NQE Administration

SG–2150 3.3

Document Number 007–3799–001

New Features

NQE Administration SG–2150 3.3

This revised *NQE Administration*, publication SG–2150, supports the 3.3 release of the Network Queuing Environment (NQE) release.

NQE system administration documentation was revised to support the following NQE 3.3 features:

- On CRAY T3E systems, NQE now supports checkpointing and restarting of jobs. This feature was initially supported in the NQE 3.2.1 release. This feature requires the UNICOS/mk 2.0 release or later.
- On CRAY T3E systems, NQE now supports the Political Scheduling feature. This feature was initially supported in the NQE 3.2.1 release. This feature requires the UNICOS/mk 2.0 release or later.
- On CRAY T3E systems, NQE now supports mixed mode scheduling.
- Applications running on a CRAY T3E system are killed when a PE assigned to the application goes down. NQS is now notified when a job is terminated by a `SIGPEFAILURE` signal (UNICOS/mk systems only). NQE will requeue the job and either restart or rerun the job, as applicable.
- For CRAY T3E systems, this release adds CPU and memory scheduling weighting factors for application PEs. The NQS scheduling weighting factors are used with the NQS priority formula to calculate the intra-queue job initiation priority for NQS runnable jobs. This release also restores user-specified priority scheduling functionality.
- The multilevel security (MLS) feature on UNICOS/mk systems is supported with this NQE release.
- Distributed Computing Environment (DCE) support was enhanced as follows:
  - Ticket forwarding and inheritance is now supported. This feature allows users to submit jobs in a DCE environment without providing passwords.
  - IRIX systems now support access to DCE resources for jobs submitted to NQE.
  - UNICOS/mk systems now support access to DCE resources for jobs submitted to NQE.

  DCE is supported on all NQE platforms except SunOs. Ticket forwarding is not supported for the Digital UNIX operating system in the NQE 3.3 release.
- For IRIX systems running NQE, this release introduces a new scheduler called the Miser scheduler. The Miser scheduler is a predictive scheduler that evaluates the number of CPUs and the amount of memory a batch will require. NQE now supports the submission of jobs that specify Miser resources.
- Array services support was added for UNICOS systems.

- The new `nqeinfo`(5) man page documents all NQE configuration variables; the `nqeinfo`(5) man page is provided in online form only and is accessible by using the `man`(1) command or through the NQE configuration utility `Help` facility.
- This release replaces the `NQE_TYPE` variable in the `nqeinfo`(5) file with a new `NQE_DEFAULT_COMPLIST` variable, which defines the list of NQE components to be started or stopped. The NQE 3.3 release is shipped with the `NQE_DEFAULT_COMPLIST` variable set to the following components: `NQS`, `COLLECTOR`, and `NLB`.
- The following NQE database enhancements were made:
  - Increased number of simultaneous connections for clients and execution servers to the NQE database.
  - The `MAX_SCRIPT_SIZE` variable was added to the `nqeinfo` file, allowing an administrator to limit the size of the script file submitted to the NQE database. If the `MAX_SCRIPT_SIZE` variable is set to 0 or is not set, a script file of unlimited size is allowed. The script file is stored in the NQE database; if the file is bigger than `MAX_SCRIPT_SIZE`, it can affect the performance of NQE database and the `nqedbmgr`. The `nqeinfo`(5) man page includes the description of this new variable.
- The `csuspend` utility has the following two new command-line options: `-l` *loopcount* and `-p` *period*. These two new options suspend or enable batch processing based on interactive use. The amount of interactive use is determined by calls to `sar`. These options give the administrator greater control over how `sar` is used and, consequently, the frequency of checking on whether to suspend or start NQE. The `csuspend`(8) man page were revised to reflect this new capability.
- The `qstart`(8) and `qstop`(8) commands now allow an administrator to execute programs immediately before and after the NQS daemon starts (`NQE_ETC/qstart.pre` and `NQE_ETC/qstart.pst`, where `NQE_ETC` is defined in the `nqeinfo` file) and immediately before and after the the NQS daemon is shut down (`NQE_ETC/qstop.pre` and `NQE_ETC/qstop.pst`, where `NQE_ETC` is defined in the `nqeinfo` file). The administrator must create the file and it must be executable. The `nqeinit`(8), `nqestop`(8), `qstart`(8), and `qstop`(8) man pages were revised to reflect this new capability.
- NQS sets several environment variables that are passed to a login shell when NQS initiates a job. One of the environment variables set is `LOGNAME`, which is the name of the user under whose account the job will run. Some platforms, such as IRIX, use the `USER` environment variable rather than `LOGNAME`. On those platforms, `csh` writes an error message into the job's `stderr` file, noting that the USER variable is not defined. To accommodate this difference, NQS now sets both the `LOGNAME` and `USER` environment variables to the same value before initiating a job. The `ilb`(1) man page were revised to include this new variable.
- Year 2000 support for NQE has been completed.
- The chapters documenting "Preparing a Node to Run NQE" and "NQE Version Maintenance" removed from this administration guide; they are included in *NQE Installation*, publication SG–5236.
- The project ID is added to the end of the current accounting records in the NQS accounting file (`nqsacct`) written by the NQS daemon accounting.

For a complete list of new features for the NQE 3.3 release, see the *NQE Release Overview*, publication RO–5237.

Record of Revision

Version	*Description*
1.0	December 1993. Original Printing. This publication describes how to use the Network Queuing Environment (NQE) release 1.0, running on UNIX or UNICOS systems.
1.1	June 1994. Incorporates information for NQE release 1.1.
2.0	May 1995. Incorporates information for the NQE 2.0 release. This publication also supports Network Queuing EXtentions (NQX) synchronous with the UNICOS 9.0 release.
3.0	March 1996. Incorporates information for the NQE 3.0 release.
3.1	September 1996. Incorporates information for the NQE 3.1 release.
3.2	February 1997. Incorporates information for the NQE 3.2 release. This document was revised and is provided in online form only for this release.
3.3	March 1998. Incorporates information for the NQE 3.3 release.

Contents

Page

Preface

This publication describes how to configure, monitor, and control the Cray Network Queuing Environment (NQE) running on a UNIX system. NQE is a software product that lets users submit, monitor, and control batch requests for execution on an NQS server in an NQE cluster. This publication is written for NQE administrators who are responsible for defining and maintaining the NQE network configuration. This publication also may be accessed online by using the Cray Research online documentation reader.

The NQE graphical user interface (GUI) allows users to submit and control batch requests to a central storage database and to obtain status on their batch requests and file transfers.

The Network Load Balancer (NLB) uses the system load information received from NQE execution servers to offer NQS an ordered list of servers to run a request; NQS uses the list to distribute the request.

The NQE database provides an alternate mechanism for distributing work. Requests are submitted and stored centrally. The NQE scheduler examines each request and determines when and where the request is run.

The File Transfer Agent (FTA) provides asynchronous and synchronous file transfer. You can queue your transfers so that they are retried if a network link fails.

Related Publications

The following documents contain additional information that may be helpful:

- *NQE Release Overview*, publication RO–5237, provides NQE release information.
- *NQE Installation*, publication SG–5236, describes how to install or upgrade the NQE software.
- *Introducing NQE*, publication IN–2153, provides an overview of NQE functionality and describes how to access documentation online. This publication also may be accessed online by using the Cray Research online documentation reader.
- *NQE User's Guide*, publication SG–2148, provides more detailed user-level information than *Introducing NQE*, such as how to do user tasks by using

either the NQE graphical user interface (GUI) or the command-line interface, how to customize your environment, and so on. This publication may also be accessed online by using the Cray Research online documentation reader.

Ordering Cray Research Publications

The *User Publications Catalog*, Cray Research publication CP–0099, describes the availability and content of all Cray Research hardware and software documents that are available to customers. Cray Research customers who subscribe to the Cray Inform (CRInform) program can access this information on the CRInform system.

To order a document, either call the Distribution Center in Mendota Heights, Minnesota, at +1–612–683–5907, or send a facsimile of your request to fax number +1–612–452–0141. Cray Research employees may send electronic mail to `orderdsk` (UNIX system users).

Customers who subscribe to the CRInform program can order software release packages electronically by using the `Order Cray Software` option.

Customers outside of the United States and Canada should contact their local service organization for ordering and documentation information.

Conventions

The following conventions are used throughout this document:

Convention	Meaning
`command`	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.
`manpage`(*x*)	Man page section identifiers appear in parentheses after man page names. The following list describes the identifiers:

1	User commands
1B	User commands ported from BSD
2	System calls
3	Library routines, macros, and opdefs

4	Devices (special files)
4P	Protocols
5	File formats
7	Miscellaneous topics
7D	DWB-related information
8	Administrator commands

Some internal routines (for example, the `_assign_asgcmd_info`() routine) do not have man pages associated with them.

variable	Italic typeface denotes variable entries and words or concepts being defined.
`user input`	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. Output is shown in nonbold, fixed-space font.
[ ]	Brackets enclose optional portions of a command or directive line.
...	Ellipses indicate that a preceding element can be repeated.

The default shell in the UNICOS and UNICOS/mk operating systems, referred to in Cray Research documentation as the *standard shell*, is a version of the Korn shell that conforms to the following standards:

- Institute of Electrical and Electronics Engineers (IEEE) Portable Operating System Interface (POSIX) Standard 1003.2–1992
- X/Open Portability Guide, Issue 4 (XPG4)

The UNICOS and UNICOS/mk operating systems also support the optional use of the C shell.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this document, please tell us. You can contact us in any of the following ways:

- Send us electronic mail at the following address:

  `publications@cray.com`

- Contact your customer service representative and ask that an SPR or PV be filed. If filing an SPR, use `PUBLICATIONS` for the group name, `PUBS` for the command, and `NO-LICENSE` for the release name.
- Call our Software Publications Group in Eagan, Minnesota, through the Customer Service Call Center, using either of the following numbers:

  1–800–950–2729 (toll free from the United States and Canada)

  +1–612–683–5600
- Send a facsimile of your comments to the attention of “Software Publications Group” in Eagan, Minnesota, at fax number +1–612–683–5599.

We value your comments and will respond to them promptly.

NQE Overview [1]

The Network Queuing Environment (NQE) is a framework for distributing work across a network of heterogeneous systems. NQE consists of the following components:

- Cray Network Queuing System (NQS)
- NQE clients
- Network Load Balancer (NLB)
- NQE database and its scheduler
- File Transfer Agent (FTA)

Note: Cray PVP systems that do not have an NQE license are limited to accessing and using only the NQE subset (NQS and FTA components).

After you have installed NQE, follow the instructions in Chapters 3 through 6 of this administrator's guide. For information about the content of this guide, see Section 1.1.

1.1 Scope of This Manual

This manual provides information on how to configure and manage NQE. This manual contains the following chapters:

- Chapter 1, page 1 (this chapter) describes the components of NQE and provides an overview of NQE.
- Chapter 2, page 11 describes concepts and terms relevant to NQE. It is meant to act as an introduction for administrators unfamiliar with batch concepts; it also can act as a reference for more experienced administrators.
- Chapter 3, page 35 describes how to use the `nqeconfig`(8) utility to configure the NQE configuration file (`nqeinfo` file).
- Chapter 4, page 45 describes how to start and stop NQE. It also describes the list of valid NQE components set in the `NQE_DEFAULT_COMPLIST` variable in the `nqeinfo`(5) file used by the `nqeinit`(8) and `nqestop`(8) scripts.

- Chapter 5, page 47 describes configuration and management of the NQS component of NQE. It describes the use of `qmgr` commands that are available only to NQS managers.
- Chapter 6, page 133 describes the `qmgr` commands that are available to NQS operators.
- Chapter 7, page 173 describes configuration and management of the NLB components of NQE.
- Chapter 8, page 193 describes how to implement load-balancing policies with NQE.
- Chapter 9, page 231 provides an overview of the NQE database and its components, and it describes management and configuration of the NQE database and the NQE scheduler.
- Chapter 10, page 261 describes how to modify the NQE scheduler to meet the needs of your site.
- Chapter 11, page 307 describes how to use the `csuspend`(8) command to make use of unused cycles on a server.
- Chapter 12, page 313 describes how the job dependency feature affects NQE administration.
- Chapter 13, page 317 describes configuration and management of the FTA component of NQE.
- Chapter 14, page 351 describes configuration of the Distributed Computing Environment (DCE) when using NQE.
- Chapter 15, page 361 describes configuration of the `ilb`(1) command.
- Chapter 16, page 365 provides possible solutions to problems you may encounter as an NQE administrator.
- Appendixes provide supplemental information to help you administer NQE.

1.2 NQE Environment

NQE supports computing with a large number of nodes in a large network supporting two basic models:

- The NQE database model that supports up to 36 servers and hundreds of clients

- The NQS model that supports an unlimited number of servers and hundreds of clients

The grouping of servers and clients is referred to as an NQE *cluster*. The servers provide reliable, unattended processing and management of the NQE cluster. Users who have long running requests and a need for reliability can submit batch requests to an NQE cluster.

NQE clients support the submission, monitoring, and control of work from the workstation for job execution of the batch requests on the nodes. The client interface has minimal overhead and administrative cost; for example, no machine ID (*mid*) administration is needed for a client machine. NQE clients are intended to run on every node in the NQE cluster where users need an interactive interface to the NQE cluster. The NQE client provides the NQE GUI, which is accessed through the `nqe` command. The NQE client also provides a command-line interface. For a list of user-level commands, see Appendix A, page 383. For information about using the NQE GUI and the command-line interface, see the *NQE User's Guide*, publication SG–2148.

The Network Queuing System (NQS) initiates requests on *NQS servers*. An NQS server is a host on which NQS runs. As system administrator, you designate the default NQS server in the NQE configuration file (`nqeinfo` file); a user may submit a request to the default NQS server or submit it to a specific NQS server by using the NQE GUI `Config` window or by setting the `NQS_SERVER` environment variable. Cray NQS provides unattended execution of shell script files (known as *batch requests*) in batch mode. Users can monitor and control the progress of a batch request through NQE components in the NQE cluster. When the request has completed execution, standard output and standard error files are returned to the user in the default location or specified alternate location. Privileged users defined as `qmgr` managers can configure, monitor, and control NQS; users defined as `qmgr` operators can control NQS queues and requests with the `qmgr` utility.

The NQE database provides a central repository for batch requests in the NQE cluster. When a request is submitted to the NQE database, it works with an administrator-defined NQE scheduler to analyze aspects of the request and to determine which NQS server will receive and process the request. When the scheduler has chosen a server for your request, the lightweight server (LWS) on the selected server obtains request information from the NQE database, verifies validation, submits the copy of a request to NQS, and obtains exit status of completed requests from NQS. By default, the copy of the request is submitted directly into a batch queue on the NQS server. Because the original request remains in the NQE database, if a problem occurs during execution and the server copy of the request is lost, a new copy can be resubmitted for processing if you have the clusterwide rerun feature enabled. For additional information about the NQE database and NQE scheduler, see Chapter 9, page 231, and Chapter 10, page 261.

The Network Load Balancer (NLB) provides status and control of work scheduling within the group of components in the NQE cluster. Sites can use the NLB to provide policy-based scheduling of work in the cluster. NLB *collectors* periodically collect data about the current workload on the machine where they run. The data from the collectors is sent to one or more NLB servers, which store the data and make it accessible to the NQE GUI `Status` and `Load` functions. The NQE GUI `Status` and `Load` functions display the status of all requests which are in the NQE cluster and machine load data.

Cray FTA allows reliable (asynchronous and synchronous) unattended file transfer across the network using the `ftp` protocol. Network peer-to-peer authorization allows users to transfer files without specifying passwords. Transfer may be queued so that they are retried if a network link fails. Queued transfer requests may be monitored and controlled.

