
The number of job steps in this state on Schedd machines of this machine group.

**State** The state of the startd daemon, which can be:
- Busy (*number*)
- Down (*number*)
- Drained (*number*)
- Draining (*number*)
- Flush (*number*)
- Idle (*number*)
- None (*number*)
- Running (*number*)
- Suspend (*number*)

where: *number* is the number of startds in corresponding states within the machine group.

For more information, see "The startd daemon" on page 9.

**Total Jobs**
The number of jobs steps submitted to Schedd machines of this machine group.

**Unexpanded**
The number of job steps in this state on Schedd machines of this machine group.

6. This example generates the cluster-level long listing:

```
llrstatus -L cluster -l
```

Or alternatively, set the environment variable **LOADL_STATUS_LEVEL=cluster** and issue:

```
llrstatus -l
```

You should receive output similar to the following:

```
Name                       Schedd  Startd
x330n51.clusters.com       Down    Run
cnblade11n10.clusters.com  Avail   Down
cnblade11n7.clusters.com   Avail   Suspend
cnblade11n8.clusters.com   Avail   Suspend
cnblade11n9.clusters.com   Avail   Suspend

90 machines are absent:
      x330n51.clusters.com
      pccluster99.clusters.com
      pccluster141.clusters.com
      pccluster202.clusters.com
      .....
      .....
      bldaix09.clusters.com

The Region Manager for cnblade11_region is defined on
      cnblade11n10.clusters.com, but is unusable.
Alternate Region Manager is serving from cnblade11n1.clusters.com

The Region Manager for pccluster_region is defined on
```

```
      pccluster100.clusters.com, but is unusable.
There is no Region Manager serving pccluster_region.

The Resource Manager is defined on percscm1.clusters.com
```

7. This example generates a listing of all of the consumable resources associated with all of the machines in the LoadLeveler cluster.

   ```
   llrstatus -R
   ```

   You should receive output similar to the following:

   ```
   Machine                        Consumable Resource(Total)
   -------------------------      -----------------------------------------
   c209f1n01.ppd.pok.ibm.com      ConsumableCpus(4)+ ConsumableMemory
                                    (1.000 gb) n01_res(500)
   c209f1n02.ppd.pok.ibm.com      ConsumableCpus< 0 1 2 3 4 >n02_res(500)
                                    Frame5(10)
   c209f1n05.ppd.pok.ibm.com      ConsumableCpus(4)+ ConsumableMemory
                                   (1.000 gb) spice2g6(360)

   c209f1n06.ppd.pok.ibm.com      CollectiveGroups (64)

   Resources with "+" appended to their names have the Total value reported
   from Startd.
   -------------------------      -----------------------------------------
   c209f1n01.ppd.pok.ibm.com      ConsumableCpus(4)+ ConsumableMemory
                                    (1.000 gb) n01_res(500)
   c209f1n02.ppd.pok.ibm.com      ConsumableCpus< 0 1 2 3 4 >n02_res(500)
                                    Frame5(10)
   c209f1n05.ppd.pok.ibm.com      ConsumableCpus(4)+ ConsumableMemory
                                   (1.000 gb) spice2g6(360)

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

   **Note:** The individual CPU ID < > notation will be used to list individual CPU IDs instead of the CPU count ( ) notation for machines where the **RSET_SUPPORT** configuration file keyword is set to **RSET_MCM_AFFINITY**. The CPU count ( ) notation will be used when the **RSET_SUPPORT** configuration file keyword is set to **RSET_USER_DEFINED** or **RSET_NONE**, or if this keyword is not specified in the configuration file.

8. This example generates a listing of information related to MCMs of machines in the LoadLeveler cluster.

   ```
   llrstatus -M
   ```

   You should receive output similar to the following:

   ```
   Machine              MCM details
   -------------------- -------------------------------------------------
   c890f14ec03.ppd.pok.ibm.com
                MCM0
                   Available Cpus :< 0-31 >(32)
                   Adapters       :
                MCM1
                   Available Cpus :< 32-63 >(32)
                   Adapters       :network18338657682652659714[64,0 rCxt Blks]
                                   network18338657682652659713[64,0 rCxt Blks]
                MCM2
                   Available Cpus :< 64-95 >(32)
                   Adapters       :
                MCM3
                   Available Cpus :< 96-127 >(32)
                   Adapters       :
   ```

**Note:** The **-M** option will list the MCM information only when the **RSET_SUPPORT** configuration file keyword is set to **RSET_MCM_AFFINITY**.

9. This example generates a customized and formatted standard listing.

```
llrstatus -f %n %scs %inq %m %v %sts %l %o
```

You should receive output similar to the following:

```
Name            Schedd  InQ  Memory  FreeVMemory Startd LdAvg  OpSys
l15.pok.ibm.com Avail   0    128     22708       Run    0.23   AIX71
l16.pok.ibm.com Avail   3    224     16732       Run    0.51   AIX71

R6000/AIX71            2 machines      3  jobs      3  running tasks
Total Machines         2 machines      3  jobs      3  running tasks

The Resource Manager is defined on 115.pok.ibm.com

All machines on the machine_list are present.
```

10. This example generates a customized and unformatted (raw) standard listing.

**Customized, Unformatted Standard Listing:** A customized Output fields are separated by an exclamation point (!).

```
llrstatus -r %n %scs %inq %m %v %sts %l %o
```

You should receive output similar to the following:

```
l15.pok.ibm.com!Avail!0!128!22688!Running!0.14!AIX71
l16.pok.ibm.com!Avail!3!224!16668!Running!0.37!AIX71
```

11. This example generates a listing containing information about the status of adapters associated with all of the machines in the LoadLeveler cluster:

```
llrstatus -a
```

You should receive output similar to the following:

```
=====================================================================
c890f4ec05.ppd.pok.ibm.com
iba0(InfiniBand,,,10.0.0.53,-1,0,0 rCxt Blks,01,)
iba1(InfiniBand,,,10.0.0.53,-1,0,0 rCxt Blks,11,)
network18338657682652659971s(striped,ml0,10.0.0.53,,-1,0,0 rCxt Blks,
   0111,)
network18338657682652659713(aggregate,,,10.0.0.53,-1,0,0 rCxt Blks,0,)
ib0(InfiniBand,192.168.8.53,192.168.8.53,10.0.0.53,158486581,64,
   0 rCxt Blks,0,,ErrDown,HB_UP,ErrDown,1,MCM0)
network18338657682652659714(aggregate,,,10.0.0.53,-1,64,0 rCxt Blks,1,)
ib1(InfiniBand,192.168.9.53,192.168.9.53,10.0.0.53,158486581,64,
   0 rCxt Blks,1,,READY,HB_UP,READY,2,MCM0)
network18338657682652659715(aggregate,,,10.0.0.53,-1,64,0 rCxt Blks,1,)
ib2(InfiniBand,192.168.10.53,192.168.10.53,10.0.0.53,158486581,64,
   0 rCxt Blks,1,,READY,HB_UP,READY,1,MCM1)
network18338657682652659716(aggregate,,,10.0.0.53,-1,64,0 rCxt Blks,1,)
ib3(InfiniBand,192.168.11.53,192.168.11.53,10.0.0.53,158486581,64,
   0 rCxt Blks,1,,READY,HB_UP,READY,2,MCM1)
en0(ethernet,c890f4ec05.ppd.pok.ibm.com,9.114.80.53,,READY,HB_UP,READY)
ml0(multilink,10.0.0.53,10.0.0.53,,READY,HB_UP,READY)

=====================================================================
c890f4ec04.ppd.pok.ibm.com
iba0(InfiniBand,,,,-1,0,0 rCxt Blks,11,)
iba1(InfiniBand,,,,-1,0,0 rCxt Blks,11,)
network18338657682652659971s(striped,ml0,10.0.0.52,,-1,64,0 rCxt Blks,
   1111,)
network18338657682652659713(aggregate,,,10.0.0.52,-1,64,0 rCxt Blks,1,)
ib0(InfiniBand,192.168.8.52,192.168.8.52,10.0.0.52,158486580,64,
   0 rCxt Blks,1,,READY,HB_UP,READY,1,MCM0)
network18338657682652659714(aggregate,,,10.0.0.52,-1,64,0 rCxt Blks,1,)
ib1(InfiniBand,192.168.9.52,192.168.9.52,10.0.0.52,158486580,64,
   0 rCxt Blks,1,,READY,HB_UP,READY,2,MCM0)
network18338657682652659715(aggregate,,,10.0.0.52,-1,64,0 rCxt Blks,1,)
ib2(InfiniBand,192.168.10.52,192.168.10.52,10.0.0.52,158486580,64,
   0 rCxt Blks,1,,READY,HB_UP,READY,1,MCM1)
network18338657682652659716(aggregate,,,10.0.0.52,-1,64,0 rCxt Blks,1,)
ib3(InfiniBand,192.168.11.52,192.168.11.52,10.0.0.52,158486580,64,
   0 rCxt Blks,1,,READY,HB_UP,READY,2,MCM1)
en0(ethernet,c890f4ec04.ppd.pok.ibm.com,9.114.80.52,,READY,HB_UP,READY)
ml0(multilink,10.0.0.52,10.0.0.52,,READY,HB_UP,READY)
```

- For HFI adapters, the information format is:

  ```
  adapter_name(network_type, interface_name, interface_address,
  multilink_address,switch_node_number or adapter_logical_id,
  total_adapter_windows, total rCxt blocks,total Imme Send Buffers,
  adapter_fabric_connectivity, config_state, heartbeat_state,adapter_state,
  port_number, MCM_id)
  ```

  For example:

  ```
  network8078501872035430s(striped,,,,-1,440,
     0 rCxt Blks, 256 Imm Send Buffers1,READY)
  network80785018720354304(aggregate,,,,-1,440,
     0 rCxt Blks, 256 Imm Send Buffers,1,READY)
  hf1(hfi,21.8.6.1,21.8.6.1,,0,220,
     0 rCxt Blks,128 Imm Send Buffers,
     1,READY,READY,HB_UNKNOWN,READY,,,MCM-1)
  hf0(hfi,20.8.6.1,20.8.6.1,,0,220,
     0 rCxt Blks,128 Imm Send Buffers,
     1,READY,READY,HB_UNKNOWN,READY,,,MCM-1)
  en0(ethernet,c250f08c06ap01.ppd.pok.ibm.com,10.8.6.1,,READY,HB_UNKNOWN,READY)
  ```

- For a switch adapter, the information format is:

  ```
  adapter_name(network_type, interface_name, interface_address,
  multilink_address, switch_node_number or adapter_logical_id,
  total_adapter_windows, total rCxt blocks,
  adapter_fabric_connectivity)
  ```

  For example:

  ```
  iba0(InfiniBand,,,,-1,0,0 rCxt Blks,11,)
  network1833865768265265971s(striped,ml0,10.0.0.52,,-1,64,
     0 rCxt Blks,1111,)
  network18338657682652659713(aggregate,,,10.0.0.52,-1,64,0 rCxt Blks,1,)
  ```

  Note that InfiniBand adapters contain additional information. The format is:

  ```
  adapter_name(network_type, interface_name, interface_address,
  multilink_address, switch_node_number or adapter_logical_id,
  total_adapter_windows, total rCxt blocks, adapter_fabric_connectivity,
  config_state, heartbeat_state, adapter_state, port_number, MCM_id)
  ```

  For example:

  ```
  ib0(InfiniBand,192.168.8.52,192.168.8.52,10.0.0.52,158486580,64,
  0 rCxt Blks,1,,READY,HB_UP,READY,1,MCM0)
  ```

- For non-switch adapters, the format is:

  ```
  adapter_name(network_type, interface_name, interface_address,
  multilink_address, config_state, heartbeat_state, adapter_state)
  ```

  For example:

  ```
  en0(ethernet,c197blade8b09.ppd.pok.ibm.com,9.114.198.233,,
  READY,HB_UP, READY)
  ```

- For InfiniBand, this example displays the port number on each InfiniBand adapter. The **llrstatus** command does not show port information for adapters that are not InfiniBand adapters:

  ```
  llrstatus -a
  ```

  You should receive output similar to the following:

```
======================================================================
c890f4ec05.ppd.pok.ibm.com
iba0(InfiniBand,,,10.0.0.53,-1,0,0 rCxt Blks,01,)
iba1(InfiniBand,,,10.0.0.53,-1,0,0 rCxt Blks,11,)
network1833865768265265971s(striped,ml0,10.0.0.53,,-1,0,
   0 rCxt Blks,0111,)
network18338657682652659713(aggregate,,,10.0.0.53,-1,0,0 rCxt Blks,0,)
ib0(InfiniBand,192.168.8.53,192.168.8.53,10.0.0.53,158486581,64,
   0 rCxt Blks,0,,ErrDown,HB_UP,ErrDown,1,MCM0)
network18338657682652659714(aggregate,,,10.0.0.53,-1,64,0 rCxt Blks,1,)
ib1(InfiniBand,192.168.9.53,192.168.9.53,10.0.0.53,158486581,64,
   0 rCxt Blks,1,,READY,HB_UP,READY,2,MCM0)
network18338657682652659715(aggregate,,,10.0.0.53,-1,64,0 rCxt Blks,1,)
ib2(InfiniBand,192.168.10.53,192.168.10.53,10.0.0.53,158486581,64,
   0 rCxt Blks,1,,READY,HB_UP,READY,1,MCM1)
network18338657682652659716(aggregate,,,10.0.0.53,-1,64,0 rCxt Blks,1,)
ib3(InfiniBand,192.168.11.53,192.168.11.53,10.0.0.53,158486581,64,
   0 rCxt Blks,1,,READY,HB_UP,READY,2,MCM1)
en0(ethernet,c890f4ec05.ppd.pok.ibm.com,9.114.80.53,,READY,HB_UP,READY)
ml0(multilink,10.0.0.53,10.0.0.53,,READY,HB_UP,READY)

======================================================================
c890f4ec04.ppd.pok.ibm.com
iba0(InfiniBand,,,,-1,0,0 rCxt Blks,11,)
iba1(InfiniBand,,,,-1,0,0 rCxt Blks,11,)
network1833865768265265971s(striped,ml0,10.0.0.52,,-1,64,
   0 rCxt Blks,1111,)
network18338657682652659713(aggregate,,,10.0.0.52,-1,64,0 rCxt Blks,1,)
ib0(InfiniBand,192.168.8.52,192.168.8.52,10.0.0.52,158486580,64,
   0 rCxt Blks,1,,READY,HB_UP,READY,1,MCM0)
network18338657682652659714(aggregate,,,10.0.0.52,-1,64,0 rCxt Blks,1,)
ib1(InfiniBand,192.168.9.52,192.168.9.52,10.0.0.52,158486580,64,
   0 rCxt Blks,1,,READY,HB_UP,READY,2,MCM0)
network18338657682652659715(aggregate,,,10.0.0.52,-1,64,0 rCxt Blks,1,)
ib2(InfiniBand,192.168.10.52,192.168.10.52,10.0.0.52,158486580,64,
   0 rCxt Blks,1,,READY,HB_UP,READY,1,MCM1)
network18338657682652659716(aggregate,,,10.0.0.52,-1,64,0 rCxt Blks,1,)
ib3(InfiniBand,192.168.11.52,192.168.11.52,10.0.0.52,158486580,64,
   0 rCxt Blks,1,,READY,HB_UP,READY,2,MCM1)
en0(ethernet,c890f4ec04.ppd.pok.ibm.com,9.114.80.52,,READY,HB_UP,READY)
ml0(multilink,10.0.0.52,10.0.0.52,,READY,HB_UP,READY)
```

12. The following example shows output for the **llrstatus -l** command that displays the SMT state for the machine:

```
======================================================================
     Name = devf1n01.clusters.com
  Machine = devf1n01.clusters.com
     Arch = ppc64
   OpSys = Linux2
.
.
.
  Feature = SMT
.
.
.
      SMT = Enabled | Disabled | Not Supported
TimeStamp = Fri 23 Jan 2009 04:44:06 PM CST
.
.
.
```

13. On a system that has one InfiniBand adapter and multiple MCMs, the **llrstatus -a** or **llrstatus -M** command displays the InfiniBand network interfaces connected to only one MCM. Processors in the other MCMs, however, can still communicate through the same network interfaces.

- For example, if you issue:

  ```
  llrstatus -a c395f2n02
  ```

  You should receive output similar to the following:

  ```
  ==========================================================================
  c395f3n07.ppd.pok.ibm.com
  iba0(InfiniBand,,,,-1,0,0 rCxt Blks,11,)
  network18338657682652659715(striped,ml0,192.168.200.22,,-1,64,
     0 rCxt Blks,11,)
  network18338657682652659713(aggregate,,,192.168.200.22,-1,64,
     0 rCxt Blks,1,)
  ib0(InfiniBand,c395f3n07ib0,192.168.100.22,192.168.200.22,158473815,64,
     0 rCxt Blks,1,,READY,HB_UP,READY,1,MCM1)
  network18338657682652659714(aggregate,,,192.168.200.22,-1,64,
     0 rCxt Blks,1,)
  ib1(InfiniBand,c395f3n07ib1,192.168.101.22,192.168.200.22,158473815,64,
     0 rCxt Blks,1,,READY,HB_UP,READY,2,MCM1)
  en0(ethernet,c395f3n07.ppd.pok.ibm.com,9.114.30.87,,READY,HB_UP,READY)
  ml0(multilink,c395f3n07ml0,192.168.200.22,,READY,HB_UP,READY)
  ```

- For example, if you issue:

  ```
  llrstatus -M c395f2n02
  ```

  You should receive output similar to the following:

  ```
  Machine              MCM details
  -------------------- --------------------------------------------------
  c395f2n02.ppd.pok.ibm.com
        MCM0
           Available Cpus :< 0-3 >(4)
           Adapters       :
        MCM1
           Available Cpus :< 4-7 >(4)
           Adapters       :network18338657682652659713[64,0 rCxt Blks]
                           network18338657682652659714[64,0 rCxt Blks]
        MCM2
           Available Cpus :< 8-11 >(4)
           Adapters       :
        MCM3
           Available Cpus :< 12-15 >(4)
           Adapters       :
  ```

14. To display energy coefficients information, issue:

    ```
    llrstatus -e
    ```

    You should receive output similar to the following:

    ```
    Frequency: 2.73 GHZ
    Coefficients: A = 4.0 B = 11.7 C = 3477.6 D = 3.9 E = 7.2 F = 2639.0

    Frequency: 2.56 GHZ
    Coefficients: A = 5.0 B = 11.0 C = 3200.0 D = 3.0 E = 6.2 F = 2600.0

    Frequency: 2.31 GHZ
    Coefficients: A = 4.0 B = 11.7 C = 3477.6 D = 3.9 E = 7.2 F = 2639.0

    Frequency: 2.27 GHZ
    Coefficients: A = 5.0 B = 11.0 C = 3200.0 D = 3.0 E = 6.2 F = 2600.0
    ```

## Security

LoadLeveler administrators and users can issue this command.

# Chapter 9. Application programming interfaces (APIs)

The LoadLeveler resource manager application programming interfaces (APIs) allow application programs written by customers to interact with the LoadLeveler environment using specific data and functions that are a part of LoadLeveler.

These interfaces can be subroutines within a library or installation exits.

The header file **llrapi.h** defines all of the API data structures and subroutines. This file is located in the **include** subdirectory of the LoadLeveler release directory. You must include this file when you call any API subroutine.

The library **libllrapi.a** (AIX) or **libllrapi.so** (Linux) is a shared library containing all of the LoadLeveler API subroutines. A link to the library is located in **/usr/lib**. A link to the library on 64-bit Linux platforms is located in **/usr/lib64**.

**Attention:** These APIs are *thread safe* and can be linked to by a threaded application. On AIX, xlC must be used to link a C object with the **libllrapi.a** library.

Table 26 lists all of the LoadLeveler APIs, along with the intended users and supported operating systems for each, and a reference to the full descriptions of each interface.

*Table 26. LoadLeveler API summary*

| Task / API category | Interface name | Intended users | Supported operating systems |
|---|---|---|---|
| Generate accounting reports<br><br>"Accounting API" on page 223 | **"llr_get_history subroutine" on page 224** | Both administrators and general users | AIX and Linux |
| Process configuration files<br><br>"Configuration API" on page 225 | **"llr_config subroutine" on page 226** | Both administrators and general users | AIX and Linux |
| Access LoadLeveler objects and retrieve data from objects<br><br>"Data access API" on page 229 | **"llr_get_data subroutine" on page 235** | Both administrators and general users | AIX and Linux |
| | **"llr_query_free_data subroutine" on page 272** | Both administrators and general users | AIX and Linux |
| | **"llr_query_get_data subroutine" on page 273** | Both administrators and general users | AIX and Linux |
| | **"llr_query_set subroutine" on page 275** | Both administrators and general users | AIX and Linux |

*Table 26. LoadLeveler API summary (continued)*

| Task / API category | Interface name | Intended users | Supported operating systems |
|---|---|---|---|
| Register for job and machine-related events<br><br>"Event handling API" on page 279 | **"llr_free_event subroutine" on page 280** | Both administrators and general users | AIX and Linux |
| | **"llr_free_resmgr subroutine" on page 281** | Both administrators and general users | AIX and Linux |
| | **"llr_get_event subroutine" on page 282** | Both administrators and general users | AIX and Linux |
| | **"llr_get_event_fd subroutine" on page 287** | Both administrators and general users | AIX and Linux |
| | **"llr_init_resmgr subroutine" on page 288** | Both administrators and general users | AIX and Linux |
| | **"llr_read_events subroutine" on page 289** | Both administrators and general users | AIX and Linux |
| | **"llr_register_for_events subroutine" on page 290** | Both administrators and general users | AIX and Linux |
| | **"llr_unregister_for_events subroutine" on page 292** | Both administrators and general users | AIX and Linux |
| | **"llr_version_resmgr subroutine" on page 294** | Both administrators and general users | AIX and Linux |

Table 26. LoadLeveler API summary (continued)

| Task / API category | Interface name | Intended users | Supported operating systems |
|---|---|---|---|
| Perform LoadLeveler control operations and work with an external scheduler<br><br>"Workload management API" on page 295 | **"llr_add_job subroutine" on page 296** | Both administrators and general users | AIX and Linux |
| | **"llr_change_node_powerstate subroutine" on page 298** | Administrators only | Linux |
| | **"llr_cluster_auth subroutine" on page 300** | Both administrators and general users | AIX and Linux |
| | **"llr_control_job subroutine" on page 304** | Both administrators and general users | AIX and Linux |
| | **"llr_control subroutine" on page 302** | Both administrators and general users can specify the following control operations:<br>• `LLR_CONTROL_DRAIN_JOBMGR`<br>• `LLR_CONTROL_DUMP_LOGS`<br>• `LLR_CONTROL_RECONFIG`<br>• `LLR_CONTROL_RECYCLE`<br>• `LLR_CONTROL_RESUME_JOBMGR`<br>• `LLR_CONTROL_START`<br>• `LLR_CONTROL_STOP`<br><br>All other control operations defined in the **llrapi.h** header file are intended for use by administrators only. | AIX and Linux |
| | **"llr_delete_job subroutine" on page 306** | Both administrators and general users | AIX and Linux |
| | **"llr_error subroutine" on page 307** | Both administrators and general users | AIX and Linux |
| | **"llr_free_job subroutine" on page 308** | Both administrators and general users | AIX and Linux |
| | **"llr_move_spool subroutine" on page 309** | Both administrators and general users | AIX and Linux |
| | **"llr_remove_energy_tags subroutine" on page 311** | Both administrators and general users | Linux |
| | **"llr_start_job_migration subroutine" on page 313** | Both administrators and general users | AIX and Linux |
| | **"llr_start_job_step subroutine" on page 315** | Both administrators and general users | AIX and Linux |

## Accounting API

Use the accounting API to access the resource manager's job accounting data.

This API consists of the following subroutine:
- **"llr_get_history subroutine" on page 224**

Job accounting information saved in a history file can also be queried by using the data access API.

## llr_get_history subroutine

Use the **llr_get_history** subroutine to process local or global LoadLeveler history files.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_get_history (llr_resmgr_handle_t *rm_handle, const char *filename,
                     int (*func) (llr_element_t *), llr_element_t **errObj)
```

### Parameters

*rm_handle*
: Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*filename*
: Specifies the name of the history file.