The `csuspend`(8) utility lets you suspend and restart batch activity on a server when interactive use occurs.

The `NQE_DEFAULT_COMPLIST` variable in the `nqeinfo`(5) file contains the list of NQE components to be started or stopped (see Chapter 4, page 45). You can set this list to one or more of any of the following valid NQE components:

`NQS`	Network Queuing System
`NLB`	Network Load Balancer
`COLLECTOR`	NLB collector
`NQEDB`	NQE database
`MONITOR`	NQE database monitor
`SCHEDULER`	NQS scheduler
`LWS`	Lightweight server

Beginning with the NQE 3.3 release, the default component list consists of the following components: `NQS`, `NLB`, and `COLLECTOR`.

Below find a brief a description of the valid NQE components:

- The Network Load Balancer (NLB) server which receives and stores information from the NLB collectors in the NLB database which it manages.

  For more information on the NLB, see Section 2.9.1, page 29.

- The NQE database server which serves connections from clients, the scheduler, the monitor and lightweight server (LWS) components in the cluster to add, modify, or remove data from the NQE database. Currently, NQE uses the mSQL database.

  For more information on the NQE database server, see Section 2.9.2, page 31.

- The NQE scheduler which analyses data in the NQE database, making scheduling decisions.

  For more information on the NQE scheduler, see Section 2.9.2.3, page 33.

- The NQE database monitor which monitors the state of the database and which NQE database components are connected.

  For more information on the NQE database monitor, see Section 2.9.2.2, page 33.

- NQE *clients* (running on numerous machines) contain software so users may submit, monitor, and control requests by using either the NQE graphical user interface (GUI) or the command-line interface. From clients, users also may monitor request status, delete or signal requests, monitor machine load, and receive request output using the FTA.

The machines in your network where you run NQS are usually machines where there is a large execution capacity. Job requests may be submitted from components in an NQE cluster, but they will only be initiated on an NQS server node.

FTA can be used from any NQS server to transfer data to and from any node in the network by using the `ftpd` daemon. It also can provide file transfer by communicating with `ftad` daemons that incorporate network peer-to-peer authorization, which is a more secure method than `ftp`.

On NQS servers you need to run a collector process to gather information about the machine for load balancing and request status for the NQE GUI `Status` and `Load` windows programs. The collector forwards this data to the NLB server.

The NLB server runs on one or more NQE nodes in a cluster, but it is easiest to run it initially on the first node where you install NQE. Redundant NLB servers ensure greater availability of the NLB database if an NLB server is unreachable through the cluster.

Note: The NQE database must be on only one NQE node, there is no redundancy.

You can start the `csuspend`(8) utility on any NQS server to monitor interactive terminal session (tty) activity on that server. If tty activity equals or exceeds the input or output thresholds you set, NQE suspends batch activity. When interactive activity drops below the thresholds you specify, batch work is resumed.

Client nodes provide access to the following client commands: `nqe`(1) (which invokes the NQE GUI), `cevent`(1), `cqsub`(1), `cqstatl`(1), and `cqdel`(1). Client nodes also require access to the FTA `ftad` daemon (which services FTA requests issued by the requests and NQE itself). In a typical configuration, there would be many more client nodes than any other type of NQE node. Also, you can configure the client nodes to use the `ilb`(1) utility so that a user may execute a load-balanced interactive command; for more information, see Chapter 15, page 361.

In addition to the client commands, server nodes provide access to the following commands: `qalter`(1), `qchkpnt`(1), `qconfigchk`(1), `qdel`(1), `qlimit`(1), `qmgr`(1), `qmsg`(1), `qping`(1), `qstat`(1), and `qsub`(1).

For a complete list of commands, see Appendix A, page 383.

1.3 Running NQE As an Environment

The example given in this section shows how the NQE components work as an environment. For information about NQE user features and about setting NQE environment variables, see *Introducing NQE*, publication IN–2153, and the *NQE User's Guide*, publication SG–2148. Both documents, as well as this administration document, are available online (see *Introducing NQE*, publication IN–2153, for information about accessing NQE documentation online).

Figure 1 shows a possible NQE configuration. The user `mary` uses the client workstation `snow`, which has an NQE client interface to the NQS server `latte` (set by the environment variable `NQS_SERVER` to `latte`). `mary` wants the output from her batch request to go to her research assistant (`fred`) at another NQE client workstation, `gale`.

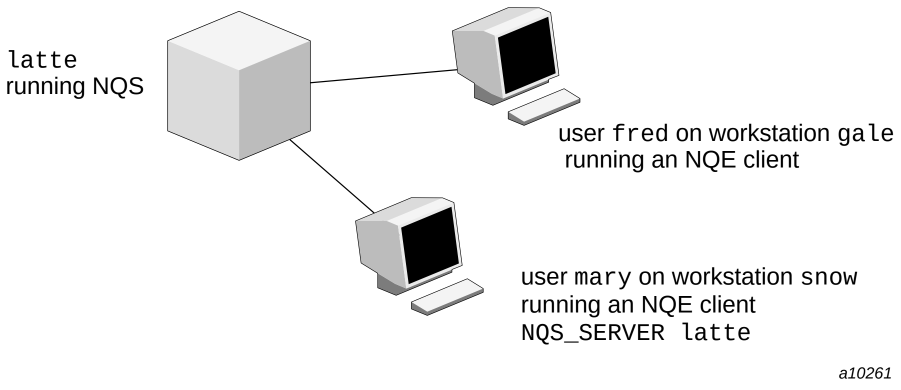


Figure 1. Sample NQE Configuration

User `mary` has several batch requests to run. One of the requests (named `jjob`) looks like the following example:

```
#QSUB -eo                #merge stdout and stderr
#QSUB -J m   #append NQS job log to stdout
#QSUB -o "%fred@gale/nppa_latte:/home/gale/fred/mary.jjob.output"
   #returns stdout to fred@gale
#QSUB -me   #sends mail to submitter at completion
#QSUB   #optional qsub delimiter
date   #prints date
rft -user mary -host snow -domain nppa_latte -nopassword -function get jan.data nqs.data
   #use FTA to transfer jan.data from latte to the NQS server (latte)
cc loop.c -o prog.out   #compile loop.c./prog.out
rm -f loop.c prog.out jan.data nqs.data   #delete files
echo job complete
```

The following embedded `qsub` option uses FTA to return the standard output file from the request to `fred` at the workstation `gale`; `nppa_latte` is the FTA domain:

```
#QSUB -o "%fred@gale/nppa_latte:/home/gale/fred/mary.jjob.output"
```

The request script uses FTA (the `rft` command) to transfer its files as shown in the following example:

```
rft -user mary -host snow -domain nppa_latte -nopassword -function get jan.data nqs.data
```

The FTA domain name `nppa_latte` and the option `-nopassword` indicate that peer-to-peer authorization is used, so `mary` does not need to specify a password; however, `mary` must have a `.netrc` file on `latte` to log into `fred`'s account and `mary` must also have an account on `gale` with permission to read `fred`'s file (see the `#QSUB` example, above).

User `mary` submits the request by using the following command line (alternatively, `mary` could submit the request by using the NQE GUI):

```
cqsub jjob
```

The request is sent to the NQS server `latte`, since `mary` 's environment variable `NQS_SERVER` is set to `latte`.

The load-balancing policy for `mary`'s site allows work to be shared among NQS systems on `latte`, `pendulum`, `telltale`, and `gevalia`. Because of the workload on the machines, `mary`'s first request is sent to `gevalia`, the second and third are sent to `latte`, the fourth and fifth are sent to `pendulum`, and the sixth is sent to `telltale`. User `mary` does not need to know where her requests are executing to find out their status. She can use the NQE GUI `Status` window to determine their status.

Because NQE GUI `Status` window is refreshed periodically, user `mary` can monitor the progress of all her requests. Because she used the embedded `#QSUB -me` option, she receives mail when each request completes.

For more information about FTA and `rft` syntax, about using `#QSUB` directives, or about using the NQE GUI `Status` window to determine status of requests, see the *NQE User's Guide*, publication SG–2148.

1.4 NQE File Structure

Throughout this guide, the path /*nqebase* is used in place of the default NQE path name, which is `/opt/craysoft/nqe` on UNICOS, UNICOS/mk, and Solaris systems and is `/usr/craysoft/nqe` on all other supported platforms.

Figure 2 shows the NQE file structure.

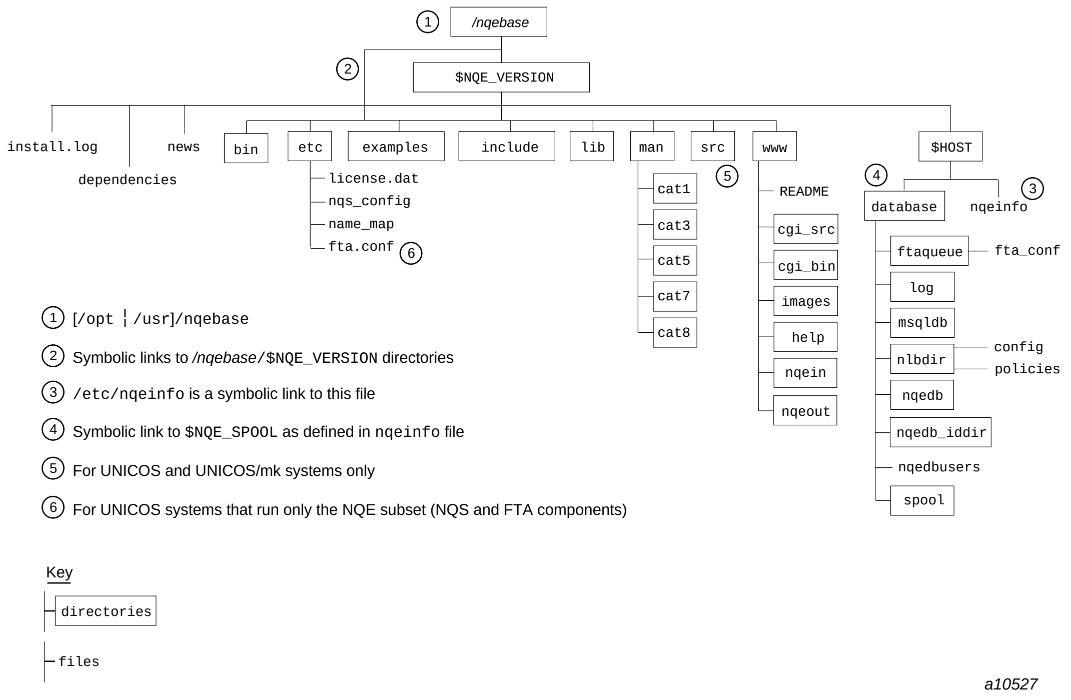


Figure 2. NQE File Structure

1.5 Using the World Wide Web Interface

The NQE release contains a World Wide Web (WWW) interface to NQE. You can access the interface through WWW clients such as Mosaic or Netscape. A single interface lets users submit requests (from a file or interactively), obtain status on their requests, delete requests, signal requests, view output, and save output. Online help is provided.

NQE administrators are encouraged to configure and customize this interface. It is provided so that administrators may supply users of nonsupported NQE platforms (such as personal computers) a tool that allows them to access NQE resources.

For information about managing the interface, read the /*nqebase*/`www/README` file. You can obtain the most current version of the NQE WWW interface at `ftp.cray.com` in the `/pub/nqe/www` file.

Note: The WWW interface is not available to UNICOS systems that run only the NQE subset (NQS and FTA components).

Concepts and Terms [2]

This chapter discusses concepts and terms important to NQE use and administration.

2.1 Batch Requests

Batch requests are shell scripts that are executed independently of any interactive terminal connection. The operating system treats this shell script as the standard input file (`stdin`). The only difference between this and a conventional shell script is that you can include NQS options (preceded by a special prefix) in the batch request file as comments before the first executable shell command.

When a request is submitted from the local or remote host, the standard output (`stdout`) and standard error (`stderr`) files are returned to the directory from which the request was submitted or returned to a user-specified alternative path. Optionally, users can request that `stdout`, `stderr`, and the job log file be merged (a *job log file* contains messages of NQS activity about a request).

Batch requests can have the attributes that are described in the following sections.

2.1.1 Nice Values

The *nice value* is the execution priority of the processes that form the batch request. A user can explicitly set the nice value (through the `Job Limits` selection of the `Configure` menu on the NQE GUI `Submit` window or as a `cqsub` or `qsub` command option) or NQS can set the nice value implicitly, based on the queue in which the request resides. Increasingly negative nice values increase the relative execution priority of a process.

Warning: Do not specify a negative nice increment for a queue because it may cause serious CPU scheduling problems, such as timeouts in system daemons or interference with kernel scheduling.

2.1.2 Shell Interpretation

A batch request can specify the full path name of a shell to interpret its shell script. If a user does not specify a shell, the user's login shell is used.

When NQS initiates a request, it spawns a second shell.

The second shell invoked is always /bin/sh unless another shell is specified by the user. The two-shell method allows stdin read-ahead by commands such as remsh and cat.

To define a shell for the first shell invoked, users can use the General Options selection of the Configure menu on the NQE GUI Submit window or use the cqsub -s or qsub -s command.

Users can define a shell other than /bin/sh to interpret their batch request by including the following line as the first line of the file (before any #QSUB directives):

#! *shell_path*

(such as /bin/csh)

You can include shell options on this line.

NQS sets the QSUB_SHELL environment variable to the shell the user is in when executing a request. NQS sets the SHELL environment variable to the initial shell executed by NQS.

The initial shell is not necessarily the same shell as the second shell. The second shell will always be /bin/sh, unless the user includes #! *shell_path* as the first line of the batch request.

The following example uses the cqsub command and shows how to verify the shells that an NQS request is using. User jjj uses csh as a login shell. The batch request file job1 contains only one line, as follows:

```
ps -ef | fgrep jjj
```

The request is submitted by using the following command:

```
cqsub -s /bin/csh job1
```

The following output shows that the request was executed under /bin/sh because jjj did not modify the first line of the file as required:

```
jjj 11874 11873 2 15:18:17 ?      0:00 fgrep jjj
jjj 11355 11354 0 14:54:37 pts/0    0:00 -csh
jjj 11873 11871 2 15:18:17 ?        0:00 /bin/sh
/nqebase/nqeversion/poe/database/spool/scripts/++++2++cd4X+++
jjj 11871 11870 22 15:18:16 ?        0:00 -csh
jjj 11875 11874 9 15:18:17 ?        0:00 ps -ef
```

To run the request under `csh`, you must modify the batch request file as follows:

```
#!/bin/csh
ps -ef | fgrep jjj
```

The `cqsub -s /bin/csh job1` command produces the following output:

```
jjj 11969 11967 19 15:23:06 ?  0:00 /bin/csh
/nqebase/nqeversion/pendulum/database/spool/scripts/++++3++cd4X+++
jjj 11355 11354 0 14:54:37 pts/0 0:00 -csh
jjj 11967 11966 5 15:23:05 ?   0:00 -csh
jjj 11974 11969 9 15:23:08 ?   0:00 ps -ef
```

If you want to use a one-shell invocation method, you can set the `NQSCHGINVOKE` environment variable to `true` or `yes`. The environment variable is set on a per-request basis; you must export environment variables by using the `General Options` selection of the `Configure` menu on the NQE GUI `Submit` window or by using the `qsub -x` option. If `NQSCHGINVOKE` is set to `true`, and you do not request a shell (by using the `General Options` selection of the `Configure` menu on the NQE GUI `Submit` window or by using `cqsub -s` or `qsub -s`), NQS invokes the request owner's UDB shell on UNICOS and UNICOS/mk systems or their login shell on UNIX systems.