**(*func) (llr_element_t *)**
: Specifies the user-supplied function you want to call to process each history record. The function must return an integer and must accept as input a pointer to **llr_element_t *** that contains the job data. The **llr_get_data** function can be used to access the job data.

*errObj*
: Is the address of a pointer to a LoadLeveler error object. The pointer to the *errObj* should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

### Description

The **llr_get_history** subroutine opens the history file you specify, reads one accounting record, and calls a user-supplied routine, passing to the routine the address of the object that contains the job data. **llr_get_history** processes all history records one at a time and then closes the file. Any user can call this subroutine.

### Return Values

**LLR_API_OK**
: The history file was successfully processed.

**LLR_API_ERROR**
: The history file could not be processed. A description of the error is provided in the *errObj* parameter.

## Configuration API

Use the configuration API to operate on LoadLeveler configuration files.

This API consists of the following subroutine:

- **"llr_config subroutine" on page 226**

## llr_config subroutine

Use the **llr_config** subroutine to manage LoadLeveler's configuration database. This subroutine provides options to display or change the value of the LoadLeveler keywords when the database option is being used.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
#include llrapi.h
enum LLR_config_op { R_CONFIG_C, R_CONFIG_D, R_CONFIG_I, R_CONFIG_STANZA_A,
                     R_CONFIG_STANZA_C, R_CONFIG_STANZA_D, R_CONFIG_STANZA_F,
                     R_CONFIG_STANZA_R };



int llr_config (int version, enum LLR_config_op config_op, char **input,
              char ***output, char **hostlist,
              llr_element_t **err_obj,int no_reconfig););
```

### Parameters

*version*
: Specifies the version of this API.

*config_op*
: Specifies the operation type:

**R_CONFIG_C**
: Changes the value of the specified keywords relating to machines.

**R_CONFIG_D**
: Displays the value of the specified keywords relating to machines.

**R_CONFIG_I**
: Reserved for LoadLeveler internal use.

**R_CONFIG_STANZA_A**
: Adds a new stanza to the configuration database with no keyword values set.

**R_CONFIG_STANZA_C**
: Changes the value of the specified keywords relating to machine, machine_group, user, group, class, region, and cluster.

**R_CONFIG_STANZA_D**
: Displays the value of the specified keywords relating to machine, machine_group, user, group, class, region, and cluster.

**R_CONFIG_STANZA_F**
: Adds new stanzas to the database from a specified file.

**R_CONFIG_STANZA_R**
: Removes the specified stanza from the configuration database.

*input*
: A NULL-terminated **char*** array of input that depends on *config_op*.

**R_CONFIG_C**
: The elements of the array are strings in a format similar to **keyword =***value*. They are used to change the value of any configuration

keywords relating to machines. Keywords related to user, group, class, machine_group, and cluster cannot be changed using this command.

**R_CONFIG_D**
Each element of the array is one keyword whose value is to be displayed. This option shows the value of any configuration keywords relating to machines. Keywords related to user, group, class, machine_group, and cluster cannot be displayed using this option.

**R_CONFIG_STANZA_A**
The elements of the array are strings in a format similar to *stanza=value*[*:substanza*]. The stanza can be machine, machine_group, user, group, class, region, and cluster.

**R_CONFIG_STANZA_C**
The elements of the array are strings in a format similar to *stanza=value*[*:substanza*] *keyword1=value1*.... They are used to change the value of any configuration keywords relating to machine, machine_group, user, group, class, region, and cluster.

**R_CONFIG_STANZA_D**
The elements of the array are strings in a format similar to *stanza*[*=value*[*:substanza*]]. They are used to display the value of any configuration keywords relating to machine, machine_group, user, group, class, region, and cluster.

**R_CONFIG_STANZA_F**
The array can only have one element that specifies the stanza file that contains the new stanzas.

**R_CONFIG_STANZA_R**
The elements of the array are strings in a format similar to *stanza=value*[*:substanza*]. The stanza can be machine, machine_group, user, group, class, region, and cluster.

*output*
A NULL-terminated **char*** array for **R_CONFIG_D**. Each element of the array is a string in a format similar to **keyword=***value* and each string corresponds to one keyword from the input. It is the caller's responsibility to free the storage associated with this array.

This parameter is also used by **R_CONFIG_STANZA_D**. Each element of the array is a string in a format similar to how the stanza is configured in the administration file.

*hostlist*
A NULL-terminated **char*** array of host names.

*errObj*
Is the address of a pointer to a LoadLeveler error object. The pointer to the **llr_element_t** should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed through the **llr_error** function. The caller should free the error object storage before reusing the pointer. If this function fails with some keywords, but succeeds with others, the error messages for the failed keywords are stored in this object.

## Description

This subroutine performs operations that are essentially equivalent to that performed by the **llrconfig** command with the **-c** or **-d** option.

## Return Values

**LLRCONFIG_DB_CANNOT_CONNECT**
: Cannot connect to the database.

**LLRCONFIG_CHECK_FAILED**
: Configuration errors were found when checking the configuration files.

**LLRCONFIG_CHANGE_AUTH_FAILED**
: The caller is not authorized to run this operation.

**LLRCONFIG_CHANGE_OK**
: The values for all or some keywords were changed. It is possible that some keywords are not valid or that there is no related record in the database.

**LLRCONFIG_CHANGE_RECONFIG_ERROR**
: The values for all or some keywords were changed, but the reconfiguration procedure cannot finish for some or all hosts.

**LLRCONFIG_NOT_VALID_INPUT**
: An input parameter was specified that is not valid.

**LLRCONFIG_DISPLAY_OK**
: The records for all or some keywords were found and displayed.

**LLRCONFIG_LOAD_ODBC_LIBRARY_FAILED**
: The unixODBC library cannot be loaded.

**LLRCONFIG_LOADLDB_NOT_CONFIG**
: The **LoadLDB** keyword is not specified in the master configuration file.

**LLRCONFIG_CONFIG_OPT_UNSUPPORTED**
: A *config_op* was specified that is not valid.

## Related Information

Commands: **llrconfig**

## Data access API

Use the data access API for a process to query job and machine data from the resource manager and provide the functions to traverse the job and machine objects and fetch the values for specific attributes.

You can use this API to query LoadLeveler resource manager daemons for information about:
- CLUSTER (cluster information)
- JOBS (job information)
- JOBQ_SUMMARY (high-level job summary information)
- MACHINES (machine information)
- MACHINE_GROUP (machine group level information)
- WLMSTAT (Workload Manager)

This API can also be used to query a LoadLeveler history file for accounting information about JOBS.

This API consists of the following subroutines:
- **"llr_get_data subroutine" on page 235**
- **"llr_query_free_data subroutine" on page 272**
- **"llr_query_get_data subroutine" on page 273**
- **"llr_query_set subroutine" on page 275**

### Using the data access API

The data access API makes use of a query object to facilitate each request for LoadLeveler data.

The query object is used to specify the details of a query request. It is initialized by the **llr_query_set** subroutine to specify the kind of LoadLeveler objects requested and to specify filtering of those objects. It is then passed to the **llr_query_get_data** subroutine to retrieve the requested data.

To use this API, you need to call the subroutines in the following order:
- Call **llr_init_resmgr**, which must be called before using any of the data access APIs, to obtain the pointer to an opaque structure (**llr_resmgr_handle_t**). See "llr_init_resmgr subroutine" on page 288 for more information.
- Call **llr_query_set** to specify the type of query and to filter the objects you want to query. See "llr_query_set subroutine" on page 275 for more information.
- Call **llr_query_get_data** to send a query request to the specified daemon. See "llr_query_get_data subroutine" on page 273 for more information.
- Call **llr_get_data** to access the data in the objects returned in **llr_query_get_data**. See "llr_get_data subroutine" on page 235 for more information.

For more information on LoadLeveler objects, see "Understanding the LoadLeveler data access object model."

### Understanding the LoadLeveler data access object model

The **llr_get_data** subroutine of the data access API allows you to access the LoadLeveler object model.

A LoadLeveler object model consists of a root object with zero or more associated objects that have attributes and associations to other objects. An attribute is a characteristic of the object and generally has a primitive data type (such as integer, float, or character). Attributes and related objects are retrieved from an object using the “llr_get_data subroutine” on page 235. One of the arguments to the **llr_get_data** subroutine is the specification. Specifications are used to identify which attribute or related object to retrieve. The job name, submission time, and job priority are examples of attributes.

Objects are associated with one or more other objects through relationships. An object can be associated with other objects through more than one relationship, or through the same relationship. For example, A Job object is associated with a Credential object and to Step objects through two different relationships. A Job object can be associated with more than one Step object through the same relationship of “having a Step.” When an object is associated through different relationships, different specifications are used to retrieve the appropriate object.

When an object is associated with more than one object through the same relationship, there are GetFirst and GetNext specifications associated with the relationship. You must use the GetFirst specification to initialize access to the first such associated object. You must use the GetNext specification to get the remaining objects in succession. After the last object has been retrieved, GetNext will return NULL.

The specification tables for the **llr_get_data** subroutine (“llr_get_data subroutine” on page 235) in the data access API describe the specifications and elements available when you use the **llr_get_data** subroutine. Each specification name describes the object you need to specify and the attribute returned. For example, the specification **LLR_JobGetFirstStep** includes the object you need to specify (**Job**) and the value returned (**llr_element_t* Step**). The tables are sorted alphabetically by object.

You can use the **llr_get_data** subroutine to access both attributes and associated objects. See the “llr_get_data subroutine” on page 235 for more information.

It is important to keep in mind how the list of objects returned by the **llr_query_get_data** subroutine have been defined by the previous data access API calls when getting specific pieces of data using the **llr_get_data** subroutine.

Several factors can affect what and when data attributes are available or relevant:
- Data related to certain LoadLeveler features (such as Blue Gene or multicluster) either will not be available, or the data will not be relevant unless the feature is configured
- What source returned the data object
- What type of job is being queried
- What is the state of the job being queried

## Understanding the Cluster object model

To access the Cluster model, initialize the query object using **llr_query** with the LLR_CLUSTER_QUERY query_type.

The root of the Cluster model is the Cluster object.

Cluster objects can only be obtained from the LLR_QUERY_RESOURCE_MANAGER query daemon.

Figure 10 shows the Cluster object model:

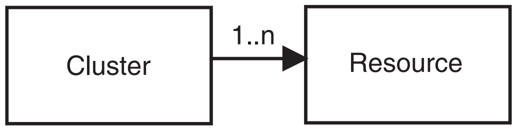

*Figure 10. LoadLeveler Cluster object model*

See the “llr_get_data subroutine” on page 235 for a listing of LLR_CLUSTER_QUERY specifications.

## Understanding the Job object model

To access the Job model, initialize the query object using llr_query with the JOBS query_type.

The root of the job model is the Job object. The Job is associated with a single Credential object, one or more Step objects, and zero or more ClusterFile objects.

The Step object represents one executable unit of the Job (all the tasks that are executed together). The Step is associated with one or more Machine objects, one or more Node objects, one or more MachUsage objects, zero or more NetworkUsage objects, and zero or more AdapterReq objects. The only relevant data within the Machine object when linked to a Step is the machine name. The list of Machines represents all of the hosts assigned to a step that is in a running-like state. If two or more nodes are running on the same host, the Machine object for the host occurs only once in the Step's Machine list.

The MachUsage object is associated with one or more DispUsage objects. The DispUsage object is associated with one or more EventUsage objects. The MachUsage, DispUsage, and EventUsage objects contain accounting information available from the history file only.

Each Node object manages a set of executables that share common requirements and preferences. The Node object is associated with one or more Task objects.

The Task object represents one or more copies of the same executable. The Task object is associated with one or more ResourceReq objects and one or more TaskInstance objects.

Job objects can only be obtained from the LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE query daemon.

Figure 11 on page 232 shows the Job object model:

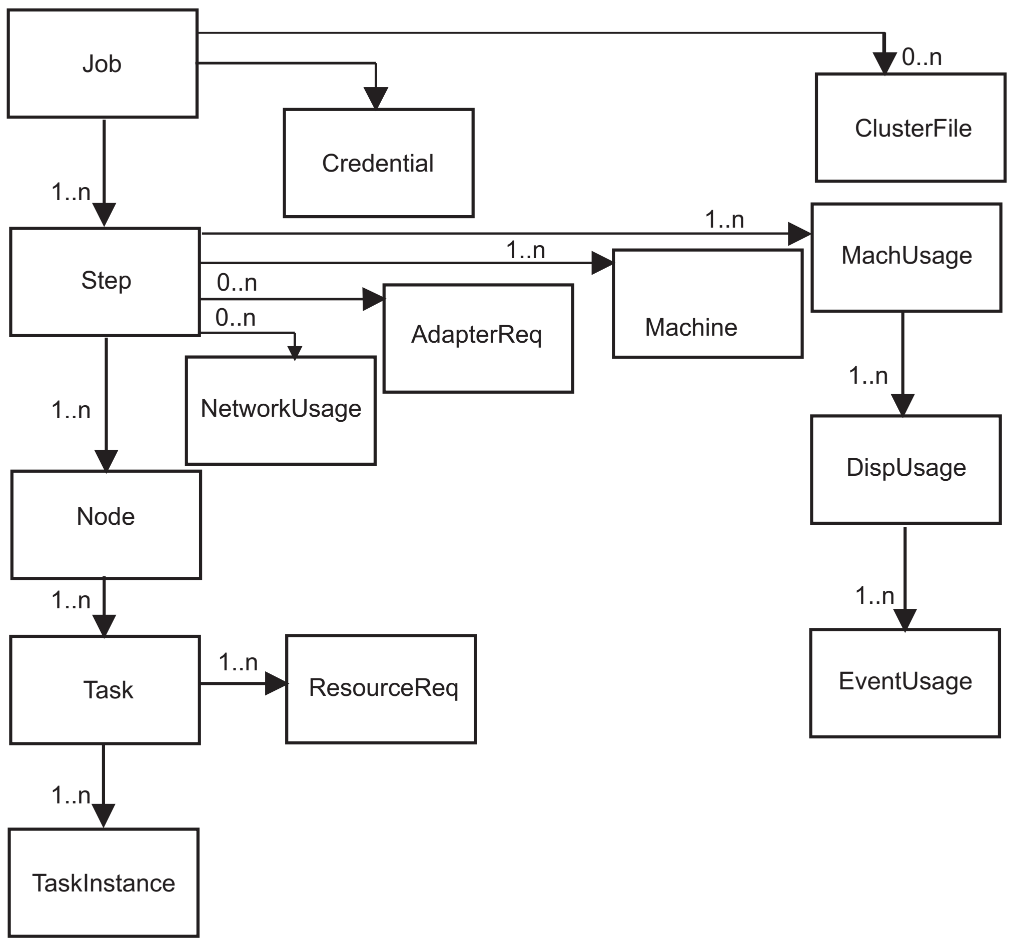

*Figure 11. LoadLeveler Job object model*

See Table 29 on page 238 for a listing of JOB specifications.

## Understanding the LLR_JOBQ_SUMMARY_QUERY object model

To access the LLR_JOBQ_SUMMARY_QUERY object model, initialize the query object using **llr_query_set** with the LLR_JOBQ_SUMMARY_QUERY query_type.

The JOBQ summary object model consists of JobSummary objects only. These objects represent high-level summary information about jobs.

The root of the JOBQ summary object model is the JobSummary object.

JobSummary objects can only be obtained from the LL_CM query daemon.

Figure 12 shows the LLR_JOBQ_SUMMARY_QUERY object model:

*Figure 12. LoadLeveler LLR_JOBQ_SUMMARY_QUERY object model*

See Table 30 on page 263 for a listing of LLR_JOBQ_SUMMARY_QUERY specifications.

## Understanding the Machine object model

To access the Machine model, initialize the query object using llr_query with the MACHINES query_type.

The root of the machine model is the Machine object. The machine is associated with one or more Resource objects, one or more Adapter objects, and one or more MCM objects depending on the configuration of the machine that the Machine object represents.

Machine objects can only be obtained from the LLR_CM query daemon.

Figure 13 shows the Machine object model:

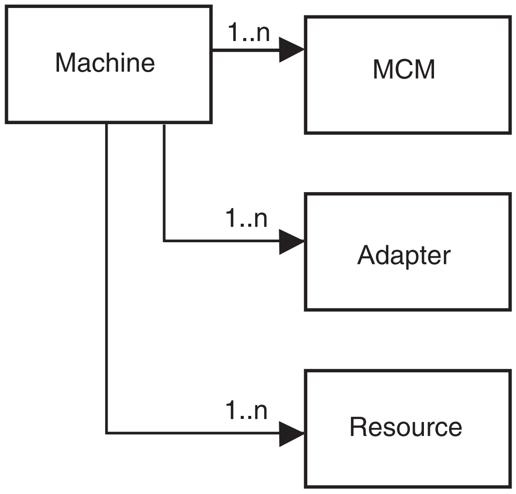

*Figure 13. LoadLeveler Machine object model*

See Table 31 on page 263 for a listing of MACHINE specifications.

## Understanding the Machine group object model

To access the Machine group model, initialize the query object using **llr_query_set** with the LLR_MACHINE_GROUP query_type.

The root of the machine group model is the MachineGroup object.

Machine group objects can only be obtained from the LLR_QUERY_RESOURCE_MANAGER query daemon.

Figure 14 shows the Machine group object model:

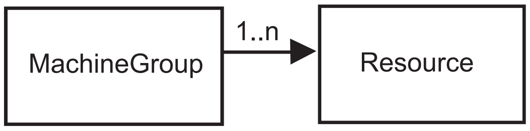

*Figure 14. LoadLeveler Machine object model*

See Table 32 on page 268 for a listing of LLR_MACHINE_GROUP specifications.

## Understanding the Wlmstat object model

To access the Wlmstat model, initialize the query object using llr_query with the WLMSTAT query_type.

The Wlmstat object model consists of Wlmstat objects only. These objects return WLM statistics about running steps. Wlmstat objects must be queried for a specific step.

WlmStat objects can only be obtained from the LLR_STARTD query daemon.

Figure 15 shows the Wlmstat object model:

*Figure 15. LoadLeveler Wlmstat object model*

See Table 33 on page 270 for a listing of WLMSTAT specifications.

## Understanding the LLEnergyTag object model

To access the energy tag model, initialize the query object using the **llr_query_set** subroutine with the **LLR_ENERGYTAG_QUERY** query type.

The LLEnergyTag object model consists of the LLEnergyTag object only. Attributes are retrieved from it using the returned pointer to a **llr_etag_policy** structure which contains the energy tag detail information.

LLEnergyTag objects can only be obtained from the **LLR_QUERY_RESOURCE_MANAGER** query daemon.

Figure 16 shows the LLEnergyTag object model:

*Figure 16. LoadLeveler LLEnergyTag object model*

See Table 34 on page 271 for a listing of LLEnergyTag specifications.

## llr_get_data subroutine

Use the **llr_get_data** subroutine to get data from **llr_element_t** subroutine.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_get_data (llr_resmgr_handle_t *rm_handle, llr_element_t *element,
                  llr_data_specification specification, void *resulting_data,
                  llr_element_t **errObj);
```

### Parameters

*rm_handle*
: Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*element*
: Is a pointer to the **llr_element_t** returned by the **llr_query** subroutine.

*specification*
: Is an enum specifying the data field within the data object to be read.

*resulting_data*
: Is a pointer to the location where the data is stored. If the call returns a nonzero value, an error has occurred and the contents of the location are undefined.

*errorObj*
: Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

### Description

**Note:** Before you use this subroutine, make sure you are familiar with the concepts discussed in "Understanding the Job object model" on page 231

*object* and *specification* are the input fields, while *resulting_data* is an output field.

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

If the specification indicates an attribute of the element that is passed in, the result pointer must be the address of a variable of the appropriate type, and must be initialized to NULL. The type returned by each specification is found in the specification tables. If the specification queries the connection to another object, the returned value is of type **llr_element_t**. You can use a subsequent **llr_get_data** call to query information about the new object.

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

**llr_element_t** pointers cannot be freed by the caller.

For the specifications, **LLR_MachineOperatingSystem** and **LLR_MachineArchitecture**, *resulting_data_type* returns the string "???" if a query is made before the associated records are updated with their actual values by the appropriate startd daemons.

## Return Values

**LLR_API_OK**
: The data was successfully read from the object.

**LLR_API_ERROR**
: The data could not be read. A description of the error is provided in the *errObj* parameter.

## Related Information

Subroutines: **llr_query_free_data**, **llr_query_get_data**, **llr_query_set**

Table 27 provides information on which daemons respond to which *query_flags*:

*Table 27. query_flags summary*

| When query type (llr_query_type) is: | Filter_type (in llr_query_filter_type) can be: | Responded to by these daemons (in llr_query_source): |
|---|---|---|
| **LLR_CLUSTER_QUERY** | No query filters are supported for this query type. | **LLR_QUERY_RESOURCE_MANAGER** |
| **LLR_ENERGYTAG_QUERY** | **LLR_QUERY_USER**<br>**LLR_QUERY_JOBID**<br>**LLR_QUERY_STEPID**<br>**LLR_QUERY_ETAGNAME**<br>NULL | **LLR_QUERY_RESOURCE_MANAGER** |
| **LLR_JOBQ_SUMMARY_QUERY** | **LLR_QUERY_GROUP** | **LLR_QUERY_RESOURCE_MANAGER** |
| | **LLR_QUERY_USER** | **LLR_QUERY_RESOURCE_MANAGER** |
| **LLR_JOBS_QUERY** | **LLR_QUERY_ENDDATE** | **LLR_QUERY_HISTORY_FILE** |
| | **LLR_QUERY_GROUP** | **LLR_QUERY_RESOURCE_MANAGER** |
| | **LLR_QUERY_HOST** | **LLR_QUERY_RESOURCE_MANAGER** |
| | **LLR_QUERY_JOBID** | **LLR_QUERY_JOBMGR, LLR_QUERY_RESOURCE_MANAGER** |
| | **LLR_QUERY_MEDIUM** | **LLR_QUERY_RESOURCE_MANAGER** |
| | **LLR_QUERY_STARTDATE** | **LLR_QUERY_HISTORY_FILE** |
| | **LLR_QUERY_STEPID** | **LLR_QUERY_JOBMGR, LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_STARTD** |
| | **LLR_QUERY_USER** | **LLR_QUERY_JOBMGR, LLR_QUERY_RESOURCE_MANAGER** |
| | NULL | **LLR_QUERY_HISTORY_FILE, LLR_QUERY_JOBMGR, LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_STARTER** |
| **LLR_MACHINE_GROUP_QUERY** | **LLR_QUERY_HOST** | **LLR_QUERY_RESOURCE_MANAGER** |

Table 27. query_flags summary (continued)

| When query type (llr_query_type) is: | Filter_type (in llr_query_filter_type) can be: | Responded to by these daemons (in llr_query_source): |
|---|---|---|
| **LLR_MACHINES_QUERY** | **LLR_QUERY_HOST** | **LLR_QUERY_RESOURCE_MANAGER** |
| | NULL | **LLR_QUERY_RESOURCE_MANAGER** |
| **LLR_WLMSTAT_QUERY** | **LLR_QUERY_STEPID** | **LLR_QUERY_RESOURCE_MANAGER** |
| | NULL | **LLR_QUERY_RESOURCE_MANAGER** |

See "Understanding the Cluster object model" on page 230 for more information on the Cluster object model.

Table 28 lists the CLUSTERS specifications for the **llr_get_data** subroutine:

Table 28. CLUSTERS specifications for llr_get_data subroutine

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Cluster | LLR_ClusterGetFirstUnavailableMachine | llr_element_t* (Machine) | A pointer to the first machine object that has either an unavailable startd or an unavailable Schedd. Only LL_MachineName, LL_MachineScheddRunsHere, LL_MachineStartdRunsHere, LL_MachineStartdState, or LL_MachineScheddStates can be used as specifications when querying the machine object. |
| Cluster | LLR_ClusterGetNextUnavailableMachine | llr_element_t* (Machine) | A pointer to the next machine object that has either an unavailable or an unavailable Schedd. Only LL_MachineName, LL_MachineScheddRunsHere, LL_MachineStartdRunsHere, LL_MachineStartdState, or LL_MachineScheddStates can be used as specifications when querying the machine object. |
| Cluster | LLR_ClusterGetUnavailableMachineList | llr_data_t * | The machine iterator. |
| Cluster | LLR_ClusterJobStepsInQueue | int* | The total number of job steps queued. |
| Cluster | LLR_ClusterMachineAbsentList | char*** | A NULL-terminated list of hostnames of machines that are present in the LoadLeveler configuration but on which no daemons are running. |
| Cluster | LLR_ClusterRegionManagerAlternateList | char*** | A NULL-terminated list of hostnames of machines that are serving as Region Managers for regions where the primary Region Manager has become unavailable. |

Table 28. CLUSTERS specifications for llr_get_data subroutine (continued)

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Cluster | LLR_ClusterRegionManagerDown | char*** | A NULL-terminated list of hostnames of machines that are configured to run a Region Manager and where the Region Manager is in the Down state. |
| Cluster | LLR_ClusterResourceManager | char** | The hostname of the machine currently running the Resource Manager daemon. |
| Cluster | LLR_ClusterScheddAvailable | int* | The count of available Schedd daemons in the cluster. |
| Cluster | LLR_ClusterScheddTotal | int* | The total number of Machines configured to be running a Schedd daemon and machines outside of the LoadLeveler configuration but known to be running a Schedd daemon. |
| Cluster | LLR_ClusterStartdAvailable | int* | The count of Startd daemons in the cluster which are currently either running jobs or able to run jobs (Avail, Idle, Busy). |
| Cluster | LLR_ClusterStartdTotal | int* | The total number of Machines configured to be running a Startd daemon and machines outside of the LoadLeveler configuration but known to be running a Startd daemon. |
| Cluster | LLR_ClusterTasksRunning | int* | The total number of tasks of all job steps running in the cluster. |

See "Understanding the Job object model" on page 231 for more information on the Job object model.

Table 29 lists the JOBS specifications for the **llr_get_data** subroutine.

Table 29. JOBS specifications for llr_get_data subroutine

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Adapter | LLR_AdapterName | char** | A pointer to a string containing the adapter name. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| AdapterReq | LLR_AdapterReqCollectiveGroups | int* | Requested collective group indexes of the HFI Adapter for the protocol. |
| AdapterReq | LLR_AdapterReqCommLevel | int* | A pointer to the integer indicating the adapter's communication level. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| AdapterReq | LLR_AdapterReqImmSendBuffers | int* | Requested immediate send buffers of the HFI Adapter for the protocol. |

*Table 29. JOBS specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| AdapterReq | LLR_AdapterReqInstances | int* | A pointer to an integer containing the requested adapter instances. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| AdapterReq | LLR_AdapterReqMode | char** | A pointer to a string containing the requested adapter mode (IP or US). Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| AdapterReq | LLR_AdapterReqProtocol | char** | A pointer to a string containing the requested adapter protocol. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| AdapterReq | LLR_AdapterReqRcxtBlocks | int* | A pointer to the integer indicating the number of rCxt blocks requested for the adapter usage. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| AdapterReq | LLR_AdapterReqTypeName | char** | A pointer to a string containing the requested adapter type. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| AdapterReq | LLR_AdapterReqUsage | int* | A pointer to the integer indicating the requested adapter usage. This integer will be one of the values defined in the Usage enum. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Class | LLR_ClassAsLimitHard64 | int64_t* | A pointer to a 64-bit integer indicating the **as** hard limit set by the user in the **as_limit** keyword. |
| Class | LLR_ClassAsLimitSoft64 | int64_t* | A pointer to a 64-bit integer indicating the **as** soft limit set by the user in the **as_limit** keyword. |
| Class | LLR_ClassCkptTimeHardLimit | int64_t* | A pointer to a 64-bit integer indicating the checkpoint time hard limit. |
| Class | LLR_ClassCkptTimeSoftLimit | int64_t* | A pointer to a 64-bit integer indicating the checkpoint time soft limit. |
| Class | LLR_ClassComment | char** | A pointer to a string containing the class comment. |
| Class | LLR_ClassCoreLimitHard | int64_t* | A pointer to a 64-bit integer indicating the core file hard limit. |
| Class | LLR_ClassCoreLimitSoft | int64_t* | A pointer to a 64-bit integer indicating the core file soft limit. |
| Class | LLR_ClassCpuLimitHard | int64_t* | A pointer to a 64-bit integer indicating the CPU hard limit. |
| Class | LLR_ClassCpuLimitSoft | int64_t* | A pointer to a 64-bit integer indicating the CPU soft limit. |

*Table 29. JOBS specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Class | LLR_ClassCpuStepLimitHard | int64_t* | A pointer to a 64-bit integer indicating the CPU hard limit. |
| Class | LLR_ClassCpuStepLimitSoft | int64_t* | A pointer to a 64-bit integer indicating the CPU soft limit. |
| Class | LLR_ClassDataLimitHard | int64_t* | A pointer to a 64-bit integer indicating the data hard limit. |
| Class | LLR_ClassDataLimitSoft | int64_t* | A pointer to a 64-bit integer indicating the data soft limit. |
| Class | LLR_ClassDefWallClockLimitHard | int64_t* | A pointer to a 64-bit integer indicating the default wall clock hard limit. |
| Class | LLR_ClassDefWallClockLimitSoft | int64_t* | A pointer to a 64-bit integer indicating the default wall clock soft limit. |
| Class | LLR_ClassName | char** | A pointer to a string containing the name of the class. |
| Class | LLR_ClassFileLimitHard | int64_t* | A pointer to a 64-bit integer indicating the file size hard limit. |
| Class | LLR_ClassFileLimitSoft | int64_t* | A pointer to a 64-bit integer indicating the file size soft limit. |
| Class | LLR_ClassLocksLimitHard | int64_t* | A pointer to a 64-bit integer indicating the **locks** hard limit set by the user in the **locks_limit** keyword. |
| Class | LLR_ClassLocksLimitSoft | int64_t* | A pointer to a 64-bit integer indicating the **locks** soft limit set by the user in the **locks_limit** keyword. |
| Class | LLR_ClassMaximumSlots | int* | A pointer to an integer indicating the total number of configured initiators. |
| Class | LLR_ClassMemlockLimitHard | int64_t* | A pointer to a 64-bit integer indicating the **memlock** hard limit set by the user in the **memlock_limit** keyword. |
| Class | LLR_ClassMemlockLimitSoft | int64_t* | A pointer to a 64-bit integer indicating the **memlock** soft limit set by the user in the **memlock_limit** keyword. |
| Class | LLR_ClassNofileLimitHard | int64_t* | A pointer to a 64-bit integer indicating the **nofile** hard limit set by the user in the **nofile_limit** keyword. |
| Class | LLR_ClassNofileLimitSoft | int64_t* | A pointer to a 64-bit integer indicating the **nofile** soft limit set by the user in the **nofile_limit** keyword. |
| Class | LLR_ClassNprocLimitHard | int64_t* | A pointer to a 64-bit integer indicating the **nproc** hard limit set by the user in the **nproc_limit** keyword. |
| Class | LLR_ClassNprocLimitSoft | int64_t* | A pointer to a 64-bit integer indicating the **nproc** soft limit set by the user in the **nproc_limit** keyword. |
| Class | LLR_ClassRssLimitHard | int64_t* | A pointer to a 64-bit integer indicating the resident set size hard limit. |
| Class | LLR_ClassRssLimitSoft | int64_t* | A pointer to a 64-bit integer indicating the resident set size soft limit. |
| Class | LLR_ClassStackLimitHard | int64_t* | A pointer to a 64-bit integer indicating the stack size hard limit. |
| Class | LLR_ClassStackLimitSoft | int64_t* | A pointer to a 64-bit integer indicating the stack size soft limit. |

*Table 29. JOBS specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Class | LLR_ClassStripingWithMinimumNetworks | int* | A pointer to a Boolean integer indicating that striping is allowed on nodes with more than half of the networks in the READY state. |
| Class | LLR_ClassWallClockLimitHard | int64_t* | A pointer to a 64-bit integer indicating the wall clock hard limit. |
| Class | LLR_ClassWallClockLimitSoft | int64_t* | A pointer to a 64-bit integer indicating the wall clock soft limit. |
| Credential | LLR_CredentialGid | int* | A pointer to an integer containing the UNIX gid of the user submitting the job. |
| Credential | LLR_CredentialGroupName | char** | A pointer to a string containing the UNIX group name of the user submitting the job. |
| Credential | LLR_CredentialUid | int* | A pointer to an integer containing the UNIX uid of the person submitting the job. |
| Credential | LLR_CredentialUserName | char** | A pointer to a string containing the user ID of the user submitting the job. |
| DispUsage | LLR_DispUsageEventUsageCount | int* | A pointer to an integer indicating the count of event usages. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageGetFirstEventUsage | llr_element_t* (EventUsage) | A pointer to the element associated with the first event usage. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageGetNextEventUsage | llr_element_t* (EventUsage) | A pointer to the element associated with the next event usage. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStarterIdrss64 | int64_t* | A pointer to a 64-bit integer indicating the amount of unshared memory in the data segment of a process. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStarterInblock64 | int64_t* | A pointer to a 64-bit integer indicating the number of times the file system performed input. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStarterIsrss64 | int64_t* | A pointer to a 64-bit integer indicating the unshared stack size. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStarterIxrss64 | int64_t* | A pointer to a 64-bit integer indicating the amount of memory used by the text segment that was also shared among other processes. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStarterMajflt64 | int64_t* | A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStarterMaxrss64 | int64_t* | A pointer to a 64-bit integer indicating the maximum resident set size utilized. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStarterMinflt64 | int64_t* | A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LLR_QUERY_HISTORY_FILE only. |

Table 29. JOBS specifications for llr_get_data subroutine (continued)

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| DispUsage | LLR_DispUsageStarterMsgrcv64 | int64_t* | A pointer to a 64-bit integer indicating the number of IPC messages received. Data is available from the LLR_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStarterMsgsnd64 | int64_t* | A pointer to a 64-bit integer indicating the number of IPC messages sent. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStarterNivcsw64 | int64_t* | A pointer to a 64-bit integer indicating the number of involuntary context switches. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStarterNsignals64 | int64_t* | A pointer to a 64-bit integer indicating the number of signals delivered. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStarterNswap64 | int64_t* | A pointer to a 64-bit integer indicating the number of times swapped out. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStarterNvcsw64 | int64_t* | A pointer to a 64-bit integer indicating the number of context switches due to voluntarily giving up processor. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStarterOublock64 | int64_t* | A pointer to a 64-bit integer indicating the number of times the file system performed output. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStarterSystemTime64 | int64_t* | A pointer to a 64-bit integer indicating the CPU system time. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStarterUserTime64 | int64_t* | A pointer to a 64-bit integer indicating the CPU user time. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStepIdrss64 | int64_t* | A pointer to a 64-bit integer indicating the amount of unshared memory in the data segment of a process. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStepInblock64 | int64_t* | A pointer to a 64-bit integer indicating the number of times the file system performed input. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStepIsrss64 | int64_t* | A pointer to a 64-bit integer indicating the unshared stack size. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStepIxrss64 | int64_t* | A pointer to a 64-bit integer indicating the amount of memory used by the text segment that was also shared among other processes. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStepMajflt64 | int64_t* | A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LLR_QUERY_HISTORY_FILE only. |

*Table 29. JOBS specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| DispUsage | LLR_DispUsageStepMaxrss64 | int64_t* | A pointer to a 64-bit integer indicating the maximum resident set size utilized. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStepMinflt64 | int64_t* | A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStepMsgrcv64 | int64_t* | A pointer to a 64-bit integer indicating the number of IPC messages received. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStepMsgsnd64 | int64_t* | A pointer to a 64-bit integer indicating the number of IPC messages sent. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStepNivcsw64 | int64_t* | A pointer to a 64-bit integer indicating the number of involuntary context switches. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStepNsignals64 | int64_t* | A pointer to a 64-bit integer indicating the number of signals delivered. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStepNswap64 | int64_t* | A pointer to a 64-bit integer indicating the number of times swapped out. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStepNvcsw64 | int64_t* | A pointer to a 64-bit integer indicating the number of context switches due to voluntarily giving up processor. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStepOublock64 | int64_t* | A pointer to a 64-bit integer indicating the number of times the file system performed output. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStepSystemTime64 | int64_t* | A pointer to a 64-bit integer indicating the CPU system time. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| DispUsage | LLR_DispUsageStepUserTime64 | int64_t* | A pointer to a 64-bit integer indicating the CPU user time. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageEventID | int* | A pointer to an integer indicating the event ID. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageEventName | char** | A pointer to a string containing the event name. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageEventTimestamp | int* | A pointer to an integer indicating the event timestamp. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStarterIdrss64 | int64_t* | A pointer to a 64-bit integer indicating the amount of unshared memory in the data segment of a process. Data is available from the LLR_QUERY_HISTORY_FILE only. |

*Table 29. JOBS specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| EventUsage | LLR_EventUsageStarterInblock64 | int64_t* | A pointer to a 64-bit integer indicating the number of times the file system performed input. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStarterIsrss64 | int64_t* | A pointer to a 64-bit integer indicating the unshared stack size. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStarterIxrss64 | int64_t* | A pointer to a 64-bit integer indicating the amount of memory used by the text segment that was also shared among other processes. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStarterMajflt64 | int64_t* | A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStarterMaxrss64 | int64_t* | A pointer to a 64-bit integer indicating the maximum resident set size utilized. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStarterMinflt64 | int64_t* | A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStarterMsgrcv64 | int64_t* | A pointer to a 64-bit integer indicating the number of IPC messages received. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStarterMsgsnd64 | int64_t* | A pointer to a 64-bit integer indicating the number of IPC messages sent. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStarterNivcsw64 | int64_t* | A pointer to a 64-bit integer indicating the number of involuntary context switches. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStarterNsignals64 | int64_t* | A pointer to a 64-bit integer indicating the number of signals delivered. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStarterNswap64 | int64_t* | A pointer to a 64-bit integer indicating the number of times swapped out. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStarterNvcsw64 | int64_t* | A pointer to a 64-bit integer indicating the number of context switches due to voluntarily giving up processor. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStarterOublock64 | int64_t* | A pointer to a 64-bit integer indicating the number of times the file system performed output. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStarterSystemTime64 | int64_t* | A pointer to a 64-bit integer indicating the CPU system time. Data is available from the LLR_QUERY_HISTORY_FILE only. |

Table 29. JOBS specifications for llr_get_data subroutine (continued)

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| EventUsage | LLR_EventUsageStarterUserTime64 | int64_t* | A pointer to a 64-bit integer indicating the CPU user time. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStepIdrss64 | int64_t* | A pointer to a 64-bit integer indicating the amount of unshared memory in the data segment of a process. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStepInblock64 | int64_t* | A pointer to a 64-bit integer indicating the number of times the file system performed input. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStepIsrss64 | int64_t* | A pointer to a 64-bit integer indicating the unshared stack size. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStepIxrss64 | int64_t* | A pointer to a 64-bit integer indicating the amount of memory used by the text segment that was also shared among other processes. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStepMajflt64 | int64_t* | A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStepMaxrss64 | int64_t* | A pointer to a 64-bit integer indicating the maximum resident set size utilized. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStepMinflt64 | int64_t* | A pointer to a 64-bit integer indicating the number of page faults. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStepMsgrcv64 | int64_t* | A pointer to a 64-bit integer indicating the number of IPC messages received. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStepMsgsnd64 | int64_t* | A pointer to a 64-bit integer indicating the number of IPC messages sent. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStepNivcsw64 | int64_t* | A pointer to a 64-bit integer indicating the number of involuntary context switches. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStepNsignals64 | int64_t* | A pointer to a 64-bit integer indicating the number of signals delivered. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStepNswap64 | int64_t* | A pointer to a 64-bit integer indicating the number of times swapped out. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStepNvcsw64 | int64_t* | A pointer to a 64-bit integer indicating the number of context switches due to voluntarily giving up processor. Data is available from the LLR_QUERY_HISTORY_FILE only. |

Table 29. JOBS specifications for llr_get_data subroutine (continued)

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| EventUsage | LLR_EventUsageStepOublock64 | int64_t* | A pointer to a 64-bit integer indicating the number of times the file system performed output. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStepSystemTime64 | int64_t* | A pointer to a 64-bit integer indicating the CPU system time. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| EventUsage | LLR_EventUsageStepUserTime64 | int64_t* | A pointer to a 64-bit integer indicating the CPU user time. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| Job | LLR_JobCredential | llr_element_t* (Credential) | A pointer to the element associated with the job credential. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Job | LLR_JobGetFirstStep | llr_element_t* (Step) | A pointer to the element associated with the first step of the job, to be used in subsequent **llr_get_data** calls. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Job | LLR_JobGetNextStep | llr_element_t* (Step) | A pointer to the element associated with the next step. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Job | LLR_JobJobManager | char** | A pointer to a string containing the name of the machine that the job manager for this job is running on. |
| Job | LLR_JobJobQueueKey | int* | A pointer to an integer indicating the key used to write the job to the spool. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Job | LLR_JobMode | int* | A pointer to an integer indicating the job mode, which can be INTERACTIVE_JOB or BATCH_JOB. |
| Job | LLR_JobName | char** | A pointer to a character string containing the job name. |
| Job | LLR_JobStepCount | int* | A pointer to an integer indicating the number of steps connected to the job. |
| Job | LLR_JobSubmitHost | char** | A pointer to a character string containing the name of the host machine from which the job was submitted. |
| Job | LLR_JobSubmitTime | time_t* | A pointer to the **time_t** structure indicating when the job was submitted. |
| Job | LLR_JobVersionNum | int* | A pointer to an integer indicating the job's version number. |
| Machine | LLR_MachineName | char** | A pointer to a string containing the machine name. |
| MachUsage | LLR_MachUsageDispUsageCount | int* | A pointer to an integer indicating the count of dispatch usages. Data is available from the LLR_QUERY_HISTORY_FILE only. |

Table 29. JOBS specifications for llr_get_data subroutine (continued)

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| MachUsage | LLR_MachUsageGetFirstDispUsage | llr_element_t* (DispUsage) | A pointer to the element associated with the first dispatch usage. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| MachUsage | LLR_MachUsageGetNextDispUsage | llr_element_t* (DispUsage) | A pointer to the element associated with the next dispatch usage. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| MachUsage | LLR_MachUsageMachineName | char** | A pointer to a string containing the machine name. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| MachUsage | LLR_MachUsageMachineSpeed | double* | A pointer to a double containing the machine speed. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| NetworkUsage | LLR_NetworkUsageMode | char** | A pointer to a string containing the adapter usage mode of IP or US. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| NetworkUsage | LLR_NetworkUsageNetworkId | uint64_t* | A pointer to an unsigned 64–bit integer indicating the network ID. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| NetworkUsage | LLR_NetworkUsageProtocol | char** | A pointer to a string containing the requested adapter protocol. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| NetworkUsage | LLR_NetworkUsageRcxtBlocks | int* | A pointer to the integer indicating the number of rCxt blocks associated with each window. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| NetworkUsage | LLR_NetworkUsageWindows | int* | A pointer to the integer indicating the number of windows associated with each instance. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Node | LLR_NodeGetFirstResourceRequirement | llr_element_t* (ResourceReq) | A pointer to the element associated with the first resource requirement. |
| Node | LLR_NodeGetFirstTask | llr_element_t* (Task) | A pointer to the element associated with the first task for this node. |
| Node | LLR_NodeGetNextResourceRequirement | llr_element_t* (ResourceReq) | A pointer to the element associated with the next resource requirement. |
| Node | LLR_NodeGetResourceRequirementList | llr_data_list_t * | The resource requirement iterator. |
| Node | LLR_NodeGetNextTask | llr_element_t* (Task) | A pointer to the element associated with the next task for this node. |
| Node | LLR_NodeInitiatorCount | int* | A pointer to an integer indicating the number of tasks running on the node. |

*Table 29. JOBS specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Node | LLR_NodeMaxInstances | int* | A pointer to an integer indicating the maximum number of machines requested. |
| Node | LLR_NodeMinInstances | int* | A pointer to an integer indicating the minimum number of machines requested. |
| Node | LLR_NodeRequirements | char** | A pointer to a string containing the node requirements. |
| Node | LLR_NodeTaskCount | int* | A pointer to an integer indicating the different types of tasks running on the node. |
| ResourceReq | LLR_ResourceRequirementName | char** | A pointer to a string containing the resource requirement name. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| ResourceReq | LLR_ResourceRequirementValue | int* | A pointer to an integer indicating the value of the resource requirement. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| ResourceReq | LLR_ResourceRequirementValue64 | int64_t* | A pointer to a 64-bit integer indicating the value of the resource requirement. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepAccountNumber | char** | A pointer to a string containing the account number specified by the user submitting the job. |
| Step | LLR_StepAcctKey | int64_t* | A pointer to a 64-bit integer that can be used to identify all of the AIX accounting records for the job step. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| Step | LLR_StepAdjustWCKLimit | int* | The wall clock time adjusted by LoadLeveler for the job. |
| Step | LLR_StepAsLimitHard64 | int64_t* | A pointer to a 64-bit integer indicating the **as** hard limit set by the user in the **as_limit** keyword. |
| Step | LLR_StepAsLimitSoft64 | int64_t* | A pointer to a 64-bit integer indicating the **as** soft limit set by the user in the **as_limit** keyword. |
| Step | LLR_StepAvgPower | double* | A pointer to a double indicating the average power in kilowatts consumed by the job step. |
| Step | LLR_StepBgBlockRequested | char** | A pointer to a string containing the ID of the Blue Gene block requested for the job step. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |

*Table 29. JOBS specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Step | LLR_StepBgConnectivityRequested | int** | A pointer to an array indicating the requested connectivity per dimension for the Blue Gene job (llr_BGQConn). Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepBgErrorText | char** | A pointer to a string containing the error text for the Blue Gene job record in the Blue Gene database. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepBgShapeRequested | int** | A pointer to an array indicating the shape of Blue Gene compute nodes requested for the job step. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepBgSizeRequested | int* | A pointer to an integer indicating the number of Blue Gene compute nodes requested for the job step. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepBlocking | int* | A pointer to an integer representing blocking as specified by the user in the job command file.<br>• Returns -1 if unlimited is specified<br>• Returns 0 if blocking is unspecified<br>Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepBulkXfer | int* | A pointer to an integer that is set to 1 if the step requested bulk transfer and 0 if it did not. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepCheckpointable | int* | A pointer to an integer indicating if checkpointing was enabled via the **checkpoint** keyword (0=disabled, 1=enabled). Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepCheckpointing | int* | A pointer to an integer indicating that a checkpoint is currently being taken for the step. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |

*Table 29. JOBS specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Step | LLR_StepCkptAccumTime | int* | A pointer to an integer indicating the amount of accumulated time, in seconds, that the job step has spent checkpointing. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepCkptExecuteDirectory | char** | A pointer to a string containing the directory where the job step's executable will be saved. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepCkptFailStartTime | time_t* | A pointer to a **time_t** structure indicating the start time of the last unsuccessful checkpoint. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepCkptGoodElapseTime | int* | A pointer to an integer indicating the amount of time, in seconds, it took for the last successful checkpoint to complete. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepCkptGoodStartTime | time_t* | A pointer to a **time_t** structure indicating the start time of the last successful checkpoint. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepCkptRestart | int* | A pointer to an integer indicating the value specified by the user for the **restart_from_ckpt** keyword (0= no, 1= yes). Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepCkptRestartSameNodes | int* | A pointer to a string indicating the value specified by the user for the **restart_on_same_nodes** keyword (0= no, 1= yes). Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepCkptSubDir | char** | A pointer to a string containing the directory that contains checkpoint information for the job step. |
| Step | LLR_StepCkptTimeHardLimit | int* | A pointer to an integer indicating the hard limit set by the user in the **ckpt_time_limit** keyword. |
| Step | LLR_StepCkptTimeHardLimit64 | int64_t* | A pointer to a 64-bit integer indicating the hard limit set by the user in the **ckpt_time_limit** keyword. |
| Step | LLR_StepCkptTimeSoftLimit | int* | A pointer to an integer indicating the soft limit set by the user in **ckpt_time_limit** keyword. |