To set a fixed shell path and invocation method, edit the `nqeinfo` file to define the following two variables:

```
NQE_NQS_SHELLPATH
NQE_NQS_SHELLINVOCATION
```

The default values are a null string for the shell path and 2 for the invocation method.

The allowed value for `NQE_NQS_SHELLPATH` is any valid shell path name. The default null string for `NQE_NQS_SHELLPATH` uses the user's login shell. If you specify a shell path name for `NQE_NQS_SHELLPATH`, the shell you specify is used for batch request processing.

The value specified by using the `General Options` selection of the `Configure` menu on the NQE GUI `Submit` window or specified by using the `cqsub -s` or `qsub -s` option, if used, overrides the system-level shell path configured in the `nqeinfo` file.

The allowed value for NQE_NQS_SHELLINVOCATION is 1 or 2. A value of 1 for NQE_NQS_SHELLINVOCATION uses a one-shell invocation method for all NQS requests. A value of 2 uses the default two-shell invocation method.

The value of any NQSCHGINVOKE environment variable received with a request overrides the system-level invocation method configured in the nqeinfo file.

During NQS startup, the values of these two variables are verified. If a stat() system call of a non-null shell path name fails, or if the value of the invocation method is not either 1 or 2, the nqsdaemon aborts its startup. Messages describing the problem are written to the NQS console file and/or the log file.

2.1.3 Alternate User Validation

Note: Alternate user validation applies only for NQS requests submitted using the cqsub command.

By default, an NQE job is submitted for execution under the user name executing the cqsub command. By using the cqsub -u command, or by setting the User Name field in the NQE GUI Submit display, a user can submit a request for execution under a different user name.

The user account on the client workstation is referred to as the originating client user account, and the user account on the NQS server is referred to as the target user account.

The nqeinfo file variable NQE_NQS_NQCACCT controls which user account is validated on the NQS_SERVER when a user submits a job for execution under a different user name. The value of this variable on the NQS_SERVER host determines which user name is used for validation on the NQS_SERVER.

The value ORIGIN means that the originating client user account is validated on the NQS_SERVER. This means that the client user must have an account on the NQS_SERVER with the same name (but not necessarily the same UID) as the originating client user account.

The value TARGET means that the target user account is validated on the NQS_SERVER. It is not necessary for the client user to have an account on the NQS_SERVER with the same name as the client user account.

If the NQE_NQS_NQCACCT variable is not set on the NQS_SERVER, it defaults to the value ORIGIN.

If NQE_NQS_NQCACCT is set to ORIGIN, and password validation is set on the NQS_SERVER, then both the client user account on the NQS_SERVER and the target user account on the NQS server must have the same password or the password validation will fail.

2.1.4 Intraqueue Priority

The *intraqueue priority* determines the order in which requests are initiated. After a request is accepted into a batch queue, the NQS scheduler calculates the intraqueue priority each time it scans the requests in the queue. The order in which queues are considered does not change; only the order in which requests are selected from a queue changes. The eligible request with the highest intraqueue priority is selected next for initiation. (*Eligible* means the request meets all quotas such as run limit, user limit, and group limit). For additional information, see Section 5.12, page 82.

2.1.5 Resource Limits

A request or the individual processes of a request can impose a set of limits on the use of a resource. Typical limits are maximum file size, CPU time, and maximum memory size. A user can explicitly set up resource limits by using NQE GUI options or `cqsub` command options, or NQS can set the resource limits implicitly, based on the batch queue in which the request executes.

The limits supported by NQS are described in Section 5.11, page 75.

2.2 Request States

The state of a request can be found by using the NQE GUI `Status` window or the `cqstatl -f` or `qstat -f` command. The state is displayed after `Status:` or at the top right of the detailed request display. (To access a detailed request display through the NQE GUI `Status` window, double-click on a request line.)

An abbreviated form of the status also is displayed on the NQE GUI `Status` window under the `Job Status` column or under the `ST` column of the request summary display of the `cqstatl -a` or `qstat -a` command.

The current state of the request shown on the `cqstatl` or `qstat` display can be composed of a major and a minor status value. Major status values are as follows:

Code	Description
A	Arriving

C	Checkpointed
D	Departing
E	Exiting
H	Held
N	NQE Database request
Q	Queued
R	Routing (pipe queues only)
R	Running (batch queues only)
S	Suspended
U	Unknown state
W	Waiting for a date and/or time specified by the `-a` option to `cqsub`, waiting for a license, or waiting for a network connection

Minor status values are described on the `cqstatl`(1) and `qstat`(1) man pages.

A batch request submitted to the NQE database typically progresses through the following states; the first four characters of a state are displayed in the `ST` column of a database request summary (shown by using the `cqstatl -a` or `qstat -a` command):

State	Description
`New`	The request is in the NQE database.
`Pending`	The request is in the NQE database awaiting scheduling.
`Scheduled`	The request is in the NQE database and has been scheduled by the NQE scheduler.
`Submitted`	A copy of the request has been submitted from the NQE database for processing.
`Completed`	The copy of the request submitted from the NQE database for processing has completed processing.
`Terminated`	The copy of the request submitted from the NQE database for processing has terminated.

A batch request submitted to a local pipe queue for routing to a batch queue (defined in the next section) typically progresses through the following states at a local pipe queue:

State	Description
`queued`	Awaiting routing to another queue
`routing`	Being moved to another queue

A request progresses through the following states in a batch queue:

State	Description
`arriving`	Being moved from another queue
`queued`	Awaiting processing
`running`	Being processed
`waiting`	Waiting for a specified execution time
`exiting`	Processing complete
`departing`	Left the queue but not yet deleted

An executing request can also have the following states:

State	Description
`held`	Held by NQS operator action; resources are not released.
`suspended`	Processing temporarily suspended by an operator; resources are released.

The ordering of requests within a queue does not always determine the order in which the request is processed; the NQS request scheduler determines the processing order, depending on the various limits that the system administrator or operator imposes.

2.3 Queues

A *queue* is a set of batch requests. NQS assigns requests to a particular queue, where they wait until they are selected for execution. NQS also assigns an execution priority to each request. NQS executes each request according to its priority and routes any output to the specified destination. The following sections describe the two types of queues: *batch* and *pipe*.

NQE is shipped with one pipe queue and one batch queue configured on each NQS server. The default pipe queue is `nqenlb` . The default batch queue is nqebatch. This configuration uses the `nqenlb` queue to send requests to the NLB, which then sends requests to `nqebatch` on the most appropriate system, based on an NLB policy. The default NLB policy, called `nqs`, sends batch requests to the system with the most available CPU cycles. If requests are submitted to the queue `nqebatch`, the request runs on the local NQS server.

2.3.1 Batch Queues

Batch queues accept and process batch requests. Each batch queue has an associated set of request and process resource limits. If a request is to be accepted in a particular batch queue, no resource limit specified by the request may exceed the corresponding limit specified by the target batch queue. If a batch request fails to specify a resource limit that is enforced, NQS sets the limit for the request to the corresponding default limit of the queue. The `qmgr` commands define the queue default values.

Note: For UNICOS and UNICOS/mk systems, NQS assigns the limit for the request to either the default limit of the queue or the user database (UDB) limit, whichever is more restrictive.

If the `qmgr` manager lowers the request and process resource of a queue, all requests that are already in the queue with quotas that exceed the new limits remain in the queue through a "grandfather" clause.

Every batch queue also has a set of associated queue limits, which are described in Section 2.5, page 24.

2.3.2 Pipe Queues

Pipe queues handle the routing and delivery of requests from one queue to another. They serve as pipelines, transporting requests to other local and remote queue destinations.

Pipe queues do not have associated quota limits (although they can have attributes). Pipe queues have a set of associated queue destinations, on both local and remote machines, to which they route requests. These destinations can be batch queues or other pipe queues.

NLB pipe queues used for load balancing do not have destinations associated with them.

Each request in a pipe queue is routed to one of the queue’s destinations. Destinations are considered in the order in which the administrator configures them. If a request cannot be accepted by a destination queue (for example, if its resource requirements exceed those allowed for membership in the queue), the next destination is considered. If no destination accepts the request, the request is deleted, and the user receives a mail message.

If a destination is inaccessible (for example, if the network connection is broken or if the `loadonly` limit for a queue has been reached), NQS tries to requeue the request at specified intervals (defined by the `qmgr` command `set default destination_retry wait`) for a specified length of time (defined by the `qmgr` command `set default destination_retry time`). If these attempts fail, NQS abandons the attempted request and the user receives a mail message.

Each pipe queue has an associated program spawned to handle every request initiated from the queue for routing and delivery. In the context of the client/server network connection, this program is referred to as the pipeclient process.

2.3.3 Queue, Queue Complex, and Global Limits

Queue, queue complex, and global limits (defined beginning with Section 2.5, page 24) limit the number of requests that can run concurrently. You can use `qmgr` commands to set a finite value that limits the maximum number of requests that can run at one time. Most limits can be set to `unlimited`. When a queue is created, these limits are defined as `unspecified`; `unspecified` means that the queue has the default value for that limit.

You do not have to set every limit. A comprehensive set of limits is provided so that you can configure the workload to meet your site's needs.

2.4 Request Processing

This section provides summary information on the request submission process, controlling requests, and monitoring requests.

2.4.1 Request Destination

By default, requests are sent to NQS. You may change the default and have requests sent to the NQE database, or your users may select the destination of their individual requests. For additional information about configuring the NQE database, see Chapter 9, page 231.

2.4.2 NQS Server Local Submission

When a user submits a batch request to NQS, the request is sent to `nqsdaemon`. The `nqsdaemon` checks the request qualifiers (if there are any) against the defined queue limits and attributes. Based on these checks, `nqsdaemon` accepts or rejects the request. If the queue is a pipe queue, when `nqsdaemon` determines that a request should be routed, it spawns the `pipeclient` program to route the request from the pipe queue to another pipe or batch queue. Figure 3 shows this process.

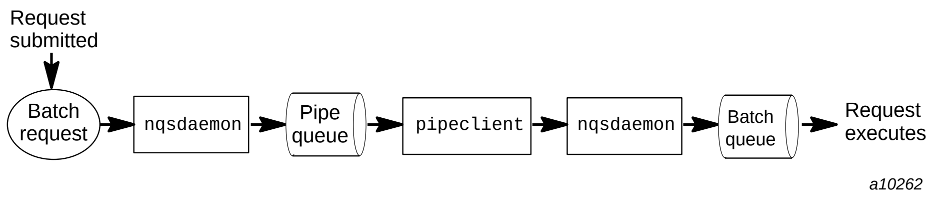


Figure 3. Request Processing for Local Submission

Note: The concept of local submission does not apply for requests submitted to the NQE database.

2.4.3 NQE Client Submission

A user can submit a request to NQS or to the NQE database. The following sections describe the process of submitting a request to each destination.

2.4.3.1 Client Submission to NQS

When a user submits a request to NQS using the NQE GUI `Submit` window or the NQE `cqsub` command, no queues are involved on the local host. The request is sent to the `netdaemon` process on the server machine as specified by the `NQS_SERVER` variable. The `netdaemon` creates a child process (`netserver`) to handle the request. The child process tries to queue the request with `nqsdaemon`. `netserver` ensures that the user has an account on the target host and that the user name on the local machine is authorized to run requests on the target machine.

The target `nqsdaemon` then checks the request qualifiers against the defined queue limits and attributes and, based on these checks, accepts or rejects the request. Figure 4 shows this process.

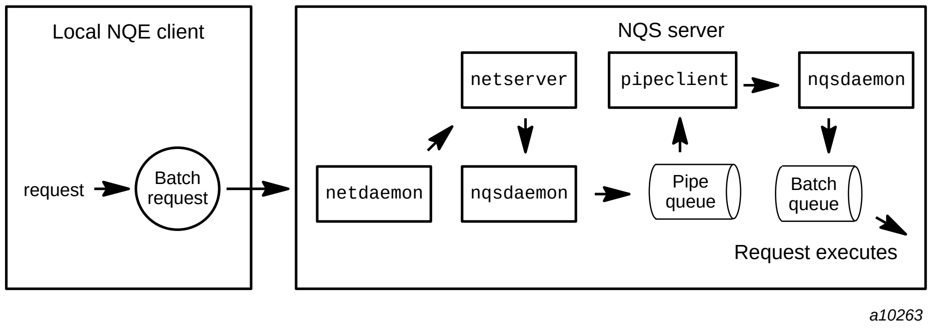


Figure 4. Request Processing for Client Submission to NQS

2.4.3.2 Client Submission to NQE Database

When a user submits a request to the NQE database using the NQE GUI `Submit` window or the NQE `cqsub` command, the NQE scheduler determines when and where the request will execute. The NQE database routes a copy of the request to the lightweight server (LWS) on the node running the NQS server. The LWS verifies validation, sends a copy of the request to the NQS batch queue for processing, and obtains exit status of completed requests.

Figure 5 shows request processing for client submission to the NQE database.

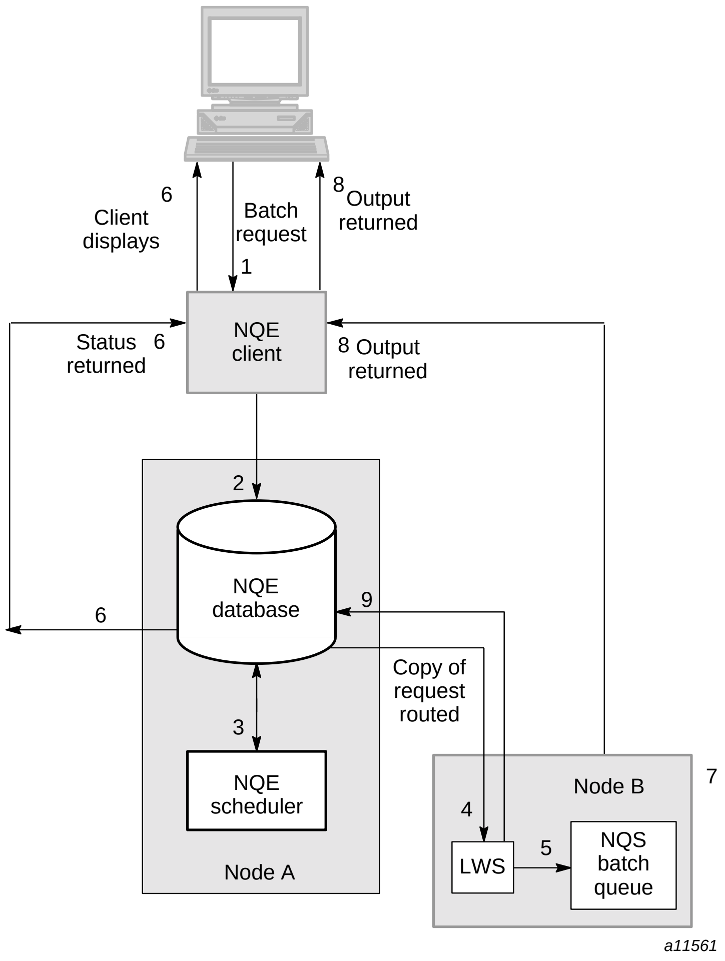


Figure 5. Request Processing for Client Submission to the NQE Database

2.4.4 NQS Server Remote Submission

If the request is sent from one NQS system to a remote NQS system, `pipeclient` sends the request to the remote `netdaemon`. The rest of the processing is the same as it is for client submission to NQS. Figure 6 shows this process.

Note: The concept of remote submission does not apply for requests submitted to the NQE database.

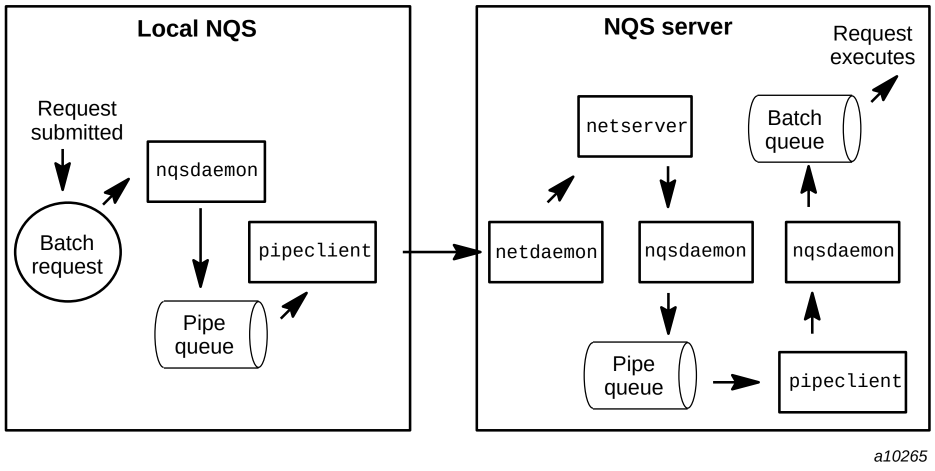


Figure 6. Request Processing for Remote Submission

2.4.5 Daemon Processing

When `nqsdaemon` determines that a request is eligible to run, it spawns a shepherd process. The shepherd process sets up the environment for the request and runs it. It sends mail to the user if it cannot execute the request. The shepherd process waits for the request to complete and returns output file(s) to the specified or default file destination.

2.4.6 Controlling and Deleting Requests

NQS users can control or delete their own requests by using the NQE GUI `Status` window `Actions` menu options or by using the `cqdel` or the `qdel` command. NQS operators and managers can control requests with the following `qmgr` commands (see Section 6.6, page 162, for a description of these commands and examples of their use):

- `abort request`
- `delete request`
- `hold request`
- `modify request`
- `move request`

- `release request`
- `rerun request`
- `resume request`
- `schedule request` (`first`, `next`, `now`, `system`)
- `suspend request`

Managers and operators can also delete requests from any user by using the `cqdel` or `qdel` command.

2.4.7 Displaying Queue Status

The `cqstatl` command, the `qstat` command, and the `qmgr` commands `show long queues` and `show queues` display the current status of NQS queues.

2.5 Queue Attributes

Queue attributes define the properties of a queue. A queue attribute can be assigned by using the `qmgr set attribute` command. For a description of how to set attributes, use the `qmgr help set attribute` command.

The following sections describe possible attributes for NQS queues. For additional information about defining queue properties, see Section 5.6, page 58.

2.5.1 Batch and Pipe Queue Attributes

The queue attributes described in the following sections apply to both batch and pipe queues.

2.5.1.1 Access Restrictions

Queue access can be either restricted or unrestricted. If access is restricted, only requests submitted by users defined in the access set for the queue are accepted. If access is unrestricted, any request can enter the queue.

The access set is composed of individual users or groups and is defined by the system administrator.

2.5.1.2 `pipeonly`

A batch queue or a pipe queue can be declared `pipeonly`, meaning that it can accept requests only from a pipe queue. This declaration is useful when the `pipeclient` process selects the final queue, based on the network queue configuration. For example, if all requests are submitted to a pipe queue, the pipe queue can retry submissions if no batch queues are currently accepting requests. Another example might be that you want to configure your queues such that requests of a certain size go to certain queues; if all requests are submitted to pipe queues, the pipe queue can route the requests to the proper queues without users having to know which queue to use.

2.5.1.3 Interqueue and Intraqueue Priority

The interqueue priority dictates the order in which queues of each type (batch or pipe) are scanned in a search for the next request to process. The queue with the highest priority is scanned first.

The difference between *interqueue* and *intraqueue* priority is important. The interqueue priority determines the order in which queues are searched for requests to process; the intraqueue priority determines the order in which requests within a queue are considered for initiation when the queue is searched.

2.5.1.4 Queue Run Limit

The queue run limit is the maximum number of requests allowed to be executing from the queue at any one time.

2.5.2 Batch Queue Attributes

The queue attributes described in the following sections apply only to batch queues.

2.5.2.1 `loadonly`

You may not submit requests directly to a `loadonly` queue. Requests may come only from a pipe queue. In other words, a `loadonly` queue is a `pipeonly` queue that has the additional restriction of being able to accept only a limited number of requests.

A batch queue can be declared `loadonly`, meaning that it can accept only a limited number of requests. The number of requests that can be queued is restricted to the run limit of that queue. Therefore, if no other limits exist, a `loadonly` queue accepts only the number of requests that it can run. Specifically, the algorithm is the *number of requests queued + number of requests running + number of requests waiting + number of requests arriving <= run limit*.

This algorithm can be used for load sharing between multiple machines or across multiple queues on a single machine. In the example in Figure 7, requests are submitted to the pipe queues `batch@hot` and `batch@ice`; they are routed from these queues to the central pipe queue, which queries all three batch queues on both `hot` and `ice`. The central pipe queue would try to route a small request to `smalljobs@hot` first, next try `smalljobs@ice`, and continue down the list until a place in a queue became available. Without using `loadonly` queues, the queue `smalljobs@hot` may accept the request, and the request may remain queued.

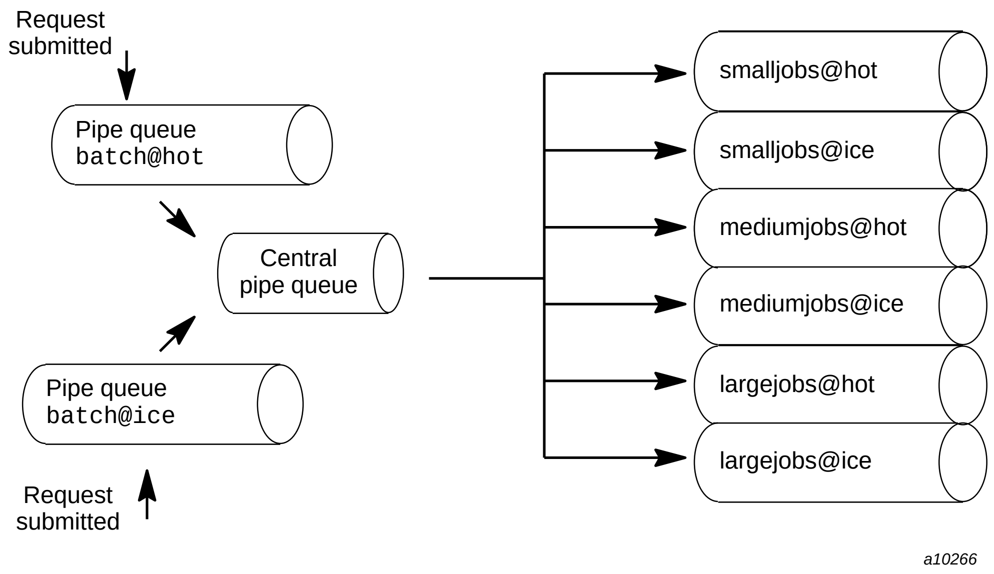


Figure 7. Queue Configuration That Uses the`loadonly` Attribute

2.5.2.2 Queue Group Limit

The queue group limit associated with each batch queue is the maximum number of requests allowed to run in the queue at any one time that were submitted by all users who are members of one group.

2.5.2.3 Queue Memory Limit

The queue memory limit associated with a batch queue is the maximum amount of memory that can be requested by all running requests in the queue at one time.

2.5.2.4 Queue User Limit

The queue user limit associated with a batch queue is the maximum number of requests allowed to be run in the queue at any time by one user.

2.5.2.5 Queue MPP Processing Element (PE) Limit

For Cray MPP systems, each batch queue has an associated MPP PE limit. This is the maximum number of MPP PEs that can be requested by all running jobs in the queue at any one time.

2.5.2.6 Queue Quick-file Limit

For UNICOS systems with SSD solid-state storage devices, each batch queue has an associated quick-file limit. This is the maximum size of secondary data segments that can be allocated to all batch requests running concurrently in the queue.

2.6 Queue States

At any time, the state of a queue is defined by two properties:

- The queue's ability to accept requests. The states associated with this property are as follows:

State	Description
`CLOSED`	The NQS daemon (`nqsdaemon`) is not running at the local host.
`DISABLED`	The queue is not currently accepting requests.
`ENABLED`	The queue is currently accepting requests.

- The queue's ability to route (pipe queue) or initiate (batch queue) requests. The states associated with this property are as follows:

State	Description
`INACTIVE`	Queued requests are allowed to run, although none are currently running.
`RUNNING`	Queued requests are allowed to run, and some are currently running.
`SHUTDOWN`	NQS is not running on the host in which the queue resides.
`STOPPED`	Queued requests are not allowed to run, and none are currently running.
`STOPPING`	New queued requests are not allowed to run, although currently executing requests are running to completion.

2.7 Queue Complexes

A *queue complex* is a set of local batch queues. Each complex has a set of associated attributes, which provide for control of the total number of concurrently running requests in member queues. This, in turn, provides a level of control between queue limits and global limits (see Section 2.5, page 24, for information about queue attributes and Section 2.8, page 29, for information about global limits). The following queue complex limits can be set:

- Group limits
- Memory limits
- Run limits
- User limits
- MPP processing element (PE) limits (CRAY T3D systems), or MPP application processing elements (CRAY T3E systems, or number of processors (IRIX systems)
- Quick-file limits (UNICOS systems with SSDs only)

Several queue complexes can be created at a given host, and a batch queue can be a member of several complexes.

Note: A batch request is considered for running only when all limits of all complexes of which the request is a member have been met.

Section 5.9, page 71, describes how to set queue complex limits.

2.8 Global Limits

Global limits restrict the total workload executing concurrently under NQS control. While queue limits restrict requests in queues and complex limits restrict requests in a complex, global limits restrict the activity in the entire NQS system. The following global limits may be set:

- Batch limits
- Group limits
- Memory limits
- Pipe limits
- Tape drive limits
- User limits
- MPP processing element (PE) limits (CRAY T3D systems), or MPP application processing elements (CRAY T3E systems, or number of processors (IRIX systems)
- Quick-file limits (UNICOS systems with SSDs only)

Section 5.10, page 73, describes how to set global limits.

2.9 Load Balancing and Destination Selection

Load balancing is the process of allocating work (such as NQS batch requests) in order to spread the work more evenly among the available hosts in a group of NQE nodes in the NQE cluster. Load balancing can minimize instances in which one machine has idle time while another remains saturated with work.

Load balancing can be done either by using the Cray Network Load Balancer (NLB) or by using the NQE database and its associated scheduler. The two methods are described in the following sections.

2.9.1 NLB

The following sections describe the concepts used in the Cray Network Load Balancer (NLB).

2.9.1.1 NLB Servers

The *NLB server* provides a generic, network-accessible data storage mechanism that can be used for a variety of purposes, including load balancing, NQS request status, and network configuration. Because the data can be used for various purposes, the server holds data as generic network data objects; it has no knowledge of the meaning of the data it stores. The programs that load and interrogate the database, known as collectors and clients, define the meaning of the data in the server.

NLB servers can be replicated to ensure that if connection to one is lost, a duplicate can be reached by collectors and clients. The server is a passive process; it does not actively solicit data from other processes. The NLB collectors periodically generate new information and send it to all configured NLB servers. The various NLB clients query NLB servers one by one in the configured order until they find one that is running or until they exhaust the list.

Some object types require access control to prevent unauthorized entry and extraction of data. Such objects are controlled with access control lists (ACLs) used in conjunction with authentication data sent with each message to the NLB server. ACLs are also groups of objects and can be edited and viewed using `nlbconfig`. The master ACL controls access to all other ACLs; it is defined in the NLB configuration file.

The server associates an ACL with each group of objects it maintains. Records in the ACL consist of a user name, host name, and a set of privileges (`read`, `update`, or `delete`). There are two special ACL user names: `OWNER` and `WORLD`. If an object has an `owner` attribute associated with it, the `OWNER` ACL controls the particular user’s permissions. An ACL record with the same name and host as the user making a request controls that user’s access rights, and the `WORLD` record controls all other users.

The read privilege lets users extract an object from the server; the `update` privilege lets users add new objects of that type or modify existing objects; and the `delete` privilege lets users remove objects from the server.

When an object is sent to the server by an authenticated collector, a special `OWNER` attribute may be present. This attribute is used to check permissions to read the object on subsequent queries.

2.9.1.2 Destination Selection

Destination selection is the process of determining the best host to which to send work. A destination selection policy (also called a *load-balancing policy*) is the mechanism whereby the NLB server determines which host to recommend as the target for work (such as a batch request). When a batch request is submitted to a load-balanced queue within NQS, the NLB server applies a policy to the current information it has on hosts in the group of NQE nodes in the NQE cluster. It then provides NQS with a list of machines, ordered according to the specifications of the policy.

An NQS pipe queue can be configured to obtain its destinations by applying a policy rather than having the destinations defined with `qmgr` commands. If you define an NLB pipe queue, you do not define any destinations for it, because the NLB selects destinations for you.

2.9.1.3 Policies

A *load-balancing policy* is a set of equations used to select hosts and sort them into an order based on the data held in the NLB.

2.9.1.4 NLB Collectors

NLB collectors periodically collect data about the machines on which they are running and forward this data to the NLB server. Some of the data on machine load and batch request status is gathered dynamically from the operating system. Other data can be read from a file by the collector or written directly into the NLB server using the `nlbconfig` program. These collectors reside in the program `ccollect`. (The `ccollect` program will reflect the port number.) For more information about NLB collectors, see Chapter 7, page 173.

2.9.1.5 NLB Clients

NLB clients query servers for information to use for destination selection or for use in displays. Examples of NLB clients are the NQS `pipeclient` and the NQE graphical user interface (GUI).

2.9.2 NQE Database

The NQE database provides the following:

- A mechanism for the client user to submit requests to a central storage database.
- A scheduler that analyses the requests in the NQE database and chooses when and where a request will run.

- The ability for a request to be submitted to the selected NQS server once the NQE scheduler has selected it.

Figure 8 shows the basic concept of the NQE database in the NQE:

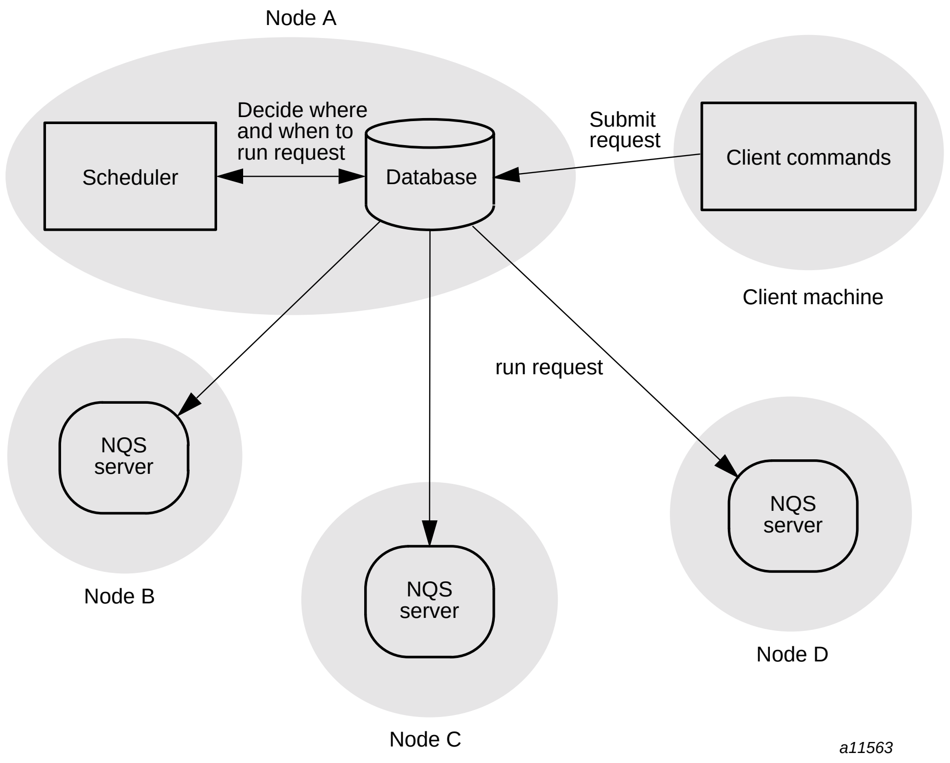


Figure 8. Basic Concept of the NQE Database

The components of the NQE database are described in the following sections.