*Table 29. JOBS specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Step | LLR_StepCkptTimeSoftLimit64 | int64_t* | A pointer to a 64-bit integer indicating the soft limit set by the user in **ckpt_time_limit** keyword. |
| Step | LLR_StepComment | char** | A pointer to a string indicating the comment specified by the user submitting the job. |
| Step | LLR_StepCompletionCode | int* | A pointer to an integer indicating the completion code of the step. |
| Step | LLR_StepCompletionDate | time_t* | A pointer to a **time_t** structure indicating the completion date of the step. |
| Step | LLR_StepCoreLimitHard | int* | A pointer to an integer indicating the core hard limit set by the user in the **core_limit** keyword. |
| Step | LLR_StepCoreLimitHard64 | int64_t* | A pointer to a 64-bit integer indicating the core hard limit set by the user in the **core_limit** keyword. |
| Step | LLR_StepCoreLimitSoft | int* | A pointer to an integer indicating the core soft limit set by the user in the **core_limit** keyword. |
| Step | LLR_StepCoreLimitSoft64 | int64_t* | A pointer to a 64-bit integer indicating the core soft limit set by the user in the **core_limit** keyword. |
| Step | LLR_StepCoschedule | int* | A pointer to an integer indicating if a job step is coscheduled set by the user in the **coschedule** keyword. |
| Step | LLR_StepCPI | double* | A pointer to a double indicating the number of cycles per instruction while running job step. |
| Step | LLR_StepCpuLimitHard | int* | A pointer to an integer indicating the CPU hard limit set by the user in the **cpu_limit** keyword. |
| Step | LLR_StepCpuLimitHard64 | int64_t* | A pointer to a 64-bit integer indicating the CPU hard limit set by the user in the **cpu_limit** keyword. |
| Step | LLR_StepCpuLimitSoft | int* | A pointer to an integer indicating the CPU soft limit set by the user in the **cpu_limit** keyword. |
| Step | LLR_StepCpuLimitSoft64 | int64_t* | A pointer to a 64-bit integer indicating the CPU soft limit set by the user in the **cpu_limit** keyword. |
| Step | LLR_StepCpusPerCore | int* | A pointer to an integer indicating the CPUs per processor core requested by the job using the **cpus_per_core** keyword. |
| Step | LLR_StepCpuStepLimitHard | int* | A pointer to an integer indicating the CPU step hard limit set by the user in the **job_cpu_limit** keyword. |
| Step | LLR_StepCpuStepLimitHard64 | int64_t* | A pointer to a 64-bit integer indicating the CPU step hard limit set by the user in the **job_cpu_limit** keyword. |
| Step | LLR_StepCpuStepLimitSoft | int* | A pointer to an integer indicating the CPU step soft limit set by the user in the **job_cpu_limit** keyword. |
| Step | LLR_StepCpuStepLimitSoft64 | int64_t* | A pointer to a 64-bit integer indicating the CPU step soft limit set by the user in the **job_cpu_limit** keyword. |

*Table 29. JOBS specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Step | LLR_StepDataLimitHard | int* | A pointer to an integer indicating the data hard limit set by the user in the **data_limit** keyword. |
| Step | LLR_StepDataLimitHard64 | int64_t* | A pointer to a 64-bit integer indicating the data hard limit set by the user in the **data_limit** keyword. |
| Step | LLR_StepDataLimitSoft | int* | A pointer to an integer indicating the data soft limit set by the user in the **data_limit** keyword. |
| Step | LLR_StepDataLimitSoft64 | int64_t* | A pointer to a 64-bit integer indicating the data soft limit set by the user in the **data_limit** keyword. |
| Step | LLR_StepDependency | char** | A pointer to a string containing the step dependency value. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepDispatchTime | time_t* | A pointer to a **time_t** structure indicating the time the negotiator dispatched the job. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepEligibilityTime | time_t* | A pointer to a **time_t** structure indicating the last time the job became eligible for negotiator dispatch. Data is available from LL_CM, LL_SCHEDD, and LL_HISTORY_FILE. |
| Step | LLR_StepEnergyPolicyTag | char* | A pointer to a string containing the energy policy tag the job used. |
| Step | LLR_StepEnvironment | char** | A pointer to a string containing the environment variables set by the user in the executable. Data is available from LLR_QUERY_JOBMGR and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepErrorFile | char** | A pointer to a string containing the standard error file name used by the executable. |
| Step | LLR_StepExecSize | int* | A pointer to an integer indicating the executable size. |
| Step | LLR_StepFileLimitHard | int* | A pointer to an integer indicating the file hard limit set by the user in the **file_limit** keyword. |
| Step | LLR_StepFileLimitHard64 | int64_t* | A pointer to a 64-bit integer indicating the file hard limit set by the user in the **file_limit** keyword. |
| Step | LLR_StepFileLimitSoft | int* | A pointer to an integer indicating the file soft limit set by the user in the **file_limit** keyword. |
| Step | LLR_StepFileLimitSoft64 | int64_t* | A pointer to a 64-bit integer indicating the file soft limit set by the user in the **file_limit** keyword. |

Table 29. JOBS specifications for llr_get_data subroutine (continued)

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Step | LLR_StepGetFirstAdapterReq | llr_element_t* (AdapterReq) | A pointer to the element associated with the first adapter requirement. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepGetFirstMachine | llr_element_t* (Machine) | A pointer to the element associated with the first machine in the step. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_STARTD. |
| Step | LLR_StepGetFirstMachUsage | llr_element_t* (MachUsage) | A pointer to the element associated with the first machine usage in the list of machine usages where a single machine usage corresponds to any machine that was ever assigned to the step. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| Step | LLR_StepGetFirstNetworkUsage | llr_element_t* (Network Usage) | A pointer to the element associated with the first network usage of the step. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepGetFirstNode | llr_element_t* (Node) | A pointer to the element associated with the first node of the step. |
| Step | LLR_StepGetFirstStepResourceRequirement | llr_element_t * (ResourceReq) | A pointer to the element associated with the first resource requirement. |
| Step | LLR_StepGetMasterTask | llr_element_t* (Task) | A pointer to the element associated with the master task of the step. |
| Step | LLR_StepGetNextAdapterReq | llr_element_t* (AdapterReq) | A pointer to the element associated with the next adapter requirement. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepGetNextMachine | llr_element_t* (Machine) | A pointer to the element associated with the next machine of the step. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_STARTD. |
| Step | LLR_StepGetNextMachUsage | llr_element_t* (MachUsage) | A pointer to the element associated with the next machine usage in the list of machine usages where a single machine usage corresponds to any machine that was ever assigned to the step. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| Step | LLR_StepGetNextNetworkUsage | llr_element_t* (Network Usage) | A pointer to the element associated with the next network usage of the step. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepGetNextNode | llr_element_t* (Node) | A pointer to the element associated with the next node of the step. |

*Table 29. JOBS specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Step | LLR_StepGetNextStepResourceRequirement | llr_element_t * (ResourceReq) | A pointer to the element associated with the next resource requirement. |
| Step | LLR_StepGetStepResourceRequirementList | llr_data_list_t * | The resource requirement iterator. |
| Step | LLR_StepHostList | char*** | A pointer to an array containing the list of hosts in the **host.list** file associated with the step. The array ends with a null string. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepID | char** | A pointer to a string containing the ID of the step. |
| Step | LLR_StepImageSize | int* | A pointer to an integer indicating the image size of the executable. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepImageSize64 | int64_t* | A pointer to a 64-bit integer indicating the image size of the executable. Data is available from LR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepInputFile | char** | A pointer to a string containing the standard input file name used by the executable. |
| Step | LLR_StepIslandCount | int** | An array containing two integers corresponding to the **island_count** job command file keyword. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOB_MANAGER, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepIwd | char** | A pointer to a string containing the initial working directory name used by the executable. |
| Step | LLR_StepJobClass | char** | A pointer to a string containing the class of the step. |
| Step | LLR_StepLargePage | char** | A pointer to a string containing the Large Page level of support associated with the job step. |
| Step | LLR_StepLoadLevelerGroup | char** | A pointer to a string containing the name of the LoadLeveler group specified by the step. |
| Step | LLR_StepLocksLimitHard64 | int64_t* | A pointer to a 64-bit integer indicating the **locks** hard limit set by the user in the **locks_limit** keyword. |
| Step | LLR_StepLocksLimitSoft64 | int64_t* | A pointer to a 64-bit integer indicating the **locks** soft limit set by the user in the **locks_limit** keyword. |

*Table 29. JOBS specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Step | LLR_StepMachineCount | int* | A pointer to an integer indicating the number of machines assigned to the step when it is in a running-like state. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_STARTD. |
| Step | LLR_StepMachUsageCount | int* | A pointer to an integer indicating the number of machine usage objects corresponding to the set of machines that were ever assigned to the step. Data is available from the LLR_QUERY_HISTORY_FILE only. |
| Step | LLR_StepMaxPower | double* | A pointer to a double indicating the maximum power level in kilowatts measured while running the job step. |
| Step | LLR_StepMaxProtocolInstances | int* | A pointer to an integer indicating the largest number of instances allowed on the network statement. |
| Step | LLR_StepMcmAffinityOptions | char** | A pointer to a string containing the MCM affinity options. |
| Step | LLR_StepMemlockLimitHard64 | int64_t* | A pointer to a 64-bit integer indicating the **memlock** hard limit set by the user in the **memlock_limit** keyword. |
| Step | LLR_StepMemlockLimitSoft64 | int64_t* | A pointer to a 64-bit integer indicating the **memlock** soft limit set by the user in the **memlock_limit** keyword. |
| Step | LLR_StepMessages | char** | A pointer to a string containing a list of messages from LoadLeveler. The messages are updated right after the step is considered for scheduling by LoadLeveler. The data is available from the LLR_QUERY_RESOURCE_MANAGER only. |
| Step | LLR_StepMetaClusterJob | int* | A pointer to an integer containing the value associated with the **metacluster_job** keyword (1 if **metacluster_job** keyword is set to **yes**). |
| Step | LLR_StepMetaClusterJobID | int* | A pointer to an integer indicating the MetaCluster job ID that LoadLeveler has assigned to the job step. |
| Step | LLR_StepMetaClusterPoeHostname | char** | A pointer to a string containing the host name where the master task of a parallel MetaCluster job step is running. |
| Step | LLR_StepMetaClusterPoePmdPhysnet | char** | A pointer to a string containing the physnet of a MetaCluster job step. |
| Step | LLR_StepName | char** | A pointer to a string containing the name of the step. |
| Step | LLR_StepNodeCount | int* | A pointer to an integer indicating the number of node objects associated with the step. |
| Step | LLR_StepNodeUsage | int* | A pointer to an integer indicating the node usage specified by the user (SHARED or NOT_SHARED). The value returned is in the enum Usage. |

*Table 29. JOBS specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Step | LLR_StepNofileLimitHard64 | int64_t* | A pointer to a 64-bit integer indicating the **nofile** hard limit set by the user in the **nofile_limit** keyword. |
| Step | LLR_StepNofileLimitSoft64 | int64_t* | A pointer to a 64-bit integer indicating the **nofile** soft limit set by the user in the **nofile_limit** keyword. |
| Step | LLR_StepNprocLimitHard64 | int64_t* | A pointer to a 64-bit integer indicating the **nproc** hard limit set by the user in the **nproc_limit** keyword. |
| Step | LLR_StepNprocLimitSoft64 | int64_t* | A pointer to a 64-bit integer indicating the **nproc** soft limit set by the user in the **nproc_limit** keyword. |
| Step | LLR_StepOutputFile | char** | A pointer to a character string containing the standard output file name used by the executable. |
| Step | LLR_StepParallelMode | int* | A pointer to an integer indicating the mode of the step. |
| Step | LLR_StepPerfDegradation | int* | The job performance degradation percentage that is used. |
| Step | LLR_StepPowerConsumption | double* | A pointer to a double indicating the power consumption of the job. |
| Step | LLR_StepPowerSavPercent | int* | The energy saving percentage required. |
| Step | LLR_StepPredictedPower | double* | The predicted power consumption. |
| Step | LLR_StepPredictedTime | int* | The predicted elapsed time. |
| Step | LLR_StepPreemptable | int* | A pointer to an integer indicating whether the job step is preemptable. The integer is set to 0 if the job step is not preemptable and is set to 1 if the job step is preemptable. |
| Step | LLR_StepRestart | int* | A pointer to an integer representing whether restart is specified as yes (default value) or no by the user in the job command file.<br>• 1 indicates yes<br>• 0 indicates no |
| Step | LLR_StepRunningFrequency | int* | A pointer to an integer representing the current running frequency used by the job. |
| Step | LLR_StepTopologyRequire | char** | The type of topology required by the step corresponding to the **node_topology** job command file keyword. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOB_MANAGER, and LLR_QUERY_HISTORY_FILE. |
| Job | LLR_StepRsetName | char** | A pointer to a character string containing the RSet name used by the step. If no RSet name was used, the value is NULL. |
| Step | LLR_StepRssLimitHard | int* | A pointer to an integer indicating the RSS hard limit set by the user in the **rss_limit** keyword. |
| Step | LLR_StepRssLimitHard64 | int64_t* | A pointer to a 64-bit integer indicating the RSS hard limit set by the user in the **rss_limit** keyword. |

Table 29. JOBS specifications for llr_get_data subroutine (continued)

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Step | LLR_StepRssLimitSoft | int* | A pointer to an integer indicating the RSS soft limit set by the user in the **rss_limit** keyword. |
| Step | LLR_StepRssLimitSoft64 | int64_t* | A pointer to a 64-bit integer indicating the RSS soft limit set by the user in the **rss_limit** keyword. |
| Step | LLR_StepShell | char** | A pointer to a character string containing the shell name used by the executable. |
| Step | LLR_StepSMTRequired | int* | A pointer to an integer indicating the required SMT state. The value returned is in the enum SMTRequiredState. |
| Step | LLR_StepStackLimitHard | int* | A pointer to an integer indicating the stack hard limit set by the user in the **stack_limit** keyword. |
| Step | LLR_StepStackLimitHard64 | int64_t* | A pointer to a 64-bit integer indicating the stack hard limit set by the user in the **stack_limit** keyword. |
| Step | LLR_StepStackLimitSoft | int* | A pointer to an integer indicating the stack soft limit set by the user in the **stack_limit** keyword. |
| Step | LLR_StepStackLimitSoft64 | int64_t* | A pointer to a 64-bit integer indicating the stack soft limit set by the user in the **stack_limit** keyword. |
| Step | LLR_StepStartCount | int* | A pointer to an integer indicating the number of times the step has been started. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStartDate | time_t* | A pointer to a **time_t** structure indicating the value the user specified in the **startdate** keyword. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStarterIdrss64 | int64_t* | A pointer to a 64-bit integer indicating the amount of unshared memory in the data segment of a process. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStarterInblock64 | int64_t* | A pointer to a 64-bit integer indicating the number of times the file system performed input. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStarterIsrss64 | int64_t* | A pointer to a 64-bit integer indicating the unshared stack size. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |

*Table 29. JOBS specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Step | LLR_StepStarterIxrss64 | int64_t* | A pointer to a 64-bit integer indicating the amount of memory used by the text segment that was also shared among other processes. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStarterMajflt64 | int64_t* | A pointer to a 64-bit integer indicating the number of page faults. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStarterMaxrss64 | int64_t* | A pointer to a 64-bit integer indicating the maximum resident set size utilized. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStarterMinflt64 | int64_t* | A pointer to a 64-bit integer indicating the number of page faults. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStarterMsgrcv64 | int64_t* | A pointer to a 64-bit integer indicating the number of IPC messages received. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStarterMsgsnd64 | int64_t* | A pointer to a 64-bit integer indicating the number of IPC messages sent. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStarterNivcsw64 | int64_t* | A pointer to a 64-bit integer indicating the number of involuntary context switches. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStarterNsignals64 | int64_t* | A pointer to a 64-bit integer indicating the number of signals delivered. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStarterNswap64 | int64_t* | A pointer to a 64-bit integer indicating the number of times swapped out. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStarterNvcsw64 | int64_t* | A pointer to a 64-bit integer indicating the number of context switches due to voluntarily giving up processor. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |

*Table 29. JOBS specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Step | LLR_StepStarterOublock64 | int64_t* | A pointer to a 64-bit integer indicating the number of times the file system performed output. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStarterSystemTime64 | int64_t* | A pointer to a 64-bit integer indicating the CPU system time. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStarterUserTime64 | int64_t* | A pointer to a 64-bit integer indicating the CPU user time. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStartTime | time_t* | A pointer to a **time_t** structure indicating the time at which the starter process for the job started. Data is available from LLR_QUERY_JOBMGR and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepState | int* | A pointer to an integer indicating the state of the Step (Idle, Pending, Starting, and so on). The value returned is in the StepState enum. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStepIdrss64 | int64_t* | A pointer to a 64-bit integer indicating the amount of unshared memory in the data segment of a process. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStepInblock64 | int64_t* | A pointer to a 64-bit integer indicating the number of times the file system performed input. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStepIsrss64 | int64_t* | A pointer to a 64-bit integer indicating the unshared stack size. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStepIxrss64 | int64_t* | A pointer to a 64-bit integer indicating the amount of memory used by the text segment that was also shared among other processes. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStepMajflt64 | int64_t* | A pointer to a 64-bit integer indicating the number of page faults. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |

Table 29. JOBS specifications for llr_get_data subroutine (continued)

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Step | LLR_StepStepMaxrss64 | int64_t* | A pointer to a 64-bit integer indicating the maximum resident set size utilized. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStepMinflt64 | int64_t* | A pointer to a 64-bit integer indicating the number of page faults. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStepMsgrcv64 | int64_t* | A pointer to a 64-bit integer indicating the number of IPC messages received. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStepMsgsnd64 | int64_t* | A pointer to a 64-bit integer indicating the number of IPC messages sent. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStepNivcsw64 | int64_t* | A pointer to a 64-bit integer indicating the number of involuntary context switches. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStepNsignals64 | int64_t* | A pointer to a 64-bit integer indicating the number of signals delivered. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStepNswap64 | int64_t* | A pointer to a 64-bit integer indicating the number of times swapped out. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStepNvcsw64 | int64_t* | A pointer to a 64-bit integer indicating the number of context switches due to voluntarily giving up processor. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStepOublock64 | int64_t* | A pointer to a 64-bit integer indicating the number of times the file system performed output. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStepSystemTime64 | int64_t* | A pointer to a 64-bit integer indicating the CPU system time. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepStepUserTime64 | int64_t* | A pointer to a 64-bit integer indicating the CPU user time. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |

*Table 29. JOBS specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Step | LLR_StepTaskAffinity | char** | A pointer to a string containing the task affinity requirement of the job requested with the **task_affinity** keyword. |
| Step | LLR_StepTaskGeometry | char** | A pointer to a string containing the values specified in the task_geometry statement by the user in the job command file. The syntax is the same as specified in the statement , {(task id, task id, ...) (task id, task id, ...) ...}. If unspecified, a null string is returned. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepTaskInstanceCount | int* | A pointer to an integer indicating the number of task instances in the step. Data is available from LLR_QUERY_JOBMGR, LLR_QUERY_STARTD, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepTasksPerNodeRequested | int* | A pointer to an integer representing the tasks per node specified by the user in the job command file. If unspecified, the integer will contain a 0. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepTotalNodesRequested | char** | A pointer to a string containing the values specified by the user in the job command file node statement. The syntax is the same as specified in the statement, [min],[max], where min contains the minimum number of nodes requested and max contains the maximum nodes requested. If unspecified, a null string is returned. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepTotalRcxtBlocks | int* | A pointer to an integer indicating the total number of rCxt blocks application needs. |
| Step | LLR_StepTotalTasksRequested | int* | A pointer to an integer representing the total tasks specified by the user in the job command file. If unspecified, the integer will contain a 0. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| Step | LLR_StepUserRcxtBlocks | int* | A pointer to an integer indicating the number of user rCxt blocks application requests. |
| Step | LLR_StepWallClockLimitHard | int* | A pointer to an integer indicating the wall clock hard limit set by the user in the **wall_clock_limit** keyword. |
| Step | LLR_StepWallClockLimitHard64 | int64_t* | A pointer to a 64-bit integer indicating the wall clock hard limit set by the user in the **wall_clock_limit** keyword. |

*Table 29. JOBS specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Step | LLR_StepWallClockLimitSoft | int* | A pointer to an integer indicating the wall clock soft limit set by the user in the **wall_clock_limit** keyword. |
| Step | LLR_StepWallClockLimitSoft64 | int64_t* | A pointer to a 64-bit integer indicating the wall clock soft limit set by the user in the **wall_clock_limit** keyword. |
| Step | LLR_StepWallClockUsed | int* | A pointer to an integer that is the number of seconds of elapsed time for this step. |
| Task | LLR_TaskExecutable | char** | A pointer to a string containing the name of the executable. |
| Task | LLR_TaskExecutableArguments | char** | A pointer to a string containing the arguments passed by the user in the executable. |
| Task | LLR_TaskGetFirstResourceRequirement | llr_element_t* (ResourceReq) | A pointer to the element associated with the first resource requirement. |
| Task | LLR_TaskGetFirstTaskInstance | llr_element_t* (TaskInstance) | A pointer to the element associated with the first task instance. |
| Task | LLR_TaskGetNextResourceRequirement | llr_element_t* (ResourceReq) | A pointer to the element associated with the next resource requirement. |
| Task | LLR_TaskGetNextTaskInstance | llr_element_t* (TaskInstance) | A pointer to the element associated with the next task instance. |
| Task | LLR_TaskGetResourceRequirementList | llr_data_list_t * | The resource requirement iterator. |
| Task | LLR_TaskIsMaster | int* | A pointer to an integer, where 1 indicates master task. |
| Task | LLR_TaskTaskInstanceCount | int* | A pointer to an integer indicating the number of task instances. |
| TaskInstance | LLR_TaskInstanceCpuList | int* | A pointer to the integer indicating the number of CPUs used by a given task instance object. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| TaskInstance | LLR_TaskInstanceMachine | llr_element_t* (Machine) | A pointer to the element associated with the machine object. |
| TaskInstance | LLR_TaskInstanceMachineAddress | char** | A pointer to a string containing the IP address of the machine assigned to a task. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |
| TaskInstance | LLR_TaskInstanceMachineName | char** | A pointer to the string indicating the machine assigned to a task. |
| TaskInstance | LLR_TaskInstanceTaskID | int* | A pointer to the integer indicating the task ID. Data is available from LLR_QUERY_RESOURCE_MANAGER, LLR_QUERY_JOBMGR, and LLR_QUERY_HISTORY_FILE. |

See "Understanding the LLR_JOBQ_SUMMARY_QUERY object model" on page 232 for more information on the LLR_JOBQ_SUMMARY_QUERY object model.

Table 30 on page 263 lists the LLR_JOBQ_SUMMARY_QUERY specifications for the **llr_get_data** subroutine:

*Table 30. JOBQ_SUMMARY specifications for llr_get_data subroutine*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| JobSummary | LLR_JobSummaryHeldCount | int* | The number of job steps in Held, Deferred, or NotQueued state. |
| JobSummary | LLR_JobSummaryName | char** | The name of the user, group, or class. |
| JobSummary | LLR_JobSummaryPendingCount | int* | The number of job steps in a Pending state, including Pending, Starting, Preempt Pending, and Resume Pending. |
| JobSummary | LLR_JobSummaryPreemptedCount | int* | The number of job steps in the Preempted state. |
| JobSummary | LLR_JobSummaryRunningCount | int* | The number of job steps in Running state. |
| JobSummary | LLR_JobSummaryWaitingCount | int* | The number of job steps in Idle state. |

See "Understanding the Machine object model" on page 233 for more information on the Machine object model.