2.9.2.1 NQE Database Server (MSQL)

The NQE database server (mSQL) serves connections from clients in the network or locally to access data in the database. The database used is mSQL, and it has the following features:

- Simple, single-threaded network database server

- Relational database structure with queries issued in a simple form of SQL

2.9.2.2 NQE Database Monitor

The monitor is responsible for monitoring the state of the database and which NQE database components are connected. Specifically, it determines which component processes are connected and alive and, optionally, it purges old requests (called tasks when they are within the database) from the database.

2.9.2.3 Scheduler

The scheduler analyses data in the database, making scheduling decisions. Specifically, it does the following:

- Tracks the lightweight servers (LWSs) currently available for use in destination selection.
- Receives new requests, verifies them for validity, and accepts or rejects them.
- Executes a regular scheduling pass to find NQS servers to run pending requests.

Administrator-defined functions may be written in Tcl to perform site-specific scheduling (see Chapter 10, page 261).

2.9.2.4 Lightweight Server (LWS) (Bridge to NQS)

The LWS performs the following tasks:

- Obtains request information from the database for requests assigned to it by the scheduler.
- Checks authorization files locally.
- Changes context and submits requests to NQS.
- Obtains the exit status of completed requests from NQS and updates the NQE database with this information.

2.10 File Transfer

The following sections describe terms commonly used with the File Transfer Agent (FTA). FTA provides file transfer between remote systems on a network.

2.10.1 File Transfer Services

A *file transfer service* is a program that provides FTA with access to a network system that uses a specific file transfer protocol.

2.10.2 Domain Names

A unique FTA *domain_name* is assigned to each of the file transfer services that are available to FTA. A domain name is subsequently used to identify a specific transfer service.

2.10.3 FTA Users

FTA defines specific types of users, as follows:

- *Admin users*: A user or a member of a group who is listed in the FTA configuration file as an administrator.
- *Status users*: A user or member of a group who is listed in the FTA configuration file as a status user.

2.10.4 Network Peer-to-peer Authorization

Network peer-to-peer authorization (NPPA) lets users transfer files without sending a password across the network. It requires FTA on the local system and support for network peer-to-peer authorization on the remote system. It can be used to authorize both batch and interactive file transfers.

2.10.5 File Transfer Requests

A *file transfer request* is an object containing information that describes the file transfer operations to be performed by FTA on behalf of a user. Each request is a separate file, which is located in the FTA queue directory.

2.10.6 Queue Directories

A *queue directory* is a file system directory that contains file transfer request files.

Configuring NQE variables [3]

The NQE configuration utility (`nqeconfig`(8)) assists administrators in creating and maintaining the NQE configuration file (`nqeinfo` file). The NQE configuration file contains the NQE configuration variables and is created by the NQE installation procedure. All configured NQE server nodes and clients must each have a unique NQE configuration file.

When you use the NQE configuration utility, each NQE configuration variable is displayed along with its value. Most variables are configurable. However, some variables are derived from the values of other variables and may not be changed. This helps to ensure that changes are propagated throughout the file and reduces the risk of misconfiguration due to conflicting values of related variables.

Note: Some variables may be set in the user's environment to indicate individual preferences. The NQE configuration file variables are used only if an individual has not indicated a preference.

This chapter describes how to start and use the NQE configuration utility. For a complete list of all NQE configuration variables, see the `nqeinfo`(5) man page or the NQE configuration utility `Help` facility.

Note: For UNICOS 9.0.*x* and 9.1 systems upgrading to this NQE release, Appendix C, page 395, includes a list of NQE configuration file (`nqeinfo` file) names that were previously included in the `config.h` file.

3.1 Starting the NQE Configuration Utility

To start the NQE configuration utility, use the `nqeconfig`(8) command. The `nqeconfig`(8) command has the following syntax:

```
nqeconfig [-a] [-f filename][-o filename]
   [-D variable=value[,variable=value,...]]
```

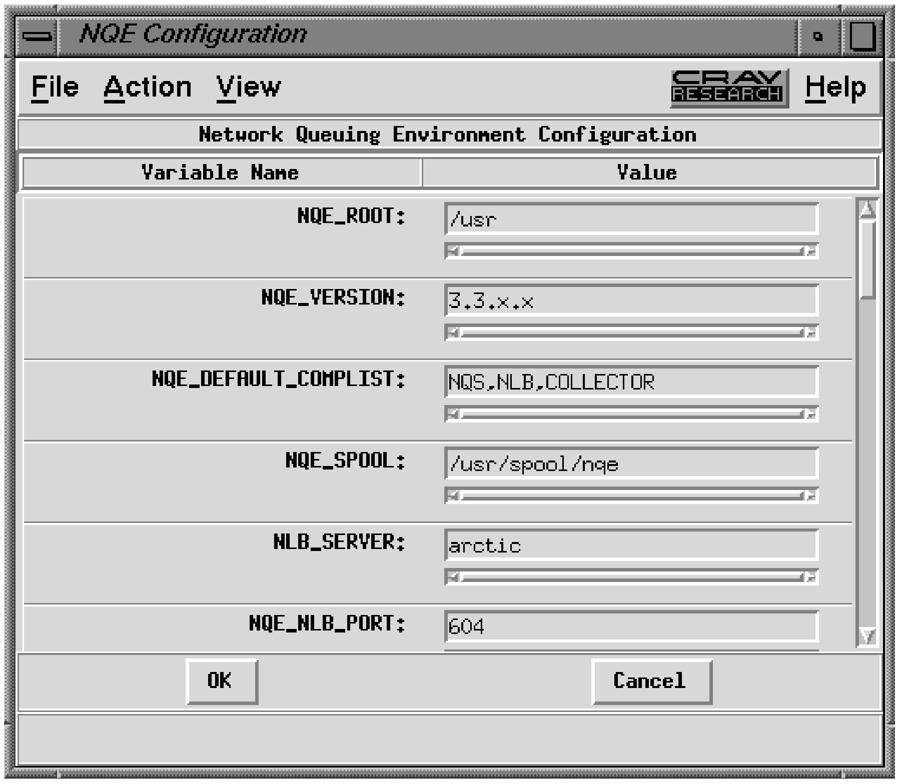
The `nqeconfig`(8) command accepts the following options:

Option	Description
`-a`	Instructs `nqeconfig`(8) to run in *autopilot mode.* No prompting occurs when this command is run

	in this mode. The configuration file is generated using the values from the file indicated by the `NQEINFOFILE` environment variable (or from the `/etc/nqeinfo` file if the `NQEINFOFILE` variable is not set), from the values in the input file specified on the command line with the `-f` option, and from other values specified with the `-D` option.
`-f` *filename*	Indicates the input file used to specify additional configuration variables and new values for existing variables. The format of this file is the same as that of the `nqeinfo` file.
`-o` *filename*	Specifies the file location for the new configuration file. This location may be overridden in the `File` menu of the `nqeconfig`(8) display.
`-D` *variable=value* [*,variable=value*,...]	Allows you to specify, on the command line, additional configuration variables and new values for existing variables. The argument to this variable is a comma-separated list of *variable=value* pairs. Multiple `-D` options may be specified.

3.2 Editing the NQE Configuration Variables

By default, after you execute the `nqeconfig`(8) command, the following `NQE Configuration` display appears, as shown in Figure 9:



Variable Name	Value
NQE_ROOT:	/usr
NQE_VERSION:	3.3.x.x
NQE_DEFAULT_COMPLIST:	NQS,NLB,COLLECTOR
NQE_SPOOL:	/usr/spool/nqe
NLB_SERVER:	arctic
NQE_NLB_PORT:	604

Figure 9. `NQE Configuration` Display, `Condensed` View

Caution: Always stop NQE before making changes to the NQE configuration. Each NQE command and server program reads the configuration file once during command or server initialization. Changes made to the configuration file while NQE is running can result in NQE commands and servers using different configuration file values, which will result in unpredictable problems.

The `NQE Configuration` display presents the NQE configuration variables and their associated values. Values with a dark background are derived from the values of other variables and may not be edited. Other values may be edited either by entering the appropriate text or by choosing a value from a set of options. The scrollbar allows you to scroll through the list.

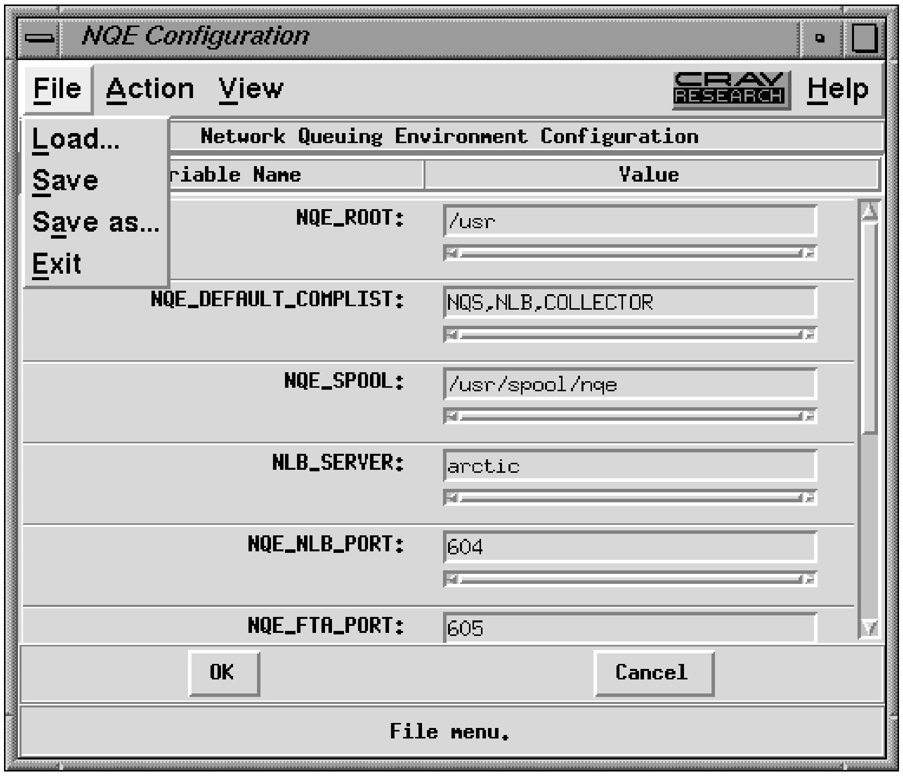
When you have finished making configuration changes, write the configuration changes to the new configuration file by clicking on the `OK` button, or you can cancel the changes by clicking on the `Cancel` button.

Context-sensitive help is available at the bottom of the display.

3.3 `NQE Configuration` Display Menus

The following menus provide additional functionality:

Menu	Description
`File` menu	Contains options that allow you to save your changes to the default file location, to the location specified with the `-o` option on the command line, to specify another file and save the changes, or to exit the utility without saving changes. The `File` menu is as follows:



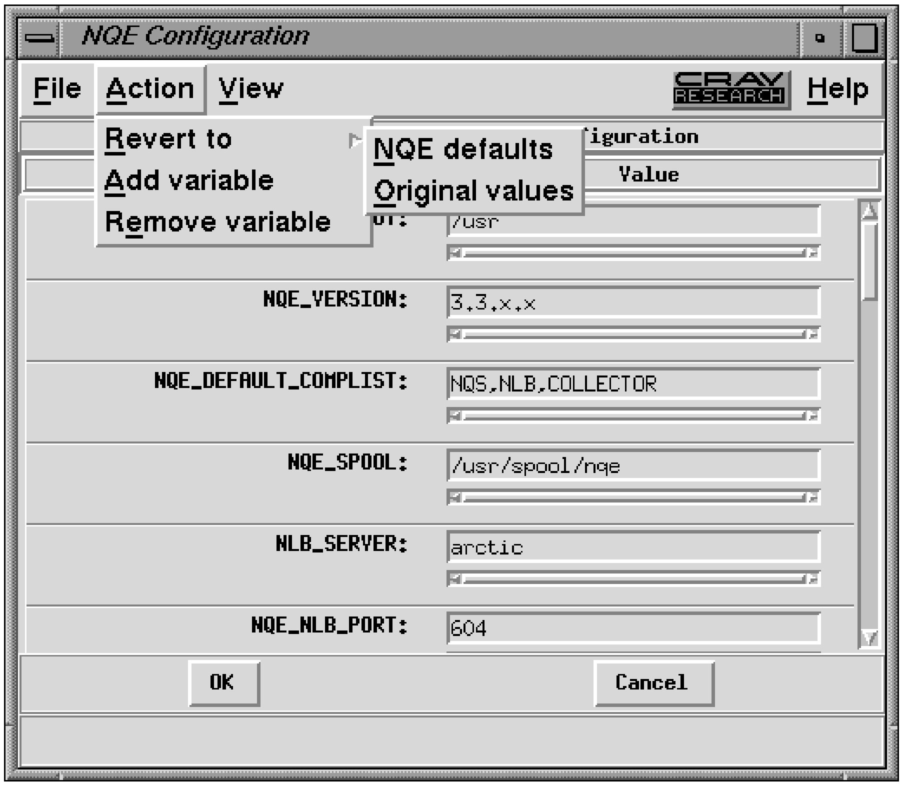
Network Queuing Environment Configuration

riable Name	Value
NQE_ROOT:	/usr
NQE_DEFAULT_COMPLIST:	NQS,NLB,COLLECTOR
NQE_SPOOL:	/usr/spool/nqe
NLB_SERVER:	arctic
NQE_NLB_PORT:	604
NQE_FTA_PORT:	605

File menu.

Figure 10. `NQE Configuration` Display, `File` Menu

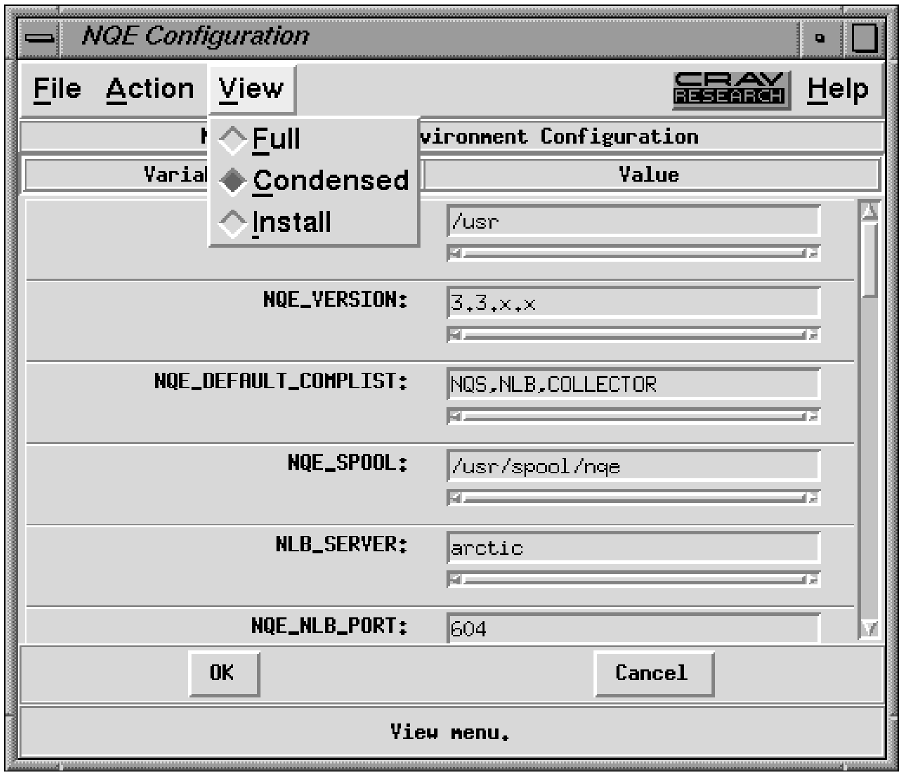
`Action` menu	Allows you to revert to the NQE default configuration values, to revert to the values in use when the utility started, or to add variables to or remove variables from the configuration. The `Action` menu is as follows:



NQE_VERSION: 3.3.x.x
NQE_DEFAULT_COMPLIST: NQS,NLB,COLLECTOR
NQE_SPOOL: /usr/spool/nqe
NLB_SERVER: arctic
NQE_NLB_PORT: 604

Figure 11. `NQE Configuration` Display, `Action` Menu

`View` menu	Allows you to select one of three views: `Condensed`, `Full`, or `Install`. The `View` menu is as follows:



vironment Configuration

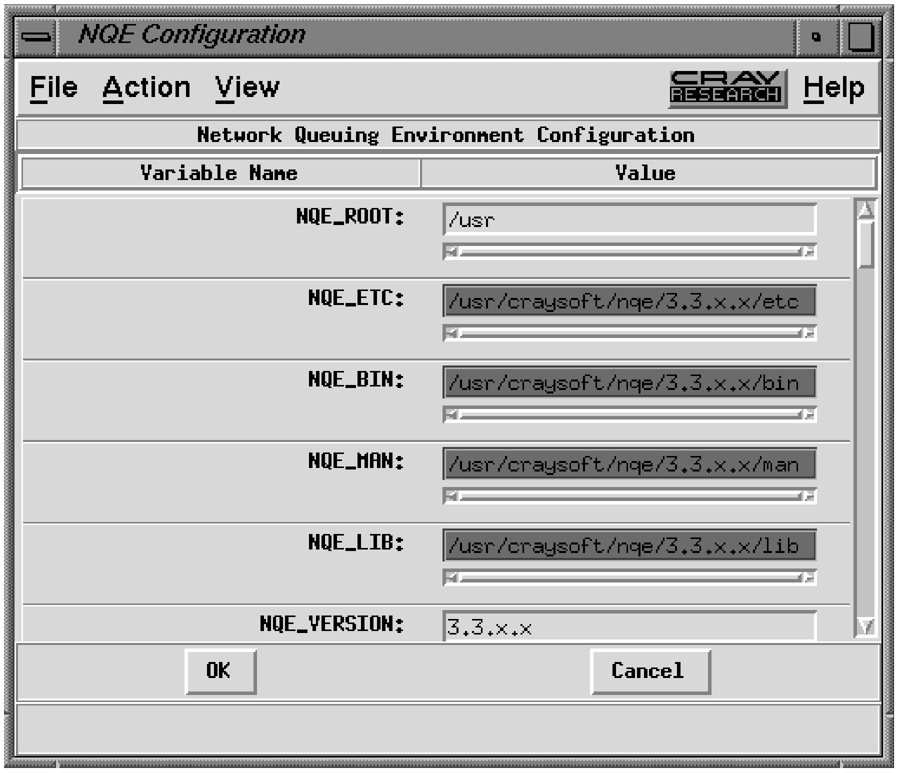
	/usr
NQE_VERSION:	3.3.x.x
NQE_DEFAULT_COMPLIST:	NQS,NLB,COLLECTOR
NQE_SPOOL:	/usr/spool/nqe
NLB_SERVER:	arctic
NQE_NLB_PORT:	604

View menu.

Figure 12. `NQE Configuration` Display, `View` Menu

The `Condensed` view displays only the variables that you may modify. This is the default view when you execute the `nqeconfig`(8) command. The `Condensed` view is shown in Figure 9, page 37.

The `Full` view, as shown below, displays all variables that will be placed in the output file:

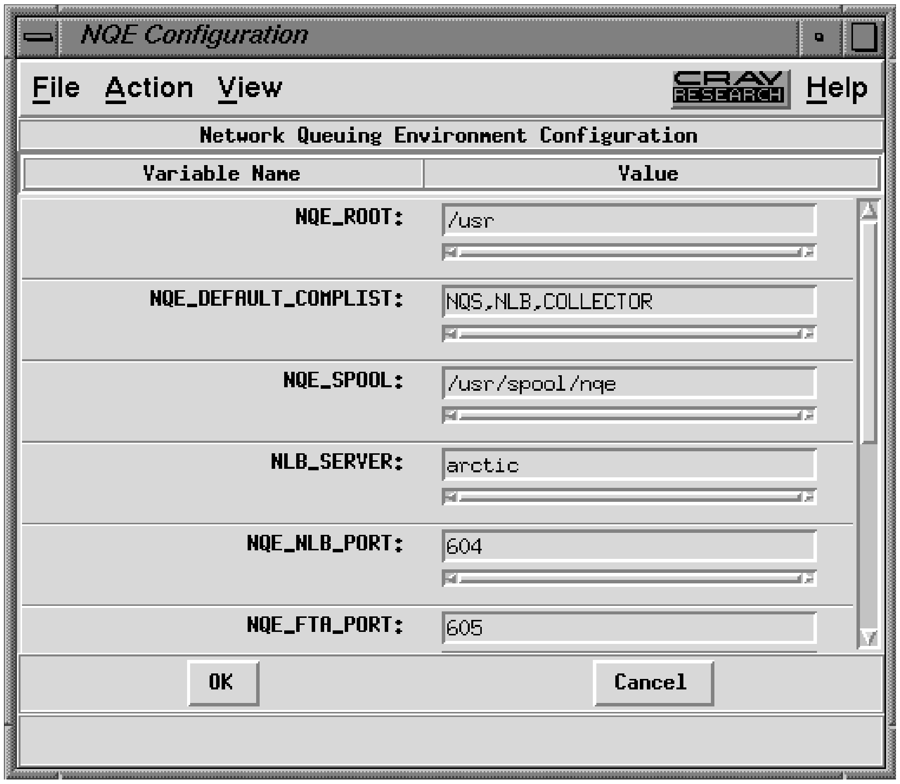


Network Queuing Environment Configuration

Variable Name	Value
NQE_ROOT:	/usr
NQE_ETC:	/usr/craysoft/nqe/3.3.x.x/etc
NQE_BIN:	/usr/craysoft/nqe/3.3.x.x/bin
NQE_MAN:	/usr/craysoft/nqe/3.3.x.x/man
NQE_LIB:	/usr/craysoft/nqe/3.3.x.x/lib
NQE_VERSION:	3.3.x.x

Figure 13. `NQE Configuration` Display, `Full` View

The `Install` view, which is shown as follows, is functional only during the NQE installation process; this view applies to all systems except UNICOS/mk systems, UNICOS systems, and IRIX systems that use the `inst` utility:



Network Queuing Environment Configuration

Variable Name	Value
NQE_ROOT:	/usr
NQE_DEFAULT_COMPLIST:	NQS,NLB,COLLECTOR
NQE_SPOOL:	/usr/spool/nqe
NLB_SERVER:	arctic
NQE_NLB_PORT:	604
NQE_FTA_PORT:	605

Figure 14. `NQE Configuration` Display, `Install` View

Starting and Stopping NQE [4]

This chapter describes how to start and stop NQE.

⚠ **Caution:** You must monitor the files in `$NQE_SPOOL/log` to ensure the log files for the startup and shutdown utilities (for `nqeinit`(8), `nqestop`(8), `qstart`(8), and `qstop`(8)) and related utilities do not accumulate and consume space needlessly.

4.1 Starting NQE

The `nqeinit`(8) script is called to start the NQE cluster and to initialize its components. It checks the `NQE_DEFAULT_COMPLIST` variable in the `nqeinfo`(5) file for the list of components to be started. You can set this list to one or more of any of the following valid NQE components:

`NQS`	Network Queuing System
`NLB`	Network Load Balancer
`COLLECTOR`	NLB collector
`NQEDB`	NQE database
`MONITOR`	NQE database monitor
`SCHEDULER`	NQS scheduler
`LWS`	Lightweight server

Beginning with the NQE 3.3 release, the default component list consists of the following components: `NQS`, `NLB`, and `COLLECTOR`.

Component names in the `NQE_DEFAULT_COMPLIST` component list can be specified in upper or lower case, but not mixed case for one component name. When more than one component name is specified (that is, a list of components), the component names are separated by a comma (`,`).

You can use the `-C` *component* option of the `nqeinit` command to override the `NQE_DEFAULT_COMPLIST` variable component list. When you use the `nqeinit -C` option, only the components specified (see valid component names in Section 4.1, page 45, above) are started and the `NQE_DEFAULT_COMPLIST` variable in the `nqeinfo` file is ignored for this invocation of the `nqeinit`(8) command.

Note: For UNICOS systems that run only the NQE subset (NQS and FTA components), use the `qstart`(8) command instead of the `nqeinit`(8) script to start NQS. For additional information, see Section 6.2, page 134.

Note: Start any NQE nodes configured to run the NQE database server or the NLB server first because each node once started sends system status information to the NLB server for use in load balancing.

Typically, `nqeinit`(8) should be called as part of the normal system startup procedure by using either a direct call in a system startup script or a startup utility such as `sdaemon`(8).

For information about `nqeinit`(8) command-line options, see the `nqeinit`(8) man page.

For information about symbolic links used by NQE, see the `nqemaint`(8) man page.

4.2 Stopping NQE

The `nqestop`(8) script is called to stop the NQE cluster and to shutdown its components. It checks the `NQE_DEFAULT_COMPLIST` variable in the `nqeinfo`(5) file for the list of components to be stopped. You can set this list to one or more of any of the following valid NQE components: `NQS`, `COLLECTOR`, `NLB`, `NQEDB`, `MONITOR`, `SCHEDULER`, and `LWS`. For more information on these components, see Section 4.1, page 45. For more information on the `NQE_DEFAULT_COMPLIST` variable, see Section 4.1, page 45.

You can use the `-C` *component* option of the `nqestop` command to override the `NQE_DEFAULT_COMPLIST` variable component list. When you use the `nqeinit -C` option, only the components specified (see valid component names in , above) are stopped and the `NQE_DEFAULT_COMPLIST` variable in the `nqeinfo` file is ignored for that invocation of the `nqestop`(8) command.

Note: For UNICOS systems that run only the NQE subset (NQS and FTA components), use the `qstop`(8) command to stop NQS. For additional information, see Section 6.3, page 140

Typically, `nqestop`(8) should be called as part of the normal system shutdown procedure by using either a direct call in a system shutdown script or a shutdown utility such as `sdaemon`(8).

For information about `nqestop`(8) command-line options, see the `nqestop`(8) man page.

NQS Configuration [5]

When you install NQE, a default configuration is provided for you. You may not find it necessary to make many changes to the initial installation and configuration as it is described in the installation instructions. However, if you do need to make changes to your NQE configuration, this chapter describes how to make changes to the NQS configuration in the network and on the local host. All of these actions are performed using the `qmgr` utility. An NQS manager can also perform the daily operations of the NQS system as described in Chapter 6, page 133.

This chapter discusses the following topics:

- Configuration guidelines
- Overview of manager actions
- Defining the NQS hosts in the network
- Defining managers and operators
- Configuring the scope of the user status display
- Defining NQS queues
- Defining a default queue
- Restricting user access to queues
- Defining queue complexes
- Defining global limits
- Defining per-request and per-process limits
- Defining job scheduling parameters
- Defining the political scheduling daemon threshold
- Unified Resource Manager (URM) for UNICOS systems
- Miser scheduler for IRIX systems
- NQS periodic checkpointing
- Setting the log file

- Setting the debug level
- Establishing NQS accounting procedures
- Setting the validation strategy for remote access
- Defining other system parameters (such as the default time a request can wait in a pipe queue and the name of the user who sends NQS mail)
- Using a script file as input to the `qmgr` utility, and generating a script file for the currently defined configuration
- Output validation
- NQS and multilevel security on UNICOS systems and on UNICOS/mk systems
- NQS user exits for UNICOS and UNICOS/mk systems

5.1 Configuration Guidelines

You can use the `qmgr`(8) subsystem to configure the local NQS host to meet site-specific requirements. However, many of the NQS parameters are already configured with a chosen default value and may not require modification.

Usually, NQS configuration is done only once, when NQS is started for the first time. This configuration is preserved through subsequent shutdowns and startups. You can change the configuration at any NQS startup by providing a new configuration file as input to the `qstart -i` command or the `nqeinit -i` command (see the `qstart`(8) and `nqeinit`(8) man pages for more information).

You should consider the following guidelines when configuring NQS:

- Abbreviated forms of commands can be used with `qmgr`; however, you should use the full form in standard configuration scripts to avoid confusion.
- If you do not use load balancing, you can provide better control by defining batch queues as `pipeonly` and creating pipe queues that have batch queue destinations.
- Avoid configuring pipe queues that have other local pipe queues as destinations, which can create a loop. No check exists for an occurrence of this in NQS.

- When a request is forwarded from a nonload-balancing pipe queue, the destinations are considered in the order you configured them by using the `qmgr add destination` command. A request is accepted into the first batch queue with per-process and per-request limits that are not exceeded by the request. Therefore, always set the order for pipe queue destinations and batch queue limits so that a batch request is accepted by the most appropriate queue.
- The scheduler weighting factors have significant value only when they are compared with each other. See Section 2.1.4, page 15, for a description of intraqueue priority and Section 5.12, page 82, for a description and examples of the `set sched_factor` commands.
- When defining `checkpoint_directory`, consider whether a specific device has sufficient space available to save the checkpoint images of all currently executing batch requests.

  The *absolute_pathname* that you select as the checkpoint directory must be an absolute path name to a valid directory that has only owner read, write, and search permissions (NQS sets the permission bits to 0700). The directory named by *absolute_pathname* must also be retained across UNICOS system restarts (you should not use the `/tmp` directory) and should be dedicated to NQS checkpoint files. The checkpoint files that are created are used to restart the NQS jobs when NQS restarts.

5.2 Overview of NQS Manager Actions

An NQS manager is responsible for the configuration of an NQS system. An NQS manager also can perform all of the operator actions; see Chapter 6, page 133, for a description of operator actions.

The actions for managing a configuration include using the `qmgr` utility to do the following:

- Define the NQS systems in the network
- Define the users who can be NQS managers or operators
- Create NQS queues
- Restrict user access (if desired) to NQS queues, and set the type of user validation to authenticate the users' identities
- Define other system parameters such as the name of the default NQS queue, the name of the log file, and the retry limits to use when an attempt to send a request on to another system fails

After the configuration is initially defined, an NQS manager can change it by using the `qmgr` utility. Changes can be made to any of the parameters when the NQS system is running without stopping and restarting the system.

> **Note:** Except for defining the list of NQS systems in the network, the `nqsdaemon` must be started before you can define or change any other configuration items; see Section 6.2, page 134, for a description of how to start `nqsdaemon`.