Table 31 lists the MACHINES specifications for the **llr_get_data** subroutine:

*Table 31. MACHINES specifications for llr_get_data subroutine*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Adapter | LLR_AdapterAvailWindowCount | int* | A pointer to an integer indicating the number of adapter windows not in use. |
| Adapter | LLR_AdapterCommInterface | int* | Contains the adapter's communication interface. |
| Adapter | LLR_AdapterInterfaceAddress | char** | A pointer to a string containing the adapter's interface IP address. |
| Adapter | LLR_AdapterInterfaceNetmask | char** | A pointer to a string containing the netmask of an adapter. |
| Adapter | LLR_AdapterLmc | int* | A pointer to an integer indicating the logical mask on the adapter. |
| Adapter | LLR_AdapterMaxWindowSize64 | uint64_t* | A pointer to an unsigned 64-bit integer indicating the maximum allocatable adapter window memory. |
| Adapter | LLR_AdapterMCMId | int* | A pointer to an integer indicating the MCM ID for the adapter. |
| Adapter | LLR_AdapterMemory64 | uint64_t* | A pointer to an unsigned 64-bit integer indicating the amount of total adapter memory. |
| Adapter | LLR_AdapterMinWindowSize64 | int* | A pointer to the integer indicating the minimum allocatable adapter window memory. |
| Adapter | LLR_AdapterName | char** | A pointer to a string containing the adapter name. |
| Adapter | LLR_AdapterPortNumber | int* | A pointer to the integer indicating the port number of the adapter. |

*Table 31. MACHINES specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Adapter | LLR_AdapterRcxtBlocks | int* | A pointer to the integer indicating the number of rCxt blocks available on an adapter. |
| Adapter | LLR_AdapterTotalWindowCount | int* | A pointer to the integer indicating the number of windows on the adapter. |
| Adapter | LLR_AdapterWindowList | int** | A pointer to an array indicating window numbers for the adapter. LLR_AdapterTotalWindowCount indicates the size of this array. If the adapter has no windows, LLR_AdapterTotalWindowCount is zero and LLR_AdapterWindowList is null. |
| Machine | LLR_MachineAdapterList | char*** | A pointer to an array containing a list of the types of adapters associated with the machine. The array ends with a NULL string. |
| Machine | LLR_MachineArchitecture | char** | A pointer to a string containing the machine architecture. The string may result in "???" if a query is made before the associated records are updated with their actual values by the appropriate startd daemons. |
| Machine | LLR_MachineConfigTimeStamp | int* | A pointer to an integer containing the date and time value of the last configuration or reconfiguration. |
| Machine | LLR_MachineConfiguredClassList | char*** | A pointer to an array containing the initiators on the machine. The array ends with a NULL string. |
| Machine | LLR_MachineContinueExpr | char** | A pointer to a string containing the machine's continue control expression. |
| Machine | LLR_MachineCpuList | int* | A pointer to an integer containing the list of CPUs on the machine. |
| Machine | LLR_MachineCPUs | int* | A pointer to an integer containing the number of CPUs on the machine. |
| Machine | LLR_MachineDisk | int* | A pointer to an integer indicating the disk space in KBs in the machine's execute directory. |
| Machine | LLR_MachineDisk64 | int64_t* | A pointer to a 64-bit integer indicating the disk space in KBs in the machine's execute directory. |
| Machine | LLR_MachineFeatureList | char*** | A pointer to an array containing the features defined on the machine. The array ends with a NULL string. |
| Machine | LLR_MachineFreeRealMemory | int* | A pointer to an integer indicating the amount of free real memory in MBs on the machine. |
| Machine | LLR_MachineFreeRealMemory64 | int64_t* | A pointer to a 64-bit integer indicating the amount of free real memory in MBs on the machine. |

*Table 31. MACHINES specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Machine | LLR_MachineGetFirstAdapter | llr_element_t* (Adapter) | A pointer to the element associated with the machine's first adapter. |
| Machine | LLR_MachineGetFirstMCM | llr_element_t* (MCM) | A pointer to the element associated with the machine's first MCM. |
| Machine | LLR_MachineGetFirstResource | llr_element_t* (Resource) | A pointer to the element associated with the machine's first resource. |
| Machine | LLR_MachineGetNextAdapter | llr_element_t* (Adapter) | A pointer to the element associated with the machine's next adapter. |
| Machine | LLR_MachineGetNextMCM | llr_element_t* (MCM) | A pointer to the element associated with the machine's next MCM. |
| Machine | LLR_MachineGetNextResource | llr_element_t* (Resource) | A pointer to the element associated with the machine's next resource. |
| Machine | LLR_MachineIsland | char* | A pointer to the island to which the machine belongs. Data is available from LLR_QUERY_RESOURCE_MANAGER. |
| Machine | LLR_MachineKbddIdle | int* | A pointer to an integer indicating the number of seconds since the kbdd daemon detected keyboard mouse activity. |
| Machine | LLR_MachineKillExpr | char** | A pointer to a string containing the machine's kill control expression. |
| Machine | LLR_MachineLargePageCount64 | int64_t* | A pointer to a 64–bit integer indicating the number of Large Pages defined on the machine. |
| Machine | LLR_MachineLargePageFree64 | int64_t* | A pointer to a 64–bit integer indicating the number of Large Pages free. |
| Machine | LLR_MachineLargePageSize64 | int64_t* | A pointer to a 64–bit integer indicating the size of the machine's Large Page. |
| Machine | LLR_MachineLoadAverage | double* | A pointer to a double containing the load average on the machine. |
| Machine | LLR_MachineMachineGroupName | char** | The name of the machine group it contains. |
| Machine | LLR_MachineMachineMode | char** | A pointer to a string containing the configured machine mode. |
| Machine | LLR_MachineMaxTasks | int* | A pointer to an integer indicating the maximum number of tasks this machine can run at one time. |
| Machine | LLR_MachineName | char** | A pointer to a string containing the machine name. |
| Machine | LLR_MachineOperatingSystem | char** | A pointer to a string containing the operating system on the machine. The string may result in "???" if a query is made before the associated records are updated with their actual values by the appropriate startd daemons. |
| Machine | LLR_MachinePagesFreed | int* | A pointer to an integer indicating the number of pages freed per second by the page replacement algorithm. |

Table 31. MACHINES specifications for llr_get_data subroutine (continued)

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Machine | LLR_MachinePagesFreed64 | int64_t* | A pointer to a 64-bit integer indicating the number of pages freed per second by the page replacement algorithm. |
| Machine | LLR_MachinePagesPagedIn | int* | A pointer to an integer indicating the number of pages paged in per second from paging space. |
| Machine | LLR_MachinePagesPagedIn64 | int64_t* | A pointer to a 64-bit integer indicating the number of pages paged in per second from paging space. |
| Machine | LLR_MachinePagesPagedOut | int* | A pointer to an integer indicating the number of pages paged out per second to paging space. |
| Machine | LLR_MachinePagesPagedOut64 | int64_t* | A pointer to a 64-bit integer indicating the number of pages paged out per second to paging space. |
| Machine | LLR_MachinePagesScanned | int* | A pointer to an integer indicating the number of pages scanned per second by the page replacement algorithm. |
| Machine | LLR_MachinePagesScanned64 | int64_t* | A pointer to a 64-bit integer indicating the number of pages scanned per second by the page replacement algorithm. |
| Machine | LLR_MachinePoolList | int** | A pointer to an array indicating the pool numbers to which this machine belongs. The size of the array can be determined by using LLR_MachinePoolListSize. |
| Machine | LLR_MachinePoolListSize | int* | A pointer to an integer indicating the number of pools configured for the machine. |
| Machine | LLR_MachineRealMemory | int* | A pointer to an integer indicating the physical memory in MBs on the machine. |
| Machine | LLR_MachineRealMemory64 | int64_t* | A pointer to a 64-bit integer indicating the physical memory in MBs on the machine. |
| Machine | LLR_MachineRSetSupport | int* | A pointer to an integer indicating the type of Affinity Support. Valid values are: <br> **`MCM_AFFINITY`** Indicates that the machine is configured to support MCM/CORE/CPU affinity. <br> **`USER_DEFINED_RSET`** Indicates that the machine is configured to support user-defined RSets. **Note:** User-defined RSets are not supported on Linux. <br> **`NO_AFFINITY`** Indicates that no specific affinity is configured on the machine. |

*Table 31. MACHINES specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| Machine | LLR_MachineScheddRunningJobs | int* | A pointer to an integer indicating a list of the running jobs assigned to the job manager on this machine. |
| Machine | LLR_MachineScheddRunsHere | int* | Whether this machine is configured to run Schedd. |
| Machine | LLR_MachineScheddState | int* | A pointer to an integer indicating the state of the job manager on this machine. |
| Machine | LLR_MachineScheddStepPending | int* | The number of jobs managed by this Schedd that are in Pending state. |
| Machine | LLR_MachineScheddStepRemovePending | int* | The number of jobs managed by this Schedd that are in Remove pending state. |
| Machine | LLR_MachineScheddStepStarting | int* | The number of jobs managed by this Schedd that are in Starting state. |
| Machine | LLR_MachineScheddTotalJobs | int* | A pointer to an integer indicating the total number of jobs assigned to the job manager on this machine. |
| Machine | LLR_MachineSMTState | int* | A pointer to an integer indicating the SMT state of a node. Valid values are: **SMT_ENABLED** and **SMT_DISABLED**. |
| Machine | LLR_MachineSpeed | double* | A pointer to a double containing the configured speed of the machine. |
| Machine | LLR_MachineStartdRunningJobs | int* | A pointer to an integer containing the number of running jobs known by the startd daemon. |
| Machine | LLR_MachineStartdRunsHere | int* | Whether this machine is configured to run startd. |
| Machine | LLR_MachineStartdState | char** | A pointer to a string containing the state of the startd daemon. |
| Machine | LLR_MachineStartExpr | char** | A pointer to a string containing the machine's start control expression. |
| Machine | LLR_MachineState | int * | A pointer to an integer containing the power state of the machine. |
| Machine | LLR_MachineStepList | char*** | A pointer to an array containing the steps running on the machine. The array ends with a NULL string. |
| Machine | LLR_MachineSuspendExpr | char** | A pointer to a string containing the machine's suspend control expression. |
| Machine | LLR_MachineTimeStamp | time_t* | A pointer to a **time_t** structure indicating the time the machine last reported to the negotiator. |
| Machine | LLR_MachineVacateExpr | char** | A pointer to a string containing the machine's vacate control expression. |
| Machine | LLR_MachineVirtualMemory | int* | A pointer to an integer indicating the free swap space in KBs on the machine. |
| Machine | LLR_MachineVirtualMemory64 | int64_t* | A pointer to a 64-bit integer indicating the free swap space in KBs on the machine. |

Table 31. MACHINES specifications for llr_get_data subroutine (continued)

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| MCM | LLR_MCMCPUList | int** | A pointer to an array indicating the list of CPUs on the MCM. |
| MCM | LLR_MCMCPUs | int* | A pointer to an integer containing the number of CPUs within the MCM. |
| MCM | LLR_MCMID | int* | A pointer to an integer containing the ID of the MCM. |
| Resource | LLR_ResourceAvailableValue | int* | A pointer to an integer indicating the value of available resources. |
| Resource | LLR_ResourceInitialValue | int* | A pointer to an integer indicating the initial resource value. |
| Resource | LLR_ResourceInitialValue64 | int64_t* | A pointer to a 64-bit integer indicating the initial resource value. |
| Resource | LLR_ResourceName | char** | A pointer to a string containing the resource name. |

See "Understanding the Machine group object model" on page 233 for more information on the MACHINE_GROUP object model.

Table 32 lists the MACHINE_GROUP specifications for the **llr_get_data** subroutine:

Table 32. MACHINE_GROUP specifications for llr_get_data subroutine

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| MachineGroup | LLR_MachineGroupConfiguredClassList | char*** | A NULL-terminated array of the class initiators configured in the machine group. Each string has the format "class_name(*N*)" where *N* is the number of initiators. |
| MachineGroup | LLR_MachineGroupExplicitlyDefinedMachines | char*** | A NULL-terminated list of hostnames of machines that have explicit definitions (substanzas) within the machine group. |
| MachineGroup | LLR_MachineGroupFeatureList | char*** | A NULL-terminated array of the features defined in the machine group. |
| MachineGroup | LLR_MachineGroupGetFirstResource | llr_element_t* (Resource) | A pointer to the first element associated with the machine groups configured list of resources. |
| MachineGroup | LLR_MachineGroupGetNextResource | llr_element_t* (Resource) | A pointer to the next element associated with the machine groups configured list of resources. |
| MachineGroup | LLR_MachineGroupGetResourceList | llr_data_t * | The resource iterator. |
| Machine | LLR_MachineGroupIsland | char** | A pointer to the island to which the machine group belongs. Data is available from LLR_QUERY_RESOURCE_MANAGER. |
| MachineGroup | LLR_MachineGroupMachineMode | char** | The mode configured for the machine group. |
| MachineGroup | LLR_MachineGroupMachineSpeed | double* | The speed configured on this machine group. |
| MachineGroup | LLR_MachineGroupMaxDstgStarters | int* | The number of **dstg_max_starters** on each machine in the machine group. |

*Table 32. MACHINE_GROUP specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| MachineGroup | LLR_MachineGroupMaxStarters | int* | The maximum number of tasks that can run on each machine in the machine group (the value of the **max_starters** keyword). |
| MachineGroup | LLR_MachineGroupName | char** | The name of the machine group. |
| MachineGroup | LLR_MachineGroupNumMachs | int* | The number of machines configured in the the group. |
| MachineGroup | LLR_MachineGroupPoolList | int** | An array indicating the pool numbers to which machines in this machine group belong. The size of the array can be determined by using LLR_MachineGroupPoolListSize. |
| MachineGroup | LLR_MachineGroupPoolListSize | int* | The number of pools configured for the machine group. |
| MachineGroup | LLR_MachineGroupPrestartedStarters | int* | The number of prestarted starters to start on each machine in the machine group. |
| MachineGroup | LLR_MachineGroupRange | char** | The expression which defines the range of machines included in the group. Use the **machine_expr_to_list()** subroutine to convert the range to an array of hostnames. Use the t**est_machine_in_expr()** subroutine to find out if a given hostname matches the range expression. |
| MachineGroup | LLR_MachineGroupRegion | char** | The region of this machine group. |
| MachineGroup | LLR_MachineGroupReservationPermitted | int* | A boolean which indicates whether machines in this group can be reserved. |
| MachineGroup | LLR_MachineGroupScheddAvail | int* | The count of available Schedd daemons in the machine group. |
| MachineGroup | LLR_MachineGroupScheddDown | int* | The count of down Schedd daemons in the machine group. |
| MachineGroup | LLR_MachineGroupScheddDrained | int* | The count of drained Schedd daemons in the machine group. |
| MachineGroup | LLR_MachineGroupScheddDraining | int* | The count of draining Schedd daemons in the machine group. |
| MachineGroup | LLR_MachineGroupScheddRunningJobs | int* | The total number of running job steps among all Schedd daemons in the machine group. |
| MachineGroup | LLR_MachineGroupScheddStepCompleted | int* | The number of steps queued among all Schedd daemons in the machine group with Completed state. |
| MachineGroup | LLR_MachineGroupScheddStepHeld | int* | The number of steps queued among all Schedd daemons in the machine group with Held state. |
| MachineGroup . | LLR_MachineGroupScheddStepIdle | int* | The number of steps queued among all Schedd daemons in the machine group with the Idle state. |
| MachineGroup | LLR_MachineGroupScheddStepPending | int* | The number of steps queued among all Schedd daemons in the machine group with Pending state. |
| MachineGroup | LLR_MachineGroupScheddStepRemoved | int* | The number of steps queued among all Schedd daemons in the machine group with Removed state. |

Table 32. MACHINE_GROUP specifications for llr_get_data subroutine (continued)

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| MachineGroup | LLR_MachineGroupScheddStepRemovePending | int* | The number of steps queued among all Schedd daemons in the machine group with RemovePending state. |
| MachineGroup | LLR_MachineGroupScheddStepStarting | int* | The number of steps queued among all Schedd daemons in the machine group with starting state. |
| MachineGroup | LLR_MachineGroupScheddStepUnexpanded | int* | The number of steps queued among all Schedd daemons in the machine group with Unexpanded state. |
| MachineGroup | LLR_MachineGroupScheddTotal | char** | The total number of machines in the machine group configured to be running a Schedd daemon. |
| MachineGroup | LLR_MachineGroupScheddTotalJobSteps | int* | The total number of job steps queued among all Schedd daemons in the machine group. |
| MachineGroup | LLR_MachineGroupStartdAvail | int* | The total number of Startds in the machine group which are either running jobs or available to run jobs (Idle, Run, or Busy states). |
| MachineGroup | LLR_MachineGroupStartdBusy | int* | The total number of Startds in the machine group that are busy. |
| MachineGroup | LLR_MachineGroupStartdDown | int* | The total number of Startds in the machine group that are down |
| MachineGroup | LLR_MachineGroupStartdIdle | int* | The total number of Startds in the machine group that are idle. |
| MachineGroup | LLR_MachineGroupStartdRunning | int* | The total number of Startds in the machine group that are running. |
| MachineGroup | LLR_MachineGroupStartdTotal | int* | The total number of machines in the machine group configured to be running a Startd daemon. |
| MachineGroup | LLR_MachineGroupStartdSuspend | int* | The total number of Startds in the machine group that are in suspend state. |
| MachineGroup | LLR_MachineGroupStartdTotalRunningTasks | int* | The total number of tasks of all running job steps on all machines in the machine group. |

See "Understanding the Wlmstat object model" on page 234 for more information on the WlmStat object model.

Table 33 lists the WLMSTAT specifications for the **llr_get_data** subroutine:

Table 33. WLMSTAT specifications for llr_get_data subroutine

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| WlmStat | LLR_WlmStatCpuSnapshotUsage | int* | A pointer to CPU usage obtained from the Workload Manager. |
| WlmStat | LLR_WlmStatCpuTotalUsage | int64_t* | A pointer to total CPU usage obtained from the Workload Manager. |
| WlmStat | LLR_WlmStatLargePageMemorySnapshotUsage | int* | A pointer to large page memory usage obtained from the Workload Manager. |

*Table 33. WLMSTAT specifications for llr_get_data subroutine (continued)*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| WlmStat | LLR_WlmStatMemoryHighWater | int64_t* | A pointer to real memory high water mark obtained from the Workload Manager. |
| WlmStat | LLR_WlmStatMemorySnapshotUsage | int* | A pointer to real memory usage obtained from the Workload Manager. |
| WlmStat | LLR_WlmStatVMemoryHighWater | int64_t* | A pointer to the virtual memory high-water mark obtained from the Workload Manager. |
| WlmStat | LLR_WlmStatVMemorySnapshotUsage | int64_t* | A pointer to virtual memory usage obtained from the Workload Manager. |

See “Understanding the LLEnergyTag object model” on page 234 for more information on the LLEnergyTag object model.

Table 34 lists the LLEnergyTag specifications for the **llr_get_data** subroutine:

*Table 34. LLEnergyTag specifications for the llr_get_data subroutine*

| Object | Specification | type of resulting data parameter | Description |
|---|---|---|---|
| LLEnergyTag | LLR_ETagGetPolicyList | llr_data_list_t* | The energy policy record iterator. Data is available from the LLR_QUERY_RESOURCE_MANAGER only. |
| LLEnergyTag | LLR_ETagGetFirstPolicy | llr_element_t* (llr_etag_policy) | A pointer to the first element associated with the energy policy tag. |
| LLEnergyTag | LLR_ETagGetNextPolicy | llr_element_t* (llr_etag_policy) | A pointer to the next element associated with the energy policy tag. |

## llr_query_free_data subroutine

Use the **llr_query_free_data** subroutine to free the query handle and all of the query data associated with the specified query handle. All of the **llr_element_t** objects in the object list that were obtained by the **llr_query_get_data** subroutine will be freed.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_query_free_data (llr_resmgr_handle_t *rm_handle,
                         llr_query_handle_t **q_handle, llr_element_t **errObj)
```

### Parameters

*rm_handle*
: Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*q_handle*
: Is the address of the pointer to the opaque structure returned from the **llr_init_resmgr** subroutine. The memory associated with the structure will be freed and the pointer set to NULL.

*errObj*
: Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

### Description

*rm_handle* and *q_handle* are the input fields for this subroutine.

The query handle pointed to by *q_handle* is freed and all of the query data associated with the specified query handle is freed. The *q_handle* pointer is set to NULL.

### Return Values

**LLR_API_OK**
: The **llr_element_t** objects were successfully freed.

**LLR_API_ERROR**
: The query element was not freed. A description of the error is provided in the *errObj* parameter.

### Related Information

Subroutines: **llr_get_data**, **llr_query_get_data**, **llr_query_set**

## llr_query_get_data subroutine

Use the **llr_query_get_data** subroutine to send a query request to the specified daemon along with the request data specified by the **llr_query_set** subroutine. The function will return a NULL-terminated array of objects that match the request.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_query_get_data (llr_resmgr_handle_t *rm_handle,
                        llr_query_handle_t *q_handle,
                        llr_query_source_t query_daemon, const char *hostname,
                        llr_element_t ***object_list,
                        llr_element_t **errObj)
```

### Parameters

*rm_handle*
: Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*q_handle*
: Is the pointer to an opaque structure that is returned from the **llr_query_set** subroutine.

*query_daemon*
: Is an enum specifying the LoadLeveler resource manager daemon or history file to query or whether you want to query job information stored in a history file. Possible values are:
- **LLR_QUERY_HISTORY_FILE**
- **LLR_QUERY_JOBMGR**
- **LLR_QUERY_MASTER**
- **LLR_QUERY_RESOURCE_MANAGER**
- **LLR_QUERY_STARTD**
- **LLR_QUERY_STARTER**

*hostname*
: When the *query_daemon* parameter is set to **LLR_QUERY_RESOURCE_MANAGER**, the *hostname* parameter is ignored. When the *query_daemon* parameter is set to **LLR_QUERY_HISTORY_FILE**, the *hostname* parameter must be a pointer to a character string containing the name of the history file to query. For all other values of the *query_daemon*, the *hostname* parameter must be a pointer to a character string containing the host name of the daemon to query or NULL if the daemon on the local machine is to be queried.

*object_list*
: Is the address of a pointer to a NULL-terminated array of pointers to the requested objects. The object list is returned as output from this function.

*errObj*
: Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

## Description

*rm_handle*, *q_handle*, *query_daemon*, and *hostname* are the input fields for this subroutine.

The output for this subroutine is returned in the *object_list* field. The field will contain a NULL-terminated array of **llr_element_t** objects. Each LoadLeveler daemon returns only the objects that it knows about. Use the **llr_get_data** subroutine to extract data from the objects.

## Return Values

**LLR_API_OK**
: The query request was successful.

**LLR_API_COMM_ERROR**
: The query request failed because of a communication error with a resource manager daemon. A description of the error is provided in the *errObj* parameter. The function can be retried when this error occurs.

**LLR_API_ERROR**
: The query request failed. A description of the error is provided in the *errObj* parameter.

## Related Information

Subroutines: **llr_get_data**, **llr_query_free_data**, **llr_query_set**

## llr_query_set subroutine

Use the **llr_query_set** subroutine to specify the type of query and the data that is to be requested with a subsequent call to the **llr_query_get_data** function. Queries can be filtered based on the *query_type* and *object_filter* selected.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_query_set (llr_resmgr_handle_t *rm_handle,
                   llr_query_handle_t **q_handle, llr_query_type_t query_type,
                   int filter_count, llr_query_filter_t *query_filter,
                   llr_element_t **errObj)
```

### Parameters

*rm_handle*
Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*q_handle*
Is the address of a pointer to an opaque structure that is set if the query initialization is successful. The **q_handle** structure is returned as output from this function and is passed as a parameter to other query API functions

*query_type*
Is an enum specifying the type of query. Possible values are:
- **LLR_CLUSTER_QUERY** - query cluster-level machine information
- **LLR_ENERGYTAG_QUERY** - query the energy tag information from the resource manager daemon
- **LLR_JOBQ_SUMMARY_QUERY** - query high-level summary information about jobs
- **LLR_JOBS_QUERY** - query job information
- **LLR_MACHINE_GROUP_QUERY**- query machine group information
- **LLR_MACHINES_QUERY** - query machine information
- **LLR_WLMSTAT_QUERY** - query Workload Manager

*filter_count*
Is an integer indicating the number of **llr_query_filter_t** structures in the *query_filter* array.

*query_filter*
Is an array of **llr_query_filter** structures. Table 35 provides information on *query_type* (in **llr_query_type_t**) and related *query_filter*:

*Table 35. query_filter summary*

| When query_type (in llr_query_type_t) is: | query_filter can be: | Filter description: |
|---|---|---|
| **LLR_CLUSTER_QUERY** | No query filters are supported for this query type. | No query filter can be used. |
| **LLR_ENERGYTAG_QUERY** | **LLR_QUERY_USER**<br>**LLR_QUERY_JOBID**<br>**LLR_QUERY_STEPID**<br><br>**LLR_QUERY_ETAGNAME** | Query by LoadLeveler user name.<br>Query by resource manager job ID.<br>Query by resource manager job step ID.<br>Query by energy policy tag name. |

Table 35. query_filter summary (continued)

| When query_type (in llr_query_type_t) is: | query_filter can be: | Filter description: |
|---|---|---|
| **LLR_JOBQ_SUMMARY_QUERY** | **LLR_QUERY_GROUP** | Query by LoadLeveler group. |
| | **LLR_QUERY_USER** | Query by LoadLeveler user. |
| **LLR_JOBS_QUERY** | **LLR_QUERY_ENDDATE** | Query by job end dates. History file query only. |
| | **LLR_QUERY_GROUP** | Query by LoadLeveler group. |
| | **LLR_QUERY_HOST** | Query by job manager. |
| | **LLR_QUERY_JOBID** | Query by resource manager job ID. |
| | **LLR_QUERY_MEDIUM** | Return all fields for a job except those relating to resources assigned to a running job.<br><br>The **LLR_QUERY_MEDIUM** *filter_type* can be used with other filters normally used with a **LLR_JOBS_QUERY** query type, but the result is that fewer data fields are sent. |
| | **LLR_QUERY_STARTDATE** | Query by job start dates. History file query only. |
| | **LLR_QUERY_STEPID** | Query by resource manager job step ID. |
| | **LLR_QUERY_USER** | Query by user ID. |
| **LLR_MACHINE_GROUP_QUERY** | **LLR_QUERY_HOST** | Query by machine group names and machine names. |
| **LLR_MACHINES_QUERY** | **LLR_QUERY_HOST** | Query by machine names. |
| **LLR_WLMSTAT_QUERY** | **LLR_QUERY_STEPID** | Query by resource manager job step ID. |

*filter_data*
Is a NULL-terminated array of strings to be used to filter the data to be queried. The type of filter data is related to the specified *filter_type* as shown in Table 36:

Table 36. filter_data value related to the query_filter value

| If you specify: | Note: |
|---|---|
| **LLR_QUERY_ETAGNAME** | The *filter_data* contains a list of LoadLeveler energy policy tag names. |
| **LLR_QUERY_GROUP** | The *filter_data* contains a list of LoadLeveler group names. |
| **LLR_QUERY_HOST** | The *filter_data* contains a list of LoadLeveler machine names. |
| **LLR_QUERY_JOBID** | The *filter_data* contains a list of job IDs. |
| **LLR_QUERY_MEDIUM** | The filter_data is not used. |
| **LLR_QUERY_STARTDATE** or **LLR_QUERY_ENDDATE** | The *filter_data* contains a list of two start dates or two end dates having the format *MM/DD/YYYY*. |
| **LLR_QUERY_STEPID** | The *filter_data* contains a list of step IDs. |
| **LLR_QUERY_USER** | The *filter_data* contains a list of user IDs. |

*errObj*
Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

## Description

*rm_handle*, *q_handle*, *query_type*, *filter_count*, and *query_filter* are the input fields for this subroutine.

You can request certain combinations of object filters by calling **llr_query_set** with an array of **llr_query_filter_t** structures. When you do this, the query filter types you specify are or-ed together. The following are valid combinations of object filters:

- **LLR_QUERY_JOBID** and **LLR_QUERY_STEPID**: the result is the union of both queries and any other query filter types (such as, **LLR_QUERY_HOST**) will be ignored
- **LLR_QUERY_STARTDATE** and **LLR_QUERY_ENDDATE**: the result is the intersection of both queries
- **LLR_QUERY_HOST**, **LLR_QUERY_USER**, and **LLR_QUERY_GROUP**: the result is the intersections of all of the queries

To query jobs owned by certain users and on specific machines, issue **llr_query_set** with an array of **llr_query_filter_t** structures where the first element contains **LLR_QUERY_USER** with the appropriate user IDs and the second element contains **LLR_QUERY_HOST** with the appropriate host names.

For example, suppose you issue **llr_query_set** with an array of two **llr_query_filter_t** structures where the first element contains a user ID list of anton and meg, and the second element contains a host list of k10n10 and k10n11. The objects returned are all of the jobs on k10n10 and k10n11 that belong to anton or meg.

Note that if you use two **llr_query_filter_t** structures with the same flag, the second element will replace the first element. For history file queries, **llr_query_filter_type** is restricted to **LLR_QUERY_STARTDATE** and **LLR_QUERY_ENDDATE**.

The **LLR_QUERY_MEDIUM** *filter_type* can be used with other filters normally used with a **LLR_JOBS_QUERY** query type, but the result is that fewer data fields are sent.

## Return Values

**LLR_API_OK**
Indicates that the request was successfully set.

**LLR_API_ERROR**
Indicates that the query request failed. A description of the error is provided in the *errObj* parameter.

## Related Information

Subroutines: **llr_get_data**, **llr_query_free_data**, **llr_query_get_data**

## Event handling API

Use the event handling API to allow a scheduler to register for job and machine-related events from the resource manager. The resource manager will send events such as machine down or a new job state to all registered processes. Functions will be provided to allow a registered process to wait for events and read events. Related data will be provided for some events, for example, the resource usage data will be sent with a job step complete event.

The event handling subroutines provide flexibility in how event handling is coded in a scheduler. Some options for handling events include:

- The scheduler can spawn multiple threads for each **llr_read_events()** call within a loop. Only one thread can wait on a listen socket, but as soon as **accept()** is returned, another thread will starting listening on the socket.
- If the scheduler has a single thread that must wait on several file descriptors, the file descriptor for that event listen socket will be made available for use in a **select()** function. When **select()** detects that the event listen socket is ready, **llr_read_events()** can be invoked.
- A scheduler can use post and wait logic to have one thread call **llr_read_events()** to wait on the event listen socket. When events are read, another thread can be posted to call **llr_get_event()** to process the events.

This API consists of the following subroutines:

- **“llr_free_event subroutine” on page 280**
- **“llr_free_resmgr subroutine” on page 281**
- **“llr_get_event subroutine” on page 282**
- **“llr_get_event_fd subroutine” on page 287**
- **“llr_init_resmgr subroutine” on page 288**
- **“llr_read_events subroutine” on page 289**
- **“llr_register_for_events subroutine” on page 290**
- **“llr_unregister_for_events subroutine” on page 292**
- **“llr_version_resmgr subroutine” on page 294**

## llr_free_event subroutine

Use the **llr_free_event** subroutine to free the memory associated with an event structure passed as an argument.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_free_event (llr_resmgr_handle_t *rm_handle, llr_event_t **event_data,
                    llr_element_t **errObj)
```

### Parameters

*rm_handle*
: Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*event_data*
: Is the address of a pointer to a structure that contains data for an event. The memory associated with the event will be freed and the pointer set to NULL.

*errObj*
: Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

### Description

*rm_handle* and *event_data* are the input fields for this subroutine.

The memory associated with the **llr_event_t** structure pointed to by **event_data* is freed and all of the data associated with the specified event is freed. The **event_data* pointer is set to NULL.

### Return Values

**LLR_API_OK**
: The event data was freed.

**LLR_API_ERROR**
: The event data could not be freed. A description of the error is provided in the *errObj* parameter.

### Related Information

Subroutines: **llr_get_event**, **llr_get_event_fd**, **llr_read_events**, **llr_register_for_events**, **llr_unregister_for_events**

## llr_free_resmgr subroutine

Use the **llr_free_resmgr** subroutine to free the resource manager API common data structures.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_free_resmgr (llr_resmgr_handle_t **rm_handle, llr_element_t **errObj)
```

### Parameters

*rm_handle*
: Is the address of a pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine. The memory associated with the structure will be freed and the pointer will be set to NULL.

*errObj*
: Is the address of a pointer to a LoadLeveler error object. The pointer to **llr_element_t** should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

### Description

*rm_handle* is the input field for this subroutine.

The memory associated with the **llr_resmgr_handle_t** structure pointed to by **rm_handle* and the **rm_handle* pointer is set to NULL.

### Return Values

**LLR_API_OK**
: The resource manager data was successfully freed.

**LLR_API_ERROR**
: The resource manager data was not freed. A description of the error is provided in the *errObj* parameter.

### Related Information

Subroutines: **llr_init_resmgr**, **llr_version_resmgr**

## llr_get_event subroutine

Use the **llr_get_event** subroutine to get the first event off of the event queue and to return a pointer to the event structure.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_get_event (llr_resmgr_handle_t *rm_handle, llr_event_t **event_data,
                   llr_element_t **errObj)
```

### Parameters

*rm_handle*
: Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*event_data*
: Is the address of a pointer that will be set to a structure containing the data for an event. The event structure is returned as output from this function and contains the following fields:

  *event_type*
  : Is an enum that indicates the type of event.

  *event_data*
  : Is a void pointer to a structure that contains data related to the event. The pointer can be cast to a pointer to a specific structure depending on the event type.

  If no events are on the event queue, the pointer will be set to NULL.

  Table 37 provides information resource manager event types, possible states, and associated event data.

*Table 37. Resource manager events and event data*

| *event_type* | **Event description** | **Possible states and associated** *event_data* |
|---|---|---|
| **LLR_ALL_JOBS_EVENT** | When a process invokes the **llr_register_for_events** function, the resource manager will send the **all_jobs** event. | The data associated with this event is a NULL-terminated array of **llr_element_t** structures that contain job data. |
| **LLR_ALL_MACHINES_EVENT** | When a process invokes the **llr_register_for_events** function, the resource manager will send the **all_machines** event. | The data associated with this event is a NULL-terminated array of **llr_element_t** pointers. The **llr_element_t** structures contain machine data. |
| **LLR_JOB_ADD_EVENT** | The **job_add** event is sent when a job is added to the resource manager's job database. | The data associated with this event is a pointer to an **llr_element_t** that contains the job data. |
| **LLR_JOB_DELETE_EVENT** | The **job_delete** event is sent when the resource manager removes a job from the job database. | The data associated with this event is a character string containing the resource manager job ID for the deleted job. |

*Table 37. Resource manager events and event data (continued)*

| *event_type* | **Event description** | **Possible states and associated** *event_data* |
|---|---|---|
| **LLR_JOB_ID_CHANGE_EVENT** | The **job_id_change** event is sent when the job's ID is changed. | The data associated with this event is a pointer to a structure with the following fields: <br> **new_job_id** <br> Is a character string containing the new job ID. <br> **new_job_mgr** <br> Is a character string containing the new job manager ID. <br> **newSteplist** <br> Is a pointer to a NULL-terminated array of character strings containing the new job step IDs. <br> **old_job_id** <br> Is a character string containing the old job ID. <br> **oldSteplist** <br> Is a pointer to a NULL-terminated array of character strings containing the old job step IDs. |
| **LLR_JOB_MGR_DOWN_EVENT** | The job **manager_down** event is sent to all registered processes when the resource manager daemon detects that a job manager daemon is down. Job start and control operations cannot be performed for jobs whose job manager is down. | The data associated with this event is a character string containing the host name of the machine from which the job manager was running. |
| **LLR_JOB_MGR_UP_EVENT** | When a job manager daemon starts up, the resource manager sends the job **manager_up** event to all registered processes. | The data associated with this event is a character string containing the host name of the machine from which the job manager is running. |

Table 37. Resource manager events and event data (continued)

| *event_type* | **Event description** | **Possible states and associated** *event_data* |
|---|---|---|
| **LLR_JOB_STEP_STATE_ CHANGE_ EVENT** | The **job_state_change** event is sent with the resource manager changes the state of a job. | The data associated with this event is a pointer to a structure with the following fields:<br>**completion_code**<br>Is an integer indicating the exit code of the job.<br>**completion_date**<br>Is a time_t indicating the system date step was completed<br>**job_step_id**<br>Is a character string containing the resource manager job step ID.<br>**job_step_state**<br>Is an enum indicating the type of event.<br>**job_step_state_flags**<br>Is an unsigned integer indicating the job state flag. Possible values are 0, 1 (**LLR_JOB_STEP_REJECTED**).<br>**job_step_usage**<br>Is a pointer to the **rusage64** structure.<br>**rejecting_machine**<br>Is an string indicating the machine that rejects the job.<br>**job_step_id** and **job_step_state** are intended for all states. The following are the possible states and additional associated data for those fields:<br>• IDLE - data pointer is NULL<br>• STARTING - data pointer is **llr_element_t**, which is a job object containing the resources assigned to the job step<br>• RUNNING - data pointer is NULL<br>• COMPLETED - data pointer points to usage data for the job step<br>• REJECTED - data pointer is a character string containing the rejecting machine<br>• REMOVED - data pointer points to usage data for the job step<br>• VACATED - data pointer points to usage data for the job step<br>• PREEMPTED - data pointer is NULL<br>• CHECKPOINTING - data pointer is NULL |
| **LLR_JOB_STEP_UPDATE_ EVENT** | The **job_step_update** event is sent when the job manager is starting the job after all resources assigned to the job have been received and recorded. | The data associated with this event is a NULL-terminated array of **llr_element_t** structures that contain step data. |

Table 37. Resource manager events and event data (continued)

| *event_type* | **Event description** | **Possible states and associated** *event_data* |
|---|---|---|
| **LLR_MACHINE_CONFIG_ EVENT** | The **machine_config** event is sent when the resource manager is started or configured on a machine in the cluster. | The data associated with this event is a pointer to **llr_element_t**, which is a machine object that contains all machine attributes. |
| **LLR_MACHINE_DOWN_EVENT** | The **machine_down** event is sent when the resource manager detects that a machine in the cluster is down. | The data associated with this event is a character string containing the host name of the machine that is down. |
| **LLR_MACHINE_UPDATE_ ADAPTER_EVENT** | The **machine_update_adapter** event is sent to all registered processes when the resource manager finds that the adapter status of a machine is changed. | The data associated with this event is a pointer to a **llr_element_t** structure, which is a machine object that contains the **LL_MachineAdapterList** machine attributes. |
| **LLR_MACHINE_UPDATE_ EVENT** | The **machine_update** event is sent when the resource manager changes any dynamic attribute of a machine in the cluster. | The data associated with this event is a pointer to **llr_element_t**, which is a machine object that contains the changed machine attributes. |
| **LLR_RESOURCE_MGR_DOWN_ EVENT** | The **resource_mgr_down** event is sent to all registered processes when the resource manager shuts down. | No data is associated with this event. |
| **LLR_RESOURCE_MGR_UP_ EVENT** | When the resource manager starts up, the **resource_mgr_up** event is sent to all processes that previously registered with the resource manager. The **resource_mgr_up** event is also sent to any new process registering with the resource manager. | No data is associated with this event. |

*errObj*
: Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

## Description

*rm_handle* is the input field for this subroutine.

The output for this subroutine is returned in the *event_data* field. It is the caller's responsibility to call the **llr_free_event** subroutine to free the storage associated with the event.

## Return Values

**LLR_API_OK**
: The request was successful.

**LLR_API_ERROR**
The request failed. A description of the error is provided in the *errObj* parameter.

## Related Information

Subroutines: **llr_free_event**, **llr_get_event_fd**, **llr_read_events**, **llr_register_for_events**, **llr_unregister_for_events**

## llr_get_event_fd subroutine

Use the **llr_get_event_fd** subroutine to return the file descriptor associated with the listen socket for the specified event handle. The file descriptor can be used in a select call. When the socket is ready to read the **llr_read_events** function, it can be invoked to accept the connection and read the events.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_get_event_fd (llr_resmgr_handle_t *rm_handle)
```

### Parameters

*rm_handle*
: Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

### Description

*rm_handle* is the input field for this subroutine.

The **llr_register_for_events** subroutine must be called before using this subroutine. A -1 is returned if the process has not registered for events.

### Return Values

| | |
|---|---|
| **>= 0** | The file descriptor for the listen socket. |
| **< 0** | The resource manager handle parameter is not valid. |

### Related Information

Subroutines: **llr_free_event**, **llr_get_event**, **llr_read_events**, **llr_register_for_events**, **llr_unregister_for_events**

## llr_init_resmgr subroutine

Use the **llr_init_resmgr** subroutine to initialize the common data structures for other resource manager API functions. The handle returned by this subroutine will be used as an argument in other resource manager API functions. The **llr_free_resmgr** subroutine is used to free the memory associated with the handle returned by this function.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_init_resmgr (int version, llr_resmgr_handle_t **rm_handle,
                     llr_element_t **errObj)
```

### Parameters

*version*
: Is an input parameter indicating the resource manager API version. This parameter is set with **LLR_API_VERSION** from the resource manager API header file.

*rm_handle*
: Is the address of a pointer to an opaque structure that is set if initialization is successful. The *rm_handle* structure is returned as a parameter to other resource manager API functions.

*errObj*
: Is the address of a pointer to a LoadLeveler error object. The pointer to **llr_element_t** should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

### Description

*version* is the input field for this subroutine.

The output for this subroutine is returned in the *rm_handle* field. The **llr_resmgr_handle_t** structure contains common data structures for the resource manager API functions. The returned *rm_handle* pointer must be passed to other resource manager API functions. The **llr_free_resmgr** subroutine is used to free the memory associated with the *rm_handle* returned by this function.

### Return Values

**LLR_API_OK**
: The initialization was successful.

**LLR_API_ERROR**
: The initialization failed. A description of the error is provided in the *errObj* parameter.

### Related Information

Subroutines: **llr_free_resmgr**, **llr_version_resmgr**

## llr_read_events subroutine

Use the **llr_read_events** subroutine to invoke the **accept( )** system call, which will block until the event listen socked has a connection.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

**int llr_read_events (llr_resmgr_handle_t** **rm_handle*, **llr_element_t** ***errObj***)**

### Parameters

*rm_handle*
: Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*errObj*
: Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

### Description

The **llr_read_events** subroutine invokes the **accept( )** system call, which will block until the event listen socked has a connection.

### Return Values

**LLR_API_OK**
: Events were successfully read.

**LLR_API_ERROR**
: Failed to read events. A description of the error is provided in the *errObj* parameter.

### Related Information

Subroutines: **llr_free_event**, **llr_get_event**, **llr_get_event_fd**, **llr_register_for_events**, **llr_unregister_for_events**

## llr_register_for_events subroutine

Use the **llr_register_for_events** subroutine to register the current process to receive all events generated by the resource manager.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_register_for_events (llr_resmgr_handle_t *rm_handle,
                             llr_event_registration_t *registration_data,
                             llr_element_t **errObj)
```

### Parameters

*rm_handle*
Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*registration_data*
Is a pointer to a structure that contains the following fields:

*port* Specifies the port number to use for receiving events. If *port* is set to 0, the system will assign a port number.

*scheduler_id*
Specifies a string to identify the process that is registering to receive events. When supporting multiple schedulers, the *scheduler_id* will be used by the resource manager to determine which scheduler will receive certain events. The string can contain any characters.

*errObj*
Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

### Description

The **llr_register_for_events** subroutine registers the current process to receive all events generated by the resource manager. A listen socket will be established for the process. The registration data is sent to the resource manager daemon. Registration at the resource manager daemon is persistent until the unregister function is called. If the registered scheduler goes down or a network failure occurs, a resource manager daemon will continue to send all events until a failure threshold is reached. At that point, all events will be discarded and the resource manager daemon will periodically attempt to send a **resource_mgr_up** event to the registered scheduler. Once the event is acknowledged, normal event transmission will be continued.

Multiple registrations will be allowed provided the registration is from a LoadLeveler administrator on an authorized machine. A scheduler ID string will be used to identify a registered scheduler. If a registration request is received for a scheduler ID that is already registered, the host name and port number for the new request will replace the existing registration. All machine-related events will be sent to all registered schedulers. All jobs will be created with a scheduler ID string. Job-related events will be sent only to the scheduler specified in the scheduler ID.

## Return Values

**LLR_API_OK**
: The registration was successful.

**LLR_API_COMM_ERROR**
: The registration failed because of a communication error with a resource manager daemon. A description of the error is provided in the *errObj* parameter. The function can be retried when this error occurs.

**LLR_API_ERROR**
: The registration failed. A description of the error is provided in the *errObj* parameter.

## Related Information

Subroutines: **llr_free_event**, **llr_get_event**, **llr_get_event_fd**, **llr_read_events**, **llr_unregister_for_events**

## llr_unregister_for_events subroutine

Use the **llr_unregister_for_events** subroutine to inform the resource manager that no more events are to be sent to the specified scheduler.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_unregister_for_events (llr_resmgr_handle_t *rm_handle,
                               llr_event_registration_t *registration_data,
                               llr_element_t **errObj)
```

### Parameters

*rm_handle*
Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*registration_data*
Is a pointer to a structure that contains the following fields:

*port* This field is ignored when unregistering for events.

*scheduler_id*
Specifies a string containing the name of the scheduler ID to be unregistered. If this string does not match a registered scheduler ID, an error is returned.

*errObj*
Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

### Description

*rm_handle* and *registration_data* are the input fields for this subroutine.

The listen socket that was established to listen for events will be closed by the subroutine and the resource manager will be informed that no more events are to be sent to the specified scheduler. Any remaining events in the event queue can still be accessed using the **llr_read_event** subroutine.

### Return Values

**LLR_API_OK**
The unregister was successful.

**LLR_API_COMM_ERROR**
The unregister failed because of a communication error with a resource manager daemon. A description of the error is provided in the *errObj* parameter. The function can be retried when this error occurs.

**LLR_API_ERROR**
The unregister failed. A description of the error is provided in the *errObj* parameter.

## Related Information

Subroutines: **llr_free_event**, **llr_get_event**, **llr_get_event_fd**, **llr_read_events**, **llr_register_for_events**

## llr_version_resmgr subroutine

Use the **llr_version_resmgr** subroutine to get version information for the resource manager API. The version is returned as an integer.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_version_resmgr (void)
```

### Parameters

None.

### Description

The version returned by the **llr_version_resmgr** subroutine is an integer representing the version, release, and modification level of the resource manager library.

### Return Values

An integer indicating the version of the resource manager API is returned.

### Related Information

Subroutines: **llr_free_resmgr**, **llr_init_resmgr**

## Workload management API

Use the workload management API to add and delete jobs from the resource manager database, to start a job step using specific resources, to preempt, checkpoint, vacate, cancel a job or job step, to convert an error object to an error message string and free the memory, to remove the energy policy tags from the database, to allow a scheduling application to specify the destination nodes and initiate the migration operation, and to update job attributes in the job database.

This API consists of the following subroutines:
- **"llr_add_job subroutine" on page 296**
- **"llr_change_node_powerstate subroutine" on page 298**
- **"llr_cluster_auth subroutine" on page 300**
- **"llr_control subroutine" on page 302**
- **"llr_control_job subroutine" on page 304**
- **"llr_delete_job subroutine" on page 306**
- **"llr_error subroutine" on page 307**
- **"llr_free_job subroutine" on page 308**
- **"llr_move_spool subroutine" on page 309**
- **"llr_remove_energy_tags subroutine" on page 311**
- **"llr_start_job_migration subroutine" on page 313**
- **"llr_start_job_step subroutine" on page 315**

The **llr_control_job** subroutine can be used to perform most of the LoadLeveler control operations and is designed for general use.

**Note:** The Workload Manager (WLM) and the LoadLeveler workload management API are two distinct and unrelated components.

## llr_add_job subroutine

Use the **llr_add_job** subroutine to add a job to the resource manager job database and notify all registered processes that a new job has been added.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_add_job (llr_resmgr_handle_t *rm_handle, llr_job_info_t *job_data,
                 llr_element_t **jobObj, llr_element_t **errObj)
```

### Parameters

*rm_handle*
Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*job_data*
Is a pointer to a structure containing job data. The **job_info** structure contains the following fields:

*format* An enum indicating the job description language. The LoadLeveler job description is the only supported language at this time.

*job_description*
A pointer to a string containing the job description.

*mode* An enum indicating the job mode. The mode can be **LLR_BATCH_MODE** or **LLR_INTERACTIVE_MODE**.

*scheduler_ID*
A pointer to a character string containing the ID of the scheduler that will schedule the job. All resource manager events related to the job will be sent to the scheduler that is registered with this ID.