5.3 Defining NQS Hosts in the Network

Each NQS system in the network must know about the other NQS systems in the network. If NQS will not be communicating with any remote hosts, you need only the local host defined. Typically, the `qmgr` commands used to define hosts appear in the NQS configuration file.

Because each NQS server in a TCP/IP network can have multiple host names and Internet addresses (usually one for each network to which the server is connected or for each interface the server has), it may not be sufficient to uniquely identify the server by host name and Internet address. Therefore, NQS uses machine IDs (mids) to define each server in the network.

When NQS tries to connect between two hosts, it verifies that the name provided by the TCP/IP protocol has the same name and a corresponding mid on both hosts.

The *hostname* or alias associated with the mid is used to obtain the network address from the `/etc/hosts` file. When NQS detects a connection from another machine, it is given the peer network address, which is then located in the `/etc/hosts` file, yielding the network path corresponding to an *alias* or *hostname* in the network database. Some systems may store host name information on NIS, NIS+, or DSN. The network path is then used to verify the mid of the peer. If the mids and names do not match, the connection is refused.

Table 1 describes the elements in each host definition.

Table 1. Information for defining machines

Item	Description
mid	A machine ID number in the NQS database that has a maximum value of 2^{31}-1 and is unique for each machine in the network. If you do not supply a *mid*, NQS creates one for you from the lower 31 bits of the TCP/IP address of the host.
hostname	A unique name for each machine in the network; the name specified must correspond to the machine's entry in the `/etc/hosts` file (see `hosts`(5) for more information).
agents	Agents available to spool output on the machine. The following agents are supported (see Section 5.3.1, page 52): `fta` File Transfer Agent (FTA) `nqs` NQS internal spooling
alias	A local machine name or another network interface that consists of a maximum of 255 characters. You must define all network interfaces or requests will be rejected.

If you are configuring more than one NQS server, you must add NQS mids to each NQS database. NQE clients do not need mids.

The mid and host name must be the same in the database of each machine in the network. Aliases must be unique to each machine in the local configuration, but may differ from server to server.

Note: After the NQS database is generated, the mid for the local machine should not be changed. If a change is required, the local NQS database must be re-created and all requests are lost.

The *agents* element defines how the standard error and standard output files of completed requests will be returned to the submitting user. For more information, see Section 5.3.1.

A machine *alias* is known only to the local server. However, a server can be known by more than one alias. The alias list may include host names associated with any interface on the local host. An alias is required for each possible route between hosts.

5.3.1 Configuring Remote Output Agents

You can set the output agent for each NQS host defined in the NQS database. The output agent at each host defines how the standard error and standard output files of completed requests are returned to the submitting user.

> **Note:** Output agents are not configured for clients. Client output return uses the File Transfer Agent (FTA), then RCP. NQS always uses FTA with the default domain, which is `inet`. For information about FTA and network peer-to-peer authorization (NPPA) of users, see Chapter 13, page 317.

When you specify `nqs` (the default output agent), NQS tries to connect directly to the remote NQS host to transfer the files. If any errors occur during the processing, the output is returned to the user's home directory on the execution host. If for some reason, this is not possible (for example, if the owner does not have write permission), NQS will place the files in `$NQE_NQS_SPOOL/private/root/failed`. `NQE_NQS_SPOOL` is defined in the `nqeinfo` file.

When you specify `fta`, NQS passes control of the output file transfer to FTA by creating an FTA *file transfer request* to move the output file to the remote host. When you specify `fta` as the output agent, NQS validation is used; for detailed information, see Chapter 13, page 317, and the `nqsfts`(8) man page. If any errors occur during the processing of the output file transfer, FTA will save the transfer request and try to transfer it again, or return the output to the user's home directory on the execution host. If the error is transient (for example, the network failed during the file transfer), FTA retries the transfer request.

You can use the following `qmgr` commands to add or delete the output agents, respectively. You must include either the machine ID (*mid*) or the host name:

```
add output_agent agent mid|host
delete output_agent agent mid|host
```

The order in which you add output agents is important. The order in which you add the agents becomes the order in which the agents are used to attempt transfer.

To display current configuration, use either of the following `qmgr` commands:

```
show mid
show mid hostname|mid
```

In the resulting display, the output agents are shown in the column called `AGENTS`.

The following list shows the output agents that you can specify and their descriptions:

Agent	Description
`<NONE>`	NQS immediately places the output files in the user's home directory on the host that executed the request. The user receives a mail message when the files are returned. To set the agent to `<NONE>`, use the following command: `delete output_agent nqs [`*mid*`,` *hostname*`]`
`fta`	NQS creates an FTA file transfer request to transfer the output file to the remote host.
`fta nqs`	NQS tries to create an FTA file transfer request to handle the output file transfer. If FTA fails, NQS tries to connect to the NQS system on the remote host and directly transfer the request output file.
`nqs`	NQS tries to connect to the NQS system on the remote host and directly transfer the request's output file. This option directs NQS to handle the output file transfer directly. This is the default agent.
`nqs fta`	NQS tries to connect to the NQS system on the to the remote host and directly transfer the request's output file. If NQS fails, NQS tries to create an FTA file transfer request to handle the output file transfer.

The `nqs` output agent is configured automatically unless you configure the output agent differently.

Note: In the case that all transfer agents fail to return output, the output is placed in the user's home directory on the host that executed the request. This is the same as configuring with output agent `<NONE>`.

An output agent should always be configured for the local host to avoid output being inadvertently returned to the user's home directory on the host of execution; this is usually done by specifying the `nqs` agent.

See Chapter 13, page 317, and the `fta`(8), `nqsfts`(8), and `fta.conf`(5) man pages for details about how to configure FTA. NQS uses the FTA domain name `nqs-rqs` to spool output over TCP/IP.

5.3.2 Commands for Defining Machines

You can use the following `qmgr` command to add a new entry to the *mid* database (see Table 1, page 51, for an explanation of *mid*, *hostname*, *agent*, and *alias*). You do not have to specify a *mid*; if you do not, `qmgr` creates one for you from the lower 31 bits of the TCP/IP address for the host. An optional list of aliases may be supplied on the same line. You do not enter the commands on client systems or issue `add mid` commands for client machines.

`add mid [`*mid*`]` *hostname*`[`*alias alias ...*`]`

The following command adds an alias name to the list of aliases for the system with the specified *mid* or *hostname*:

`add name [=]` *alias mid*|*hostname*

The following command adds an output agent to the list of output agents for the system with the specified *mid* or *hostname*.

`add output_agent [=]` *agent mid*|*hostname*

See Section 5.3.1, page 52, for more information on output agents.

A similar series of `delete` commands can be used to delete an entry completely from the list, delete an alias name from an entry, and delete an output agent from an entry. You can specify either a *mid* or a *hostname*:

`delete mid` *mid*|*hostname*
`delete name [=]` *alias*
`delete output_agent [=]` *agent mid*|*hostname*

To display information about a specific *mid* or *hostname*, use the following `qmgr` command:

`show mid [`*mid*|*hostname*`]`

To display information about all known *mids,* use the following `qmgr` command:

```
show mid
```

See Section 5.3.1, page 52, for an example display.

5.3.3 TCP/IP Network Configuration Example

Figure 15 shows an example of a network that consists of four machines, each with two paths. In the example, *hn* refers to host name, and *na* refers to network address.

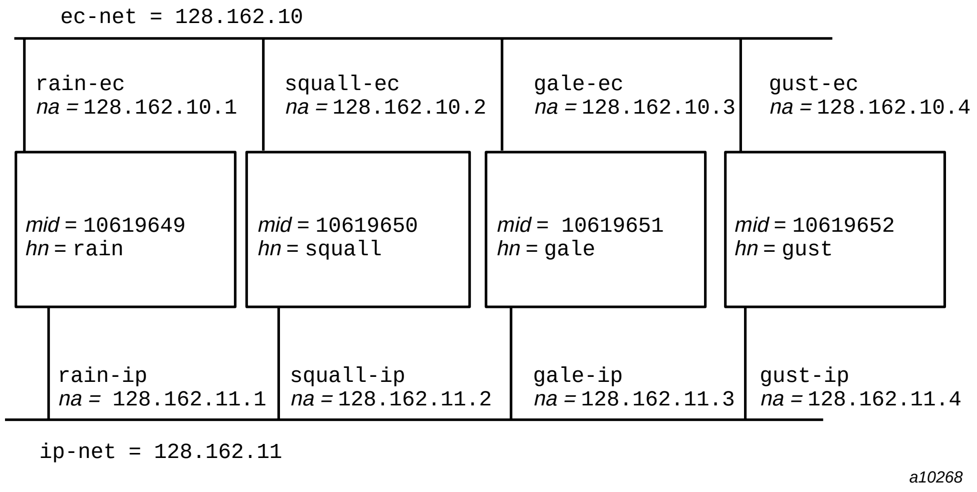


Figure 15. Example of a TCP/IP network

In this example, the `/etc/hosts` file contains the following:

```
128.162.12.0     rain rain-ip
128.162.10.1     rain-ec
128.162.11.2     squall squall-ip
128.162.10.2     squall-ec
128.162.11.3     gale gale-ip
128.162.10.3     gale-ec
128.162.11.4     gust gust-ip
128.162.10.4     gust-ec
```

Table 2 shows the contents of the NQS network database for the previous example:

Table 2. Machine IDs, names, and output agents

Machine ID	Principal name	Agents	Aliases
`10619649`	`rain`	`nqs fta`	`rain-ip, rain-ec`
`10619650`	`squall`	`nqs fta`	`squall-ip, squall-ec`
`10619651`	`gale`	`nqs`	`gale-ip, gale-ec`
`10619652`	`mist`	`nqs`	`mist-ip, mist-ec`

The `qmgr` commands used to create the database in the preceding example are as follows (you may supply your own *mid* value):

```
add mid rain rain-ip rain-ec
add output_agent fta rain
add mid squall squall-ip squall-ec
add output_agent fta squall
add mid gale gale-ip gale-ec
add mid mist mist-ip mist-ec
```

To create the database on each NQS server, you would enter these commands on each.

5.3.4 Stand-alone Configuration Example

If you are running NQS but will not be accessing any other systems in the network, a mid database must still be created, consisting of the machine on which NQS is running. This mid database is automatically created for you during installation and system startup.

5.4 Defining Managers and Operators

You can define two special classes of users: managers and operators. NQS `qmgr` *managers* have full access to all `qmgr` commands to monitor, control, and configure NQS. *Operators* are allowed to monitor and control all queues and requests. The only `qmgr` commands available to users without special privileges are `show`, `help`, `exit`, and `quit`.

When NQS is initially installed, `root` must run `qmgr` first to define other users as either an NQS manager or an operator. `root` cannot be deleted from the list of NQS managers. On a UNICOS system that is running MLS or on a UNICOS/mk system that is running Cray ML-Safe and that has a PAL assigned to the `qmgr` program, people in the default `qmgr` PAL categories can also define other users as either an NQS manager or operator.

Managers can execute all `qmgr` commands; operators can execute only a subset of the `qmgr` commands that allow them to monitor and control submitted batch requests when they are in local NQS queues. An operator cannot define another user as an operator or as a manager. Only a manager can add and delete managers or operators.

5.4.1 Commands for Defining Managers and Operators

The following `qmgr` commands add a user name to the list of managers or list of operators, respectively, for the local NQS system:

```
add managers username:m
add managers username:o
```

The suffix indicates that the user is a manager (`:m`) or an operator (`:o`).

If you want to set the list of NQS managers or NQS operators to a single user name (in addition to `root`), you can use one of the following two commands because any users defined previously as managers or operators (except `root`) are removed from the list.

```
set managers username:m
set managers username:o
```

To delete a user name from the list of managers or operators, use one of the following commands.

```
delete managers username:m
delete managers username:o
```

To display the currently defined managers and operators, use the following `qmgr` command:

```
show managers
```

See Section 6.4.2, page 150, for an example display.

5.4.2 Example of Adding Managers and Operators

To add a user called `abc` to the list of operators or to set the list of managers to one user called `xyz`, use the following `qmgr` commands:

```
add managers abc:o
set managers xyz:m
```

5.5 Configuring the Scope of the User Status Display

NQE allows the administrator to expand the default scope of the information displayed to non-NQS managers using the `qstat`(1) or `cqstatl`(1) commands. The default behavior for these commands is to only display information to non-NQS managers regarding jobs that they have submitted themselves. To allow users to display the status of all jobs residing at that NQS node when they execute the `qstat`(1) or `cqstatl`(1) command, use the `nqeconfig`(1) command and define the `nqeinfo` file variable `NQE_NQS_QSTAT_SHOWALL` to be 1.

Note: NQE must be stopped and restarted for this change to take effect.

5.6 Defining NQS Queues

NQS has both batch and pipe queues. Batch queues accept and process batch requests; pipe queues receive batch requests and then route those requests to another destination. Section 2.3, page 17, provides background information about queues and how they work.

NQE is shipped with one pipe queue and one batch queue configured on each NQS server. The default pipe queue is `nqenlb` . The default batch queue is `nqebatch`. This configuration uses the `nqenlb` queue to send requests to the NLB, which then sends requests to `nqebatch` on the most appropriate system, based on an NLB policy. The default NLB policy, called `nqs`, sends batch requests to the system with the most available CPU cycles. If requests are submitted to the queue `nqebatch`, the request runs on the local NQS server.

To create batch or pipe queues, use the following `qmgr` commands:

```
create batch_queue characteristics
create pipe_queue characteristics
```

Table 3 shows the *characteristics* you can define for an NQS queue.

Table 3. Definable queue characteristics

Characteristic	Description	Queue type
queue_name	Each NQS queue must have a name that is unique among the other NQS queues defined at this host. This is the name users use when submitting requests. The queue name can consist of a maximum of 15 characters, and it can consist of any printable character, except `@`, `=`, `(`, `)`, `]`, and the space character. The name cannot begin with a number (0 through 9).	Batch and Pipe
destination	Each NQS pipe queue must have one or more destinations, which can be either another NQS pipe or batch queue. The destination queue can be in one of the following forms: *local_queuename* *local_queuename@local_hostname* *remote_queuename@local_hostname* *remote_queuename@[remote_mid]* Any *hostname* must be defined in the NQS configuration machine ID (*mid*) database. For a remote queue, the last of the preceding forms lets you specify the *mid* rather than the host name (for example, `bqueue15@[23]`). The *mid* must be enclosed within brackets.	Pipe
run_limit	By default, a queue can process only one request at a time (although any number may be waiting in the queue). You can increase this limit by defining a run limit for a queue. This limit sets the maximum number of requests that can be processed in the queue at any one time.	Batch and Pipe

Characteristic	Description	Queue type
interqueue_priority	Each queue has an interqueue priority. This defines the order in which queues are searched to find the next request to process. The queue with the highest priority is scanned first. The priority is an unsigned integer in the range from 0 (lowest priority) through 63 (highest priority). Requests also have an *intraqueue* priority that a user can specify when submitting a request. This is different from interqueue priority. The request intraqueue priority determines the order in which requests are considered for initiation when a queue is searched.	Batch and Pipe
`loadonly` qualifier	By default, an NQS batch queue can receive requests either directly from a user submitting the request or from a pipe queue. A batch queue can be declared `loadonly`, meaning that it can accept only a limited number of requests and requests must be routed through a local or remote pipe queue. The number of requests that can be queued is restricted to the *runlimit* of that queue (the *runlimit* is the number of requests that can be processed). Therefore, if there are no other limits to consider, a `loadonly` queue accepts only the number of requests that it can run. See Section 2.5.2, page 25, for a description of how `loadonly` queues can be used.	Batch
`pipeonly` qualifier	If a queue has the characteristic `pipeonly`, a user cannot submit a request directly to the queue. The queue can receive only requests sent from a local NQS pipe queue.	Batch and Pipe

Characteristic	Description	Queue type
server	NQS pipe queues can be associated with a server from which they will receive requests. This server is an image the pipe queue runs to do work. For example, you must define a server when you create pipe queues for load balancing. If you specify `pipeclient`, NQS looks for the binary in the `NQE_bin` directory as configured in the `/etc/nqeinfo` file. If the `pipeclient` is not in that directory, then you must specify the absolute location. It is recommended that you use `pipeclient`. A pipe queue can be defined as a destination-selection (load-balancing), as described in Section 5.6.4, page 65. When you define a destination-selection queue, you must include, as part of the *server* definition, the characters `CRI_DS` in uppercase, the load-balancing policy name (the default is `nqs`), and the TCP/IP host and service name on which the NLB server(s) are running (the default is the `NLB_SERVER` defined in the `nqeinfo` file).	Pipe

These are the basic characteristics of an NQS queue. You also can define other controls on the use of a queue, as follows:

- You can define a list of users who have permission to use a queue. Requests belonging to a user who is not on this list cannot be placed in the queue. See Section 5.8, page 69.
- You can specify global system defaults for queues. See Section 5.10, page 73.
- You can specify limits on the number and size of the requests in a queue, as described in Section 5.6.1.

5.6.1 Setting Queue Limits

To specify limits on the requests running in a queue, use the following `qmgr` command:

`set queue` *option = limit queuename*

An NQS operator or manager can executed this command. The *option* can be one of the following:

Option	Description
`group_limit`	Sets the maximum number of requests allowed to run in the queue at one time from users in one group.
`memory_limit`	Sets the maximum amount of memory that can be requested by all running requests in the queue at one time.
`run_limit`	Sets the maximum number of batch requests that can run in a queue concurrently. You also can set this limit by using the command `create batch queue run_limit`.
`user_limit`	Sets the maximum number of requests allowed to be run in the queue at one time by one user.
`mpp_pe_limit`	Sets the maximum number of MPP processing elements (PEs) that are available to all batch requests running concurrently in a queue.
`quickfile_limit`	Sets the quickfile space limit for a queue. The limit determines the maximum quickfile space that can be allocated to all batch requests running concurrently in a queue. The queue must exist already and be a batch type queue.

Some queue limits are enforced only for certain machines. For more information on these limits, see Table 4, page 77.

5.6.2 Hints for Defining Queues

The NQS daemon must be started before you can create or change the characteristics of a queue because `nqsdaemon` is the process that manipulates queues.

The destinations for an NQS pipe queue are usually NQS batch queues, although they could be any type of queue, such as a pipe queue on a specific remote machine.

Before a queue can receive requests, it must be enabled by using the `qmgr` command `enable queue` (see Section 6.5.1, page 160). Enabling a queue means that requests can be placed in the queue. Before a queue can process its requests, the queue must be started using the `qmgr` command `start queue` (see Section 6.5.2, page 161).

Avoid configuring pipe queues that have other local pipe queues as destinations, which can create a loop. NQS has no check for this situation.

5.6.3 Commands Used to Define Queues

The first time a queue is defined, it must be created by using one of the following `qmgr` commands:

```
create batch_queue queuename \
priority = queue-priority[loadonly] [pipeonly] \
[run_limit = run-limit]

create pipe_queue queuename \
priority = queue-priority[pipeonly] \
[destination = (destinations)] \
[run_limit = run-limit] \
[server = (pipeclient_and_options)]
```

See Table 3, page 59, for a description of these parameters. The *queuename* argument and the `priority` parameter are required. A pipe queue (unless it is a destination-selection (load-balancing) queue) also must have at least one destination defined by using this command, the `add destination` command, or the `set destination` command.

The *destinations* argument can be one or more destinations; destinations must be separated with a comma.

Note: See Section 5.6.4, page 65, for information on configuring a destination-selection queue.

5.6.3.1 Adding Destinations to a Pipe Queue

To add destinations to those already defined for a pipe queue, use the following command:

```
add destination [=] destinations queuename \ [position]
```

The *destinations* argument is one or more destinations to be added to the list of destinations for *queuename*. If you specify more than one destination, they must be enclosed in parentheses and separated with a comma. If you omit the *position* argument, the destinations are added at the end of the existing destinations defined for *queuename*.

The *position* argument can be one of the following:

Position	Description
`first`	Places the specified *destinations* at the start of the existing list
`before` *old-des*	Places the specified destinations in the list immediately before the existing destination *old-des*
`after` *old-des*	Places the specified destinations in the list immediately after the existing destination *old-des*
`last`	Places the specified *destinations* at the end of the existing list

Destinations are not used in destination-selection (load-balancing) queues.

Note: A pipe queue's destination list should not contain any elements that are disabled batch queues. In the event that this does occur, any jobs that are submitted to the pipe queue remain stuck in the pipe queue if they cannot be sent to any of the other destinations in the list. Because the disabled batch queue exists, NQS waits for the queue to become enabled or for it to be deleted before it moves the job from the pipe queue. To ensure that jobs are not sent to a particular destination in the pipe queue's list, remove the destination from the list rather than disabling the batch queue.

5.6.3.2 Resetting Destinations for a Queue

If you want to change the destinations for a queue, you can use the following command:

`set destination [=]` *destinations queuename*

If you include more than one *destination*, separate them with a comma.

This command discards all the previous destinations.

5.6.3.3 Deleting Destinations from a Queue

To delete one or more destinations from the list of those defined for a pipe queue, use the following command:

```
delete destination [=] destinations queuename
```

Separate the *destinations* with a comma.

Any requests in the queue that are currently being transferred to a destination that is deleted by this command are allowed to transfer successfully.

Requests can be submitted to a queue that has no destinations, but the requests are not routed until a destination is defined for the queue.

5.6.3.4 Changing the Interqueue Priority for a Queue

To change the interqueue priority for an existing queue, use the following command:

```
set priority = priority queuename
```

5.6.3.5 Changing the Run Limit for a Queue

To change the run limit for an existing NQS queue, use the following command:

```
set queue run_limit = run-limit queuename
```

5.6.3.6 Changing the Pipe Client Server for a Queue

To change the pipe client server that is invoked to process a request sent to the specified NQS queue, use the following command:

```
set pipe_client = (client-and-arguments) queuename
```

5.6.4 Commands for Configuring Destination-selection Queues

Load balancing is described in Chapter 8, page 193. The pipe queue `nqenlb` is defined by default to accept requests that use load balancing. The following `qmgr` command creates a destination-selection (load-balancing) pipe queue named `nlbqueue`:

```
create pipe_queue nlbqueue priority=priority \
server=(pipeclient CRI_DS [policyname][hostname:port])
```

The arguments unique to creating a destination selection queue are as follows:

- The name of the executable for the pipe client server (`pipeclient`).
- The characters `CRI_DS` in uppercase, which mark this as a destination-selection pipe queue.
- The load-balancing policy name (such as `nqs`). This policy must be defined in the NLB `policies` file. If this field is omitted, the policy name defaults to `nqs`.
- An optional list of 1 to 8 *host:port* pairs that define the TCP/IP host and service name where the NLB server(s) are running. If this field is omitted, the NLB database server name defaults to the hosts configured in the NLB. The `NLB_SERVER` is defined in the `nqeinfo` file. There should be one pair for each NLB server from which you want to gather host information. Only one is necessary, but if servers are replicated for redundancy in case of network failure, all servers should be included. In the example, *hostname* is the host name where the server resides and *port* is a service name (from the `/etc/services` file) or a port number on which the server is listening.

After the `qmgr create` command is issued, users can submit requests for destination selection to the queue `nlbqueue`.

If you want this new queue to be the default for requests, use the following `qmgr` command:

```
set default batch_request queue nlbqueue
```

5.6.5 Displaying Details of the Currently Defined Queues

You must use the `cqstatl` or `qstat` command (instead of the `qmgr` utility) to display details about an NQS queue; for example:

```
cqstatl -f queues
```

See Section 6.4.4, page 153, for an example display.

5.6.6 Examples of Defining Queues

In the following example, an NQS pipe queue is required that routes requests to a remote NQS system called `host1`. A maximum of two requests can be processed in the NQS pipe queue at any one time. The following `qmgr` commands are required:

```
% qmgr
Qmgr: create pipe_queue standard priority=60  server=(pipeclient)
Qmgr: set queue run_limit=2 standard
Qmgr: add destination st_express@host1 standard
Qmgr: add destination st_little@host1 standard
Qmgr: add destination st_medium@host1 standard
Qmgr: quit%
```

This queue could also be defined in one command, as follows:

```
% qmgr
Qmgr: create pipe_queue standard priority=60 server=(pipeclient)
destination=(st_express@host1,st_little@host1,st_medium@host1)
run_limit=2
```

5.6.6.1 Example of Changing the Order of the Destinations

The order of the destinations is important because this is the order in which NQS tries to route a request. In this example, if you want to move `st_little` to the beginning of the destination list, you could type in the following two `qmgr` commands:

```
delete destination st_little@host1 standard
add destination st_little@host1 standard first
```

To redefine all the destinations for the queue, use the following `qmgr` command:

```
set destination=(st_little@host1, st_express@host1,
st_medium@host1) standard
```

5.6.7 Deleting a Queue

After a queue has been stopped and disabled, delete it by using the following `qmgr` command:

`delete queue` *queuename*

If the *queuename* argument is also the default queue, you should define a new default queue (see Section 5.7).

5.7 Defining a Default Queue

To define an NQS queue to be the default queue, use the following `qmgr` command:

`set default batch_request queue` *queuename*

The *queuename* argument can be the name of any NQS batch queue that you have already created.

To change the definition of the default queue so that there is no longer a default queue, use either of the following `qmgr` commands:

```
set no_default batch_request queue
set default batch_request queue none
```

5.7.1 Displaying the Current Default Queue

To display the current default queue, use the following `qmgr` command:

```
show parameters
```

The setting is shown next to `Default batch_request queue`. See Section 6.4.1, page 142, for an example display.

5.7.2 Example of Defining a Default Queue

To define the default NQS queue to be `bqueue10`, use the following `qmgr` command:

```
set default batch_request queue bqueue10
```

5.8 Restricting User Access to Queues

When you initially create an NQS queue, any user can submit a request to it. To restrict user access to a queue, create a list of the users and a list of the groups whose members can access the queue. These lists are held in the NQS configuration database and NQS manager can edit them by using `qmgr` commands.

5.8.1 Hints on Restricting User Access to Queues

For a user to have access to a queue, one of the following requirements must be true:

- Access to the queue is unrestricted (this is true when a queue is first created or when the `qmgr` command `set unrestricted_access` was issued for the queue).
- The user belongs to a group that was added to the list of groups that can access the queue by using the `qmgr` command `add groups`.

- The user was added to the list of users who can access the queue by using the `qmgr` command `add users`. `root` does not have automatic access to all queues.

If the access to the queue is unrestricted, you cannot use the `add groups` or `add users` command. You must first set the queue to have no access by using the `qmgr` command `set no_access`.

5.8.2 Commands Available to Restrict User Access

To restrict the access to the queue, use `qmgr` commands, as follows:

1. Define the queue as having no access by any user, using the following `qmgr` command:

   ```
   set no_access queuename
   ```

2. Add individual users, or user groups, to the list of users or groups that can submit requests to the queue, as follows:

   ```
   add users = (user-names) queuename
   add groups = (group-names) queuename
   ```

   The *user-names* argument is one or more user names; *group-names* is one or more group names. If you specify more than one user name or group name, you must enclose all the names in parentheses and separate them with a space or a comma. Use the numeric UNIX user IDs and group IDs (which must be enclosed in brackets `[]`) as an alternative to user names and group names.

If you later want to allow any user to access the queue, enter the following `qmgr` command:

```
set unrestricted_access queuename
```

To delete users or groups from the list of those allowed to access a queue, use the following `qmgr` commands:

```
delete users = (user-names) queuename
delete groups = (group-names) queuename
```

5.8.3 Displaying Details of the Currently Defined Queues

To display the current restrictions on user and group access to a queue, use the `cqstatl` or the `qstat` command instead of a `qmgr` utility; for example:

`cqstatl -f` *queues*

The access restrictions appear on the display under the heading `<ACCESS>`.

See Section 6.4.4, page 153, for an example display.

5.8.4 Example of Restricting User Access

In the following example, you want to restrict access to an NQS queue called `standard`. Unless the access permissions to a queue have been changed, any user can access it. You first issue the `set no_access` command to restrict all users, and then add those groups you want to have access. To restrict access to only those users belonging to a UNIX user group called `research`, enter the following two `qmgr` commands:

```
set no_access standard
add groups research standard
```

5.9 Defining Queue Complexes

To create a queue complex (a set of batch queues), use the following `qmgr` command:

`create complex = (`*queuename(s)*`)` *complexname*

To add or remove queues in an existing complex, use the following `qmgr` commands:

`add queues = (`*queuename(s)*`)` *complexname*
`remove queues = (`*queuename(s)*`)` *complexname*

Note: The difference between the commands `remove queues` and `delete queues` is important. The `remove queues` command removes a queue from the queue complex, but the queue still exists. The `delete queues` command deletes the queue completely.

After a complex has been created and the appropriate queues added to it, associate complex limits with it by using the following `qmgr` command:

`set complex` *option=limit complexname*

The *option* provides for control of the total number of concurrently running requests in the queues on the local host. *option* may be one of the following:

Option	Description
`group_limit`	Sets the maximum number of requests that all users in a group can run concurrently in all queues in the complex
`memory_limit`	Sets the maximum amount of memory for all requests running concurrently in all queues in the complex
`mpp_pe_limit`	Each batch queue has an associated MPP PE limit. This is the maximum number of MPP PEs that can be requested by all running jobs in the queue at any one time.
`quickfile_limit`	(UNICOS systems with SSD solid state storage devices) Each batch queue has an associated quick-file limit. This is the maximum size of secondary data segments that can be requested by all running jobs in the queue at any one time.
`run_limit`	Sets the maximum number of requests allowed to run concurrently in all queues in the complex
`user_limit`	Sets the maximum number of requests that any one user is allowed to run concurrently in all queues in the complex

Note: A batch request is considered for running only when all limits of all complexes of which the request is a member have been met.

NQS managers and operators can set queue complex limits. Some complex limits are enforced only on certain machines. For more information on what limits are enforced, see Table 4, page 77.

5.9.1 Queue Complex Examples

The following example creates a queue complex named `weather` that contains the `bqueue11`, `bqueue12`, and `bqueue13` queues:

```
create complex=( bqueue11,bqueue12,bqueue13) weather
```

The following example limits all users in one group to a maximum of 20 requests in the queue complex `weather`:

```
set complex group_limit=20 weather
```

5.10 Defining Global Limits

To set limits on the total workload executing concurrently under NQS control on the local host, use the following `qmgr` command:

`set global` *option=limit*

Queue limits restrict requests in queues and complex limits restrict requests in a complex. Global limits restrict the activity in the entire NQS system. NQS managers and operators can set global limits.

The *option* can be one of the following:

Option	Description
`batch_limit`	Sets the maximum number of batch requests allowed to run concurrently at the host.
`group_limit`	Sets the maximum number of batch requests all users in a group can run at the host.
`memory_limit`	Sets the maximum amount of memory that can be allocated to all batch requests running concurrently at the host.
`mpp_pe_limit`	The maximum number of MPP PEs that can be requested by all batch requests running concurrently under NQS control.
`pipe_limit`	Sets the maximum number of pipe requests allowed to run concurrently at the host.
`quickfile_limit`	(UNICOS systems with SSDs) The maximum amount of secondary data segment space that can

	be requested by all batch requests running concurrently under NQS control.
`tape_drive_limit`	Sets the maximum number of tape