*jobObj*
Is the address of a pointer that will point to a job object containing the job data if the job is successfully created. The job object is returned as output from this function. The attributes for the resource manager job ID and resource manager step IDs will be set in the job by this function.

*errObj*
Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error object will contain all parsing errors. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

### Description

The **llr_add_job** subroutine adds a job to the resource manager job database and notifies all registered processes that a new job has been added. The job description is passed as a string that specifies the job using the LoadLeveler job description language. This function parses the string and creates a job object. The job object is returned as **llr_element_t**. If the job description specifies an executable file, that file will be copied by the resource manager to a **spool** directory. If the job description does not specify an executable file, the input string is copied into a spool file and the file is used as the executable.

The credentials of the process that calls this function will be the same credentials from which the job will run. This function will pick up the user ID, the group ID, AFS and DCE credentials, and store them in the job object. These credentials will be used by the resource manager to run the job.

## Return Values

**LLR_API_OK**
The job was created.

**LLR_API_COMM_ERROR**
The job could not be added because of a communication error with a resource manager daemon. A description of the error is provided in the *errObj* parameter. The function can be retried when this error occurs.

**LLR_API_ERROR**
The job was not added. A description of the error is provided in the *errObj* parameter.

## Related Information

Subroutines: **llr_control_job**, **llr_delete_job**, **llr_free_job**, **llr_start_job_step**

## llr_change_node_powerstate subroutine

Use the **llr_change_node_powerstate** subroutine to change the power state of a compute node.

### Library

LoadLeveler API library **libllrapi.so** (Linux)

### Syntax

```
llr_change_node_powerstate(llr_resmgr_handle_t *rm_handle,
                           llr_power_state_t pstate, char**
                           hostlist, char*** reasons,
                           llr_element_t **err_obj)
```

### Parameters

*rm_handle*
: Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*pstate*
: The power state to be changed to.

*hostlist*
: Is a NULL-terminated array of host names to be processed.

*reasons*
: Is a pointer to a NULL-terminated array of reasons created when there is a failure to change the power state of one or more compute nodes. The *reasons* array stores the reason for the failure for the corresponding compute node in the *hostlist* array.

*errObj*
: Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

### Description

The **llr_change_node_powerstate** is the API for the **llrchgmstat** command.

This function is used for the energy management function to change the power mode of the compute nodes.

LoadLeveler administrators or a **root** user can change the idle nodes to standby state to save energy, or can resume the nodes from standby to working state. The node cannot go into standby state when there are running jobs.

### Return Values

**LLR_API_OK**
: Indicates the power state was successfully changed.

**LLR_API_COMM_ERROR**
: The request failed because of a communication error with a resource

manager daemon. A description of the error is provided in the **errObj** parameter. The function can be retried when this error occurs.

**LLR_API_ERROR**
The request failed. A description of the error is provided in the errObj parameter.

## llr_cluster_auth subroutine

Use the **llr_cluster_auth** subroutine to create a public key, a private key, and a security certificate.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_cluster_auth (int version, llr_cluster_auth_param_t **param,
                      llr_element_t **errObj);
```

### Parameters

*version*
: Is an input parameter that indicates the LoadLeveler API version (should have the same value as **LLR_API_VERSION** in **llrapi.h**).

*param*
: Provides the address of a pointer to an array of pointers to **llr_cluster_auth_param_t** structures. The last element of the array must be NULL.

*errObj*
: Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

### Description

The **llr_cluster_auth( )** subroutine is the API for the **llrclusterauth** command. Refer to the **llrclusterauth** command for information about other available command options.

This function must be run from a process with **root** authority.

The **llr_cluster_auth()** subroutine creates a public key, a private key, a security certificate, and a directory for authorized keys. The keys and certificate are created in the **/var/LoadL/ssl** directory for AIX and in the **/var/opt/LoadL/ssl** directory for Linux.
- The private key is stored in **id_rsa**
- The public key is stored in **id_rsa.pub**
- The security certificate is stored in **id_rsa.cert**
- The authorized keys are stored in **authorized_keys**

In order for a connection to be accepted, the public key for the node requesting the connection must be stored in the authorized keys file on the node being connected to. Only a process with **root** authority can run this subroutine.

### Return Values

**LLR_API_OK**
: The request transaction was successfully sent.

**LLR_API_ERROR**
: The request failed. A description of the error is provided in the *errObj* parameter.

## Related Information

Commands: **llrclusterauth**

Subroutines: **llr_control**, **llr_move_spool**

## llr_control subroutine

Use the **llr_control** subroutine to allow an application program to perform most of the functions that are available through the standalone **llrctl** command.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_control (llr_resmgr_handle_t *rm_handle, llr_control_op_t control_op,
                 char **host_list, llr_element_t ** errObj)
```

### Parameters

*rm_handle*
Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

**LLR_control_op** *op*
Is the control operation to be performed. Possible values are:
- LLR_CONTROL_DRAIN_JOBMGR
- LLR_CONTROL_DUMP_LOCKS
- LLR_CONTROL_DUMP_LOGS
- LLR_CONTROL_RECONFIG
- LLR_CONTROL_RECYCLE
- LLR_CONTROL_RESUME_JOBMGR
- LLR_CONTROL_START
- LLR_CONTROL_STOP

*host_list*
Is a NULL-terminated array of host names.

*errObj*
Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

### Description

The **llr_control** subroutine performs operations that are essentially equivalent to those performed by the **llrctl** command.

In LoadLeveler for Linux only, **llr_control** returns an error condition when **SEC_ENABLEMENT** is **CTSEC**.

**llrctl type of operations:** These are the **llr_control** operations which mirror operations performed by the **llrctl** command. This summary includes a brief description of each of the allowed **llrctl** types of operations. For more information about the **llrctl** command, see "llrctl - Control LoadLeveler daemons" on page 166.

**LLR_CONTROL_DRAIN_JOBMGR:**
When this operation is selected, the following happens: (1) No LoadLeveler jobs can start running on the specified machines, and (2) No LoadLeveler jobs can be submitted to the specified machines.

**LLR_CONTROL_DUMP_LOCKS:**
Dumps the locking records for all enabled daemons (that have the **D_LOCK_TRACE** flag set) into the **LoadL** daemon logs.

**LLR_CONTROL_DUMP_LOGS:**
Allows the request to dump the logging buffer.

**LLR_CONTROL_RECONFIG:**
Forces all of the LoadLeveler resource manager daemons on the specified machines to reread the configuration files.

**LLR_CONTROL_RECYCLE:**
Stops, and then restarts, all of the LoadLeveler resource manager daemons on the specified machines.

**LLR_CONTROL_RESUME_JOBMGR:**
Resumes job submission to the specified machines.

**LLR_CONTROL_START:**
Starts the LoadLeveler resource manager daemons on the specified machines. The calling program must have rsh privileges to start the LoadLeveler resource manager daemons on remote machines.

**Note:** LoadLeveler will fail to start if any value has been set for the MALLOCTYPE environment variable.

**LLR_CONTROL_STOP:**
Stops the LoadLeveler resource manager daemons on the specified machines.

For these **llrctl** type of operations, the *user_list*, *job_list*, and *priority* arguments are not used and should be set to **NULL** or zero. Unlike the standalone **llrctl** command, where the scope of the operation is either global or one host, **llr_control** operations allow the user to specify a list of hosts (through the *host_list* argument). To perform these operations, the calling program must have LoadLeveler administrator authority. The only exception to this rule is the **LLR_CONTROL_START** operation.

## Return Values

**LLR_API_OK**
The request transaction was successfully sent.

**LLR_API_COMM_ERROR**
The request failed because of a communication error with a resource manager daemon. A description of the error is provided in the *errObj* parameter. The function can be retried with this error occurs.

**LLR_API_ERROR**
The request failed. A description of the error is provided in the *errObj* parameter.

## Related Information

Commands: **llrctl**

Subroutines: **llr_cluster_auth**, **llr_move_spool**

## llr_control_job subroutine

Use the **llr_control_job** subroutine to send a job control order to the resource manager for the specified job or job step. After the control order runs, the resource manager will send an event to all registered processes.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_control_job (llr_resmgr_handle_t *rm_handle, const char *job_id,
                     llr_job_control_op_t op, void *op_data,
                     llr_element_t **errObj)
```

### Parameters

*rm_handle*
: Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*job_id*
: Is the pointer to a character string that contains the resource manager ID for the job or the job step to be controlled.

*op*
: Is an enum indicating the control operation:

**LLR_CANCEL_OP**
: Terminates the job step with the specified signal and puts it in the REMOVED state.

**LLR_VACATE_OP**
: Terminates the job step with the specified signal and puts it in the VACATED state.

**LLR_SIGNAL_OP**
: Sends the specified signal to the job step, but does not change the state.

**LLR_PREEMPT_OP**
: Stops all job step processes.

**LLR_PREEMPT_WITHOUT_ADAPTERS_OP**
: Stops all job processes, but does not free the adapters used by the job step.

**LLR_RESUME_OP**
: Resumes all job step processes.

**LLR_IDLE_OP**
: Changes the job state to idle. A job step can only be started when it is in the idle state.

**LLR_CKPT_AND_CONTINUE_OP**
: Checkpoints the job step and then resumes.

**LLR_CKPT_AND_TERMINATE_OP**
: Checkpoints and then terminates the job step.

*op_data*

If a signal is specified for **LLR_CANCEL_OP** or **LLR_VACATE_OP**, there is 120 seconds waiting time between sending the signal and taking action to cancel or vacate.

*errObj*

Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

## Description

This function initiates a job control operation, such as cancel or preempt, on a job or job step. This function must be called by the owner of the job or a LoadLeveler administrator.

## Return Values

**LLR_API_OK**

The control order was sent to the resource manager.

**LLR_API_COMM_ERROR**

The control order was not sent to the resource manager because of a communication error with a resource manager daemon. A description of the error is provided in the *errObj* parameter. The function can be retried when this error occurs.

**LLR_API_ERROR**

The control order was not sent to the resource manager. A description of the error is provided in the *errObj* parameter.

## Related Information

Subroutines: **llr_add_job**, **llr_delete_job**, **llr_free_job**,**llr_start_job_step**

## llr_delete_job subroutine

Use the **llr_delete_job** subroutine to delete a job from the resource manager job database and notify all registered processes. Only jobs in a nonrunning state will be deleted.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_delete_job (llr_resmgr_handle_t *rm_handle, const char *job_id,
                    llr_element_t **errObj)
```

### Parameters

*rm_handle*
: Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*job_id*
: Is the pointer to a character string containing the resource manager job ID for the job to be deleted.

*errObj*
: Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

### Description

This function must be called by a LoadLeveler administrator.

### Return Values

**LLR_API_OK**
: The job was deleted from the resource manager job database.

**LLR_API_COMM_ERROR**
: The job was not deleted from the resource manager job database because of a communication error with a resource manager daemon. A description of the error is provided in the *errObj* parameter. The function can be retried when this error occurs.

**LLR_API_ERROR**
: The job was not deleted from the resource manager job database. A description of the error is provided in the *errObj* parameter.

### Related Information

Subroutines: **llr_add_job**, **llr_control_job**, **llr_free_job**, **llr_start_job_step**

## llr_error subroutine

Use the **llr_error** subroutine to convert an error object to an error message string and free the memory associated with the error object. As an option, the error message can be returned as string, printed to stdout or stderr, or the memory can be free without printing or returning a string.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

**`char *llr_error (llr_element_t **`** *`errObj`***`, llr_error_opt_t`*** *`option`***`)`**

### Parameters

*errObj*
: Is the address of a pointer to a LoadLeveler error object that was returned by one of the resource manager API subroutines.

*option*
: Is an enum specifying the disposition of the error data. Possible values are:

**LLR_ERROR_FREE**
: The error object is freed.

**LLR_ERROR_PRINT_STDERR**
: The error string is written to stderr and the error object is freed.

**LLR_ERROR_PRINT_STDOUT**
: The error string is written to stdout and the error object is freed.

**LLR_ERROR_RETURN_STRING**
: A copy of the error string is returned to the caller and the error object is freed. The caller is responsible for freeing the returned error string.

### Description

It is the caller's responsibility to free the storage associated with the error message string.

The LoadLeveler error object pointed to by **errObj* is deleted upon exit and NULL is assigned to **errObj*.

### Return Values

**!NULL**
: A pointer to a character string containing the error message if the **LLR_ERROR_RETURN_STRING** option was specified. The error object has been freed. The caller is responsible for freeing the returned string.

**NULL**
: The error object has been freed.

## llr_free_job subroutine

Use the **llr_free_job** subroutine to free the memory associated with the job object passed as an argument.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_free_job (llr_resmgr_handle_t *rm_handle, llr_element_t **jobObj,
                  llr_element_t **errObj)
```

### Parameters

*rm_handle*
: Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*jobObj*
: Is the address of a pointer to a job object returned from the **llr_add_job** subroutine. The memory associated with the job object will be freed and the pointer set to NULL.

*errObj*
: Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

### Description

*rm_handle* and *jobObj* are the input fields for this subroutine.

The memory associated with the job object pointed to by **jobObj* is freed. The **jobObj* pointer is set to NULL.

### Return Values

**LLR_API_OK**
: The job data was freed.

**LLR_API_COMM_ERROR**
: The job data could not be freed. A description of the error is provided in the *errObj* parameter.

### Related Information

Subroutines: **llr_add_job**, **llr_control_job**, **llr_delete_job**, **llr_start_job_step**

## llr_move_spool subroutine

Use the **llr_move_spool** subroutine to move job records from the **spool** directory of one job manager daemon to another job manager daemon in the local cluster.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_move_spool (llr_resmgr_handle_t * rm_handle,
                    llr_move_spool_param_t ** param, int (*func) (char *jobid,
                    int rc, llr_element_t **messageObj), llr_element_t **errObj);
```

### Parameters

*rm_handle*
Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*param*
Provides the address of a pointer to an **llr_move_spool_param_t** structure. The structure contains the following fields:

**job** *manager_host*
Is a pointer to an array of character strings containing the name of the machine to which the spool is to be moved. Only one element is currently supported.

*spool_directory*
Is a pointer to a character string containing the **spool** directory name.

*data* Is an enum of type **llr_move_spool_data_t** to indicate the type of data to be moved.

**(*func) (char *jobid, int rc, llr_element_t **messageObj)**
Specifies the user-supplied function to be called after every job is processed. The function must return an integer and must accept as input:
- A char pointer representing the job ID processed.
- An integer representing the return code of the job processing.
- The address to a pointer of an **llr_element_t** data type representing any generated messages. The **llr_element_t** data type is defined in the **llrapi.h** file. If a callback is not desired, the function pointer should be set to NULL.

*errObj*
Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

### Description

The **llr_move_spool( )** subroutine is the API for the **llrmovespool** command.

The **llr_move_spool** subroutine is intended for recovery purposes only. This subroutine moves the job records from the *spool_directory* of one managing *job_manager_host* to another managing *job_manager_host* in the local cluster. The **llr_move_spool** subroutine must be run from a machine that has read and write access to the specified *spool_directory* containing the job records being moved. This machine must also have network connectivity to the machine running the Schedd daemon that will receive the job records. Jobs to be moved can be in any state.

The *job_manager_host* that created the job records to be moved must not be running during the move operation. Jobs within the job queue database will be unrecoverable if the job queue is updated during the move by any process other than the **llr_move_spool** subroutine. The *job_manager_host* that created the job records to be moved must have the **schedd_fenced** machine stanza keyword set to **true** prior to the **llr_move_spool** subroutine being issued.

All moved jobs retain their original job identifiers. The **llr_move_spool** subroutine invokes the function specified by the (**func*) parameter after each job is processed. When the job records for a job are successfully transferred, the *job_manager_host* of the job is updated to represent the new managing *job_manager_host* and the job records in the specified *spool_directory* are deleted. If the transfer for any step within a job fails, the job records for that step remain in the specified *spool_directory*. If for some reason a job fails, the **llr_move_spool** subroutine should be reissued against the specified *spool_directory* to reprocess the job.

The user-supplied function enables messages to be displayed as each job is processed without having to wait for all of the jobs to be completed. If a callback is not desired, the function pointer should be set to NULL. The callback function must call the **llr_error** API passing in the *messageObj* in order to free the memory allocated to it.

The **llr_move_spool** subroutine does not move the reservation queue or fair share scheduling data found within the specified *spool_directory*.

Only administrators can issue the **llr_move_spool** subroutine.

## Return Values

**LLR_API_OK**
: The request transaction was successfully sent.

**LLR_API_COMM_ERROR**
: The request failed because of a communication error with a resource manager daemon. A description of the error is provided in the *errObj* parameter. The function can be retried when this error occurs.

**LLR_API_ERROR**
: The request failed. A description of the error is provided in the *errObj* parameter.

## Related Information

Commands: **llrmovespool**

Subroutines: **llr_cluster_auth**, **llr_control**

## llr_remove_energy_tags subroutine

Use the **llr_remove_energy_tags** subroutine to remove the energy policy tags from the database.

### Library

LoadLeveler API library **libllrapi.so** (Linux)

### Syntax

```
int llr_remove_energy_tags(llr_resmgr_handle_t *rm_handle, const char* uname,
const char* etag, const char *job_id, time_t target, int *count,
llr_element_t **errObj)
```

### Parameters

*rm_handle*
: Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*uname*
: Is the user name.

*etag*
: Is the energy tag name.

*job_id*
: Is a pointer to a character string containing the resource manager job ID or step ID.

*target*
: Specifies energy tags be removed if they were not used since the time specified.

*count*
: Is the count of energy tags removed from the database.

*errObj*
: Is the address of a pointer to a LoadLeveler error object. The pointer to **llr_element_t** should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

### Description

*rm_handle uname*, *etag*, *job_id* and *target* are the input fields for this subroutine.

If you want to remove any energy tag generated by any user, LoadLeveler's administrator permission is required, or only the tags generated by *uname* can be removed. If *uname* is not specified (that is, NULL), this subroutine will remove the energy tags generated by the user who is running the program. *uname* should never be **root** because no energy tags are generated by **root**.

You can set *uname*, *etag*, and *job_id* to NULL and set deadline to 0 and all the energy tags generated by the user who is running the program will be removed.

## Return Values

**LLR_API_OK**
The energy policies were successfully removed from the database. The number of tags removed is returned by the count parameter.

**LLR_API_COMM_ERROR**
The remove request failed because of a communication error with a resource manager daemon. A description of the error is provided in the *errObj* parameter. The function can be retried when this error occurs.

**LLR_API_ERROR**
Failed to remove energy policies from the database. A description of the error is provided in the *errObj* parameter.

## llr_start_job_migration subroutine

Use the **llr_start_job_migration** subroutine to allow a scheduling application to specify the destination nodes and initiate the migration operation.

### Library

LoadLeveler API library **libllapi.a** (AIX) or **libllapi.so** (Linux)

### Syntax

```
#include "llapi.h"

int llr_start_job_migration(llr_resmgr_handle_t *rm_handle, char *step_id,
char **from_list,llr_job_step_resource_t *resource, llr_element_t **err_obj)
```

### Parameters

*rm_handle*
: The pointer to the opaque structure returned from the **llr_init_resmgr** subroutine.

*step_id*
: A pointer to a character string containing the resource manager ID for the job step to be migrated.

*from_list*
: A pointer to an array of character string pointers that contain the names of the machines from which the job step will be migrated. The number of entries in the array must be the same as the number of entries in the machine_list array specified with the resource parameter.

*resource*
: A pointer to a structure specifying the machine and CPU resources to which the job step will be migrated.

  The **llr_start_job_migration** structure contains the following fields:

  *machine_count*
  : An integer indicating the number of machines to which the job step will be migrated.

  *machine_list*
  : A pointer to an array of **llr_machine_data** structures that contain data for the machines to which the job step will be migrated. The number of entries in the array is *machine_count*.

    All other fields of the llr_job_step_resource are ignored for migration.

  The **llr_machine_data** structure contains the following fields:

  *machine_name*
  : A pointer to a character string that contains the name of the machine to which the job step is migrated.

  *task_count*
  : An integer indicating the number of tasks to be migrated to the machine. The value must equal the task count on the corresponding machine from which the job step is being migrated.

  *task_list*
  : A pointer to an array of **llr_task_data** structures that contain the task ID and CPU data for the tasks. The number of entries in the array is *task_count*. The task IDs must be the same as the tasks IDs on the

corresponding machine from which the job step is being migrated. If *task_list* is NULL then the resource manager will use the CPU assignments from the original machine.

The **llr_task_data** structure contains the following fields:

*task_id*
An integer specifying the task ID.

*cpu_count*
An integer indicating the number of CPUs to be assigned to the task specified in the *task_id* field.

*cpu_list*
A pointer to an array of integers. Each entry contains the ID of a CPU to assign to the task. The number of entries in the array is *cpu_count*.

*err_obj*
Is the address of a pointer to a LoadLeveler error object. Set the pointer to the error object to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the error object can be displayed using the **llr_error** function. The caller should use the **llr_error** function to free the error object storage before reusing the pointer.

## Description

The **llr_start_job_migration** interface is called to initiate a migration operation. The tasks on the machines specified in the *from_list* parameter will be stopped and moved to the machines specified in the *resource* parameter. If the tasks were assigned to specific CPUs on the source machines, then the tasks will be assigned to the same CPUs unless a new set of CPU assignments were specified in the *resource* parameter.

## Return Values

If successful, 0 is returned. If an error occurs, a non-zero value is returned and an error message can be obtained through the *err_obj* pointer.

## llr_start_job_step subroutine

Use the **llr_start_job_step** subroutine to start a specified job step on the specified resources.

### Library

LoadLeveler API library **libllrapi.a** (AIX) or **libllrapi.so** (Linux)

### Syntax

```
int llr_start_job_step (llr_resmgr_handle_t *rm_handle, const char *step_id,
                        llr_job_step_resource_t *resource,
                        llr_job_step_opts_t *opts, llr_element_t **errObj)
```

### Parameters

*rm_handle*
Is the pointer to an opaque structure that is returned from the **llr_init_resmgr** subroutine.

*step_id*
Is the pointer to a character string that contains the resource manager ID for the job step to be started.

*resource*
Is the pointer to a structure that specifies the machine, CPU, and adapter resources from which the job step is to be started.

The **llr_job_step_resource** structure contains the following fields:

*machine_count*
Is an integer indicating the number of machines from which the job step is to be started.

*machine_list*
Is a pointer to an array of **llr_machine_data** structures that contain data for the machines from which the job step will run. The number of entries in the array is *machine_count*.

*network_usage_count*
Is an integer indicating the number of network usages that the job step will use.

*network_usage_list*
Is a pointer to an array of **llr_network_usage** structures that contain data for the networks that the job step will use. The number of entries in the array is *network_usage_count*.

The **llr_network_usage** structure contains the following fields:

*endpoints*
Is the number of endpoints that can be used by each task per protocol instance.

*network_id*
Is a 64–bit unsigned integer identifier that uniquely identifies the network.

*instances*
Is an integer that specifies the number of adapters and links to be used for striping within the network.

*window_count*
Is an integer that specifies the number of user space windows on each adapter and link to be used by user space job tasks.

*adapter_memory*
Is an integer that specifies the amount of memory on each adapter and link to be used by user space job tasks.

*protocols*
Is a string that specifies the protocols used for communications. The string contains either the individual protocol name, or multiple protocols (shared mode) names separated by an underscore (_) character.

*mode* Is an enum that indicates either US or IP mode.

*network_type*
Is an enum that indicates either **LLR_SWITCH_NETWORK**, **LLR_ETHERNET_NETWORK**, or **LLR_MULTILINK_NETWORK**.

*collective_groups*
Is the number of collective groups created for a protocol instance.

*imm_send_buffers*
Is the number of immediate send buffers allocated for each window.

The **llr_machine_data** structure contains the following fields:

*machine_name*
Is a pointer to a character string that contains the name of the machine from which the job step is started.

*task_count*
If *task_list* is NULL, then *task_count* is an integer indicating the number of tasks to be started on the machine. If *task_list* is not NULL, then *task_count* is the number of entries in the array to which *task_list* points.

*task_list*
Is a pointer to an array of **llr_task_data** structures that contain task ID and CPU data for the tasks. The number of entries in the array is *task_count*. If *task_list* is NULL, the resource manager will assign the task IDs.

The **llr_task_data** structure contains the following fields:

*task_id* Is an integer specifying the task ID.

*cpu_count*
Is an integer indicating the number of CPUs to be assigned to the tasks specified in the *task_id* field.

*cpu_list*
Is a pointer to an array of integers. Each entry contains the ID of a CPU to assign to the task. The number of entries in the array is *cpu_count*.

`opts`
The argument should be set to NULL. It is reserved for future extension.

`errObj`
Is the address of a pointer to a LoadLeveler error object. The pointer to the error object should be set to NULL before calling this function. If this function fails, the pointer will point to an error object. The error messages stored in the

error object can be displayed using the **llr_error** subroutine. The caller should use the **llr_error** subroutine to free the error object storage before reusing the pointer.

## Description

The **llr_start_job_step** subroutine specifies the resources to which a job step is to be started. The specified resources do not have to match the requirements used to add the job. The resource manager will not perform any policy checking on resources.

This function must be called by a LoadLeveler administrator.

## Return Values

**LLR_API_OK**
: The start job request was successfully sent to the resource manager.

**LLR_API_COMM_ERROR**
: The job could not be sent to the resource manager because of a communication error with a resource manager daemon. A description of the error is provided in the *errObj* parameter. The function can be retried when this error occurs.

**LLR_API_ERROR**
: The job was not sent to the resource manager. A description of the error is provided in the *errObj* parameter.

## Related Information

Subroutines: **llr_add_job**, **llr_control_job**, **llr_delete_job**, **llr_free_job**

# Part 4. Appendixes

# Appendix A. llrq -l command output listing

This listing shows the output from **llrq -l** command.

```
=============== Job Step c596n17.ppd.pok.ibm.com.50.0@c596n17.ppd.pok.ibm.com ===============
        Job Step Id: c596n17.ppd.pok.ibm.com.50.0@c596n17.ppd.pok.ibm.com
           Job Name: hostname
          Step Name: 0
  Structure Version: 10
              Owner: loadl
         Queue Date: Tue 20 Dec 2011 01:10:33 AM EST
             Status: Running
   Eligibility Time: Tue 20 Dec 2011 01:10:33 AM EST
      Dispatch Time: Tue 20 Dec 2011 01:10:33 AM EST
    Completion Date:
    Completion Code:
      Notifications: Error
 Virtual Image Size: 1 kb
         Large Page: N
              Trace: no
         Coschedule: no
       SMT required: as_is
    MetaCluster Job: no
     Checkpointable: no
    Checkpoint File:
   Ckpt Execute Dir:
  Restart From Ckpt: no
 Restart Same Nodes: no
            Restart: yes
        Preemptable: yes
 Preempt Wait Count: 0
     Hold Job Until:
               RSet: RSET_NONE
Mcm Affinity Option:
      Task Affinity:
      Cpus Per Core:  0
   Parallel Threads:  0
                Cmd: /home/loadl/bm.cmd
               Args:
                Env:
                 In: /dev/null
                Out: j3.50.out
                Err: j3.50.err
Initial Working Dir: /home/loadl
         Dependency:
Data Stg Dependency:
          Resources:
     Node Resources:
     Step Resources:
       Requirements:
        Preferences:
          Step Type: Serial
     Min Processors:
     Max Processors:
     Allocated Host: c596n17.ppd.pok.ibm.com
         Node Usage: not_shared
    Submitting Host: c596n17.ppd.pok.ibm.com
        Schedd Host: c596n17.ppd.pok.ibm.com
      Job Queue Key:
        Notify User: loadl@c596n17.ppd.pok.ibm.com
              Shell: /bin/bash
  LoadLeveler Group: No_Group
              Class: No_Class
    Ckpt Hard Limit: undefined
    Ckpt Soft Limit: undefined
     Cpu Hard Limit: undefined
     Cpu Soft Limit: undefined
    Data Hard Limit: undefined
    Data Soft Limit: undefined
      As Hard Limit: undefined
      As Soft Limit: undefined
   Nproc Hard Limit: undefined
   Nproc Soft Limit: undefined
 Memlock Hard Limit: undefined
 Memlock Soft Limit: undefined
   Locks Hard Limit: undefined
   Locks Soft Limit: undefined
  Nofile Hard Limit: undefined
  Nofile Soft Limit: undefined
    Core Hard Limit: undefined
    Core Soft Limit: undefined
    File Hard Limit: undefined
    File Soft Limit: undefined
   Stack Hard Limit: undefined
   Stack Soft Limit: undefined
     Rss Hard Limit: undefined
     Rss Soft Limit: undefined
Step Cpu Hard Limit: undefined
Step Cpu Soft Limit: undefined
Wall Clk Hard Limit: 00:30:00 (1800 seconds)
Wall Clk Soft Limit: 00:30:00 (1800 seconds)
            Comment:
            Account:
         Unix Group: loadl
      Bulk Transfer: No
Adapter Requirement: (sn_single,mpi,US,not_shared,AVERAGE,instances=1,imm_send_buffers=1,collective_groups=3,endpoints=1)
          Step Cpus: 0
Step Virtual Memory: 0.000 mb
   Step Real Memory: 0.000 mb
Step Large Page Mem: 0.000 mb
Topology Requirement: none
     Network Usages:
       Scheduler ID: _LoadLevler_scheduler_ID_
    Monitor Program:
  Energy Policy Tag: long_job_energy_tag_1
  Perf. Degradation: 5%
Req. Energy Savings:
      CPU Frequency: 2.66 GHZ
Est. Power Consumed: 0.000045 KWH
Est. Execution Time: 00:04:52 (292 seconds)

1 job step(s) in queue, 0 waiting, 0 pending, 1 running, 0 held, 0 preempted
```

# Appendix B. llrstatus -a command output listing

This listing shows the output from **llrstatus -a** on a machine connected to a switch network.

```
================================================================================
c250f10c04ap09.ppd.pok.ibm.com
sit0(ethernet,0.0.0.0,0.0.0.0,,READY,HB_UNKNOWN,READY)
network8078501872035430s(striped,,,,-1,440,0 rCxt Blks,256 Imm Send Buffers,,1,)
network80785018720354304(aggregate,,,,-1,440,0 rCxt Blks,256 Imm Send Buffers,,1,)
hf1(hfi,21.10.4.9,21.10.4.9,,0,220,0 rCxt Blks,128 Imm Send Buffers,1,,READY,HB_UP,READY,,,MCM-1)
hf0(hfi,20.10.4.9,20.10.4.9,,0,220,0 rCxt Blks,128 Imm Send Buffers,1,,READY,HB_UP,READY,,,MCM-1)
en0(ethernet,c250f10c04ap09.ppd.pok.ibm.com,10.10.4.9,,READY,HB_UP,READY)

================================================================================
c250f10c04ap13.ppd.pok.ibm.com
sit0(ethernet,0.0.0.0,0.0.0.0,,READY,HB_UNKNOWN,READY)
network8078501872035430s(striped,,,,-1,440,0 rCxt Blks,256 Imm Send Buffers,,1,)
network80785018720354304(aggregate,,,,-1,440,0 rCxt Blks,256 Imm Send Buffers,,1,)
hf1(hfi,21.10.4.13,21.10.4.13,,0,220,0 rCxt Blks,128 Imm Send Buffers,1,,READY,HB_UP,READY,,,MCM-1)
hf0(hfi,20.10.4.13,20.10.4.13,,0,220,0 rCxt Blks,128 Imm Send Buffers,1,,READY,HB_UP,READY,,,MCM-1)
en0(ethernet,c250f10c04ap13.ppd.pok.ibm.com,10.10.4.13,,READY,HB_UP,READY)

================================================================================
c250f10c04ap05.ppd.pok.ibm.com
sit0(ethernet,0.0.0.0,0.0.0.0,,READY,HB_UNKNOWN,READY)
network8078501872035430s(striped,,,,-1,440,0 rCxt Blks,256 Imm Send Buffers,,1,)
network80785018720354304(aggregate,,,,-1,440,0 rCxt Blks,256 Imm Send Buffers,,1,)
hf1(hfi,21.10.4.5,21.10.4.5,,0,220,0 rCxt Blks,128 Imm Send Buffers,1,,READY,HB_UP,READY,,,MCM-1)
hf0(hfi,20.10.4.5,20.10.4.5,,0,220,0 rCxt Blks,128 Imm Send Buffers,1,,READY,HB_UP,READY,,,MCM-1)
en0(ethernet,c250f10c04ap05.ppd.pok.ibm.com,10.10.4.5,,READY,HB_UP,READY)

================================================================================
c250f10c04ap01.ppd.pok.ibm.com
sit0(ethernet,0.0.0.0,0.0.0.0,,READY,HB_UNKNOWN,READY)
network8078501872035430s(striped,,,,-1,440,0 rCxt Blks,256 Imm Send Buffers,,1,)
network80785018720354304(aggregate,,,,-1,440,0 rCxt Blks,256 Imm Send Buffers,,1,)
hf1(hfi,21.10.4.1,21.10.4.1,,0,220,0 rCxt Blks,128 Imm Send Buffers,1,,READY,HB_UP,READY,,,MCM-1)
hf0(hfi,20.10.4.1,20.10.4.1,,0,220,0 rCxt Blks,128 Imm Send Buffers,1,,READY,HB_UP,READY,,,MCM-1)
en0(ethernet,c250f10c04ap01.ppd.pok.ibm.com,10.10.4.1,,READY,HB_UP,READY)
```

# Appendix C. llrstatus -l command output listing

This listing shows the output from **llrstatus -L machine -l** or **LOADL_STATUS_LEVEL=machine llrstatus -l** on a machine connected to a switch network.

```
===============================================================================
Name                = c445f1n07.ppd.pok.ibm.com
Machine             = c445f1n07.ppd.pok.ibm.com
Machine Group       =
Arch                = x86_64
OpSys               = Linux2
Island              =
VirtualMemory       = 0 kb
Disk                = 3414240 kb
KeyboardIdle        = 9999
Tmp                 = 188480 kb
LoadAvg             = 0.000000
ConfiguredClasses   = No_Class(1)
Pool                =
FabricConnectivity  = 567488:1,144115188076488896:1
Adapter             = mlx4_0(InfiniBand,,,,-1,0,0 rCxt Blks,,11,)
                      network56748s(striped,,,,-1,256,0 rCxt Blks,,11,)
                      network567488(aggregate,,,,-1,256,0 rCxt Blks,,1,)
                      network144115188076488896(aggregate,,,,-1,256,0 rCxt Blks,,1,)
                      eth0(ethernet,c445f1n07.ppd.pok.ibm.com,9.114.101.7,,READY,HB_UP,READY)
Feature             =
Max_Starters        = 1
Max_Dstg_Starters   = 0
Prestarted_Starters = 1
Total Memory        = 32220 mb
Memory              = 32220 mb
FreeRealMemory      = 25708 mb
LargePageSize       = 2.000 mb
LargePageMemory     = 0 kb
FreeLargePageMemory = 0 kb
PagesFreed          = 0
PagesScanned        = 0
PagesPagedIn        = 0
PagesPagedOut       = 0
ConsumableResources = ConsumableCpus(16)+
ConfigTimeStamp     = Mon Apr 23 04:24:57 2012
Cpus                = 16
RSetSupportType     = RSET_NONE
Speed               = 1.000000
Subnet              = 9.114.101
MasterMachPriority  = 0.000000
CustomMetric        = 1
StartdAvail         = 1
State               = Idle
EnteredCurrentState = Mon Apr 23 04:24:57 2012
START               = T
SUSPEND             = F
CONTINUE            = T
VACATE              = F
KILL                = F
Machine Mode        = general
ScheddAvail         = 0
ScheddState         = Down
ScheddRunning       = 0
Pending             = 0
Starting            = 0
Idle                = 0
Unexpanded          = 0
Held                = 0
Removed             = 0
RemovedPending      = 0
Completed           = 0
TotalJobs           = 0
Running Steps       =
SMT                 = Not Supported
Region              = regiona
TimeStamp           = Mon Apr 23 04:24:59 2012
Current Frequency   = 2.53 GHZ
Supported Freq. List= 2.66 GHZ,2.53 GHZ,2.39 GHZ,2.26 GHZ,2.13 GHZ,2.00 GHZ,1.86 GHZ,1.73 GHZ,1.60 GHZ
Power Policy        = 55 10 * * *;11
Power State         = Standby
```

# Appendix D. llrstatus -M command output listing

This listing shows the output from the **llrstatus -M** command.

```
Machine               MCM details
--------------------- --------------------------------------------------
c890f14ec03.ppd.pok.ibm.com
             MCM0
                Available Cpus :< 0-31 >(32)
                Adapters       :
             MCM1
                Available Cpus :< 32-63 >(32)
                Adapters       :network18338657682652659714[64,0 rCxt Blks]
                                network18338657682652659713[64,0 rCxt Blks]
             MCM2
                Available Cpus :< 64-95 >(32)
                Adapters       :
             MCM3
                Available Cpus :< 96-127 >(32)
                Adapters       :
```

# Appendix E. LoadLeveler port usage

This topic describes LoadLeveler port usage.

A **port number** is an integer that specifies the port to use to connect to the specified daemon. For most ports used by LoadLeveler, you can define the port numbers in the configuration file or the **/etc/services** file or you can accept the defaults. LoadLeveler first looks in the configuration file for these port numbers. If LoadLeveler does not find the value in the configuration file, it looks in the **/etc/services** file. If the value is not found in this file, the default is used.

There are two exceptions to this rule:
1. The **LoadL_master_config** service cannot be specified in the LoadLeveler configuration data. This special port is used by LoadLeveler to retrieve configuration data from LoadLeveler nodes configured in the master configuration file as **LoadLConfigHost**. This port number can be specified in **/etc/services**.
2. Port numbers used by **sshd** daemons started by LoadLeveler for interactive jobs cannot be specified in **/etc/services**. A range of port numbers are used by LoadLeveler when starting an **sshd** daemon for an interactive job. This range can be configured using the **SSHD_PORTS** configuration keyword.

**Note:** See Table 38 on page 330 for the configuration file keywords associated with the port numbers.

The first column on each line in Table 38 on page 330 represents the name of a service. In most cases, these services are also the names of daemons with the following exceptions:
- **LoadL_schedd_status** is the service name for a second stream port used by the **LoadL_schedd** daemon.
- **LoadL_master_config** is the service name for a second stream port used by a **LoadL_master** daemon which is a configuration server when the database option is being used.

For each LoadLeveler service definition shown in Table 38 on page 330, the following information is shown:

**Service name**
Specifies the service name. The service names shown are examples of how the names might appear in the **/etc/services** file.

**Port number**
Specifies the port number used for the service.

**Protocol name**
Specifies the transport protocol used for the service.

**Source port range**
A range of port numbers used on either the client side or daemon (server) side of the service.

**Required or optional**
Whether or not the service is required.

**Description/associated keywords**
A short description of the service along with its associated configuration file keyword or keywords.

*Table 38. LoadLeveler default port usage*

| Service name | Port number | Protocol name | Source port range[1] | Required or optional | Description/associated keywords |
|---|---|---|---|---|---|
| LoadL_master | 9616 | tcp | LB | Required | Master port number for stream port<br><br>Keyword:<br><br>**MASTER_STREAM_PORT** (tcp) |
| | 9617 | udp | LB | Required | Master port number for dgram port<br><br>Keyword:<br><br>**MASTER_DGRAM_PORT** (udp) |
| LoadL_master_config | 9601 | tcp | LB | Required | Master port number for configuration stream port<br><br>Keyword: None. This port cannot be specified in the LoadLeveler configuration; only in **/etc/services**. |
| LoadL_region_mgr | 9680 | tcp | LB | Required | Region manger port number for stream port<br><br>Keyword:<br><br>**REGION_MGR_STREAM_PORT** |
| | 9684 | udp | LB | Required | Required adapter heartbeat port number<br><br>Keyword:<br><br>**ADAPTER_HEARTBEAT_PORT** (udp) |
| LoadL_resource_mgr | 9618 | tcp | LB | Required | Resource manager port number for stream port<br><br>Keyword:<br><br>**RESOURCE_MGR_STREAM_PORT** |
| | 9619 | udp | LB | | Resource manager port number for dgram port<br><br>Keyword:<br><br>**RESOURCE_MGR_DGRAM_PORT** (udp) |
| LoadL_schedd | 9605 | tcp | LB | Required | Schedd port number for stream port<br><br>Keyword:<br><br>**SCHEDD_STREAM_PORT** |

*Table 38. LoadLeveler default port usage (continued)*

| Service name | Port number | Protocol name | Source port range[1] | Required or optional | Description/associated keywords |
|---|---|---|---|---|---|
| LoadL_schedd_status | 9606 | tcp | LB | Required | Schedd stream port for job status data<br><br>Keyword:<br><br>**SCHEDD_STATUS_PORT** |
| LoadL_startd | 9611 | tcp | LB | Required | Startd port number for stream port<br><br>Keyword:<br><br>**STARTD_STREAM_PORT** |
| | 9615 | udp | LB | | Startd port number for dgram port<br><br>Keyword:<br><br>**STARTD_DGRAM_PORT** (udp) |
| N/A | 9620-9629 | tcp | LB | Required | Range of port numbers used by LoadLeveler when starting **sshd** daemons for interactive jobs<br><br>Keyword:<br><br>**SSHD_PORTS** |

**Note:** [1]A value of LB indicates that the source port range value should be left blank. In other words, no source port range value should be specified.

For more information about configuration file keyword syntax and configuring the LoadLeveler environment, see the following:

- Chapter 3, “Configuring the LoadLeveler resource manager environment,” on page 23
- Chapter 6, “Configuration keyword reference,” on page 91

# Accessibility features for LoadLeveler

Accessibility features help users who have a disability, such as restricted mobility or limited vision, to use information technology products successfully.

## Accessibility features

The following list includes the major accessibility features in IBM LoadLeveler:

- Keyboard-only operation
- Interfaces that are commonly used by screen readers
- Keys that are discernible by touch but do not activate just by touching them
- Industry-standard devices for ports and connectors
- The attachment of alternative input and output devices

The *IBM Cluster information center*, and its related publications, are accessibility-enabled. The accessibility features of the information center are described in the IBM Cluster information center (http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/topic/com.ibm.cluster.addinfo.doc/access.html).

## Keyboard navigation

This product uses standard Microsoft Windows navigation keys.

## IBM and accessibility

See the IBM Human Ability and Accessibility Center (http://www.ibm.com/able/) for more information about the commitment that IBM has to accessibility.

# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

## Trademarks

# Glossary

This glossary includes terms and definitions for LoadLeveler.

The following cross-references are used in this glossary:

- See refers you from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
- See also refers you to a related or contrasting term.

To view glossaries for other IBM products, go to IBM Terminology (http://www-01.ibm.com/software/globalization/terminology/index.jsp).

## A

**AFS** A distributed file system for large networks that is known for its ease of administration and expandability.

**AIX** A UNIX operating system developed by IBM that is designed and optimized to run on POWER microprocessor-based hardware such as servers, workstations, and blades.

**authentication**
The process of validating the identity of a user or server.

**authorization**
The process of obtaining permission to perform specific actions.

## B

**Berkeley Load Average**
The average number of processes on the operating system's ready-to-run queue.

## C

**C language**
A language used to develop application programs in compact, efficient code that can be run on different types of computers with minimal change.

**client** A system or process that is dependent on another system or process (usually called the *server*) to provide it with access to data, services, programs, or resources.

**cluster**
A collection of complete systems that work together to provide a single, unified computing capability.

## D

**daemon**
A program that runs unattended to perform continuous or periodic functions, such as network control.

**DCE** See *Distributed Computing Environment*.

**default**
Pertaining to an attribute, value, or option that is assumed when none is explicitly specified.

**DFS** See *Distributed File System*.

**Distributed Computing Environment (DCE)**
In network computing, a set of services and tools that supports the creation, use, and maintenance of distributed applications across heterogeneous operating systems and networks.

**Distributed File Service (DFS)**
A component of a Distributed Computing Environment (DCE) that enables a single, integrated file system to be shared among all DCE users and host computers in a DCE cell. DFS prevents DCE users from simultaneously modifying the same information.

## F

**flexible job**
A job step that is only used for scheduling resources for a flexible reservation.

**flexible reservation**
A reservation that starts as soon as resources are first available.

## H

**host** A computer that is connected to a network and provides an access point to that network. The host can be a client, a server, or both a client and server simultaneously.

## L

**LAPI** See *low-level application programming interface*.

**low-level application programming interface (LAPI)**
An IBM message-passing interface that implements a one-sided communication model.

## M

**MCM** See *multiple chip module*.

**memory affinity**
A feature available in AIX to allocate memory attached to the same multiple chip module (MCM) on which the process runs. Memory affinity improves the performance of applications on IBM System p® servers.

**menu** A displayed list of items from which a user can make a selection.

**Message Passing Interface (MPI)**
A library specification for message passing. MPI is a standard application programming interface (API) that can be used with parallel applications and that uses the best features of a number of existing message-passing systems.

**MPI** See *Message Passing Interface*.

**MPICH2**
A portable implementation of the Message Passing Interface (MPI).

**multiple chip module (MCM)**
The fundamental, processor, building block of IBM System p servers.

## N

**network**
In data communication, a configuration in which two or more locations are physically connected for the purpose of exchanging data.

**Network File System (NFS)**
A protocol, developed by Sun Microsystems, Incorporated, that enables a computer to access files over a network as if they were on its local disks.

**NFS** See *Network File System*.

**node** A computer location defined in a network.

**nominal CPU frequency**
The vendor specified frequency for the CPU.

## P

**parameter**
A value or reference passed to a function, command, or program that serves as input or controls actions. The value is supplied by a user or by another program or process.

**process**
A separately executable unit of work.

## R

**RDMA**
See *Remote Direct Memory Access*.

**Remote Direct Memory Access (RDMA)**
A communication technique in which data is transmitted from the memory of one computer to that of another without passing through a processor. RDMA accommodates increased network speeds.

**resource set (RSet)**
A data structure in AIX used to represent physical resources such as processors and memory. AIX uses resource sets to restrict a set of processes to a subset of the system's physical resources.

**RSet** See *resource set*.

## S

**S3 state**
A power state where everything in the system is put into a low-power state except for memory.

**server** In a network, hardware or software that provides facilities to clients. Examples of a server are a file server, a printer server, or a mail server.

**shell** A software interface between users and an operating system. Shells generally fall into one of two categories: a command line shell, which provides a command line interface to the operating system; and a graphical shell, which provides a graphical user interface (GUI).

**simultaneous multithreading (SMT)**
Pertaining to a processor design that combines hardware multithreading with superscalar processor technology. Using

SMT, a single physical processor emulates multiple processors by enabling multiple threads to issue instructions simultaneously during each cycle.

**SMT** See *simultaneous multithreading*.

**system administrator**
The person who controls and manages a computer system.

## T

**TCP** See *Transmission Control Protocol*.

**Transmission Control Protocol (TCP)**
A communication protocol used in the Internet and in any network that follows the Internet Engineering Task Force (IETF) standards for internetwork protocol. TCP provides a reliable host-to-host protocol in packet-switched communication networks and in interconnected systems of such networks.

## U

**UDP** See *User Datagram Protocol*.

**User Datagram Protocol (UDP)**
An Internet protocol that provides unreliable, connectionless datagram service. It enables an application program on one machine or process to send a datagram to an application program on another machine or process.

## W

**workflow**
An application that has been partitioned into a complex sequence of interdependent jobs is called a workflow. The execution of jobs in a workflow may depend on the success/failure of previously executed jobs. Some jobs in a workflow may need to use output of jobs that executed before them; jobs that are not interdependent may be able to execute in parallel.

**workflow engine**
A software tool that manages the execution of a workflow. By integrating with LoadLeveler, a workflow engine can benefit from features like flexible reservations and state change notifications to automate several workflows in parallel.

**working directory**
The active directory. When a file name is specified without a directory, the current directory is searched.

**workstation**
A configuration of input/output equipment at which an operator works. A workstation is a terminal or microcomputer at which a user can run applications and that is usually connected to a mainframe or a network.

# Index

Product Number: 5725-G01
5641-LL1
5641-LL3
5765-L50
5765-LLP

Printed in USA

SC23-6790-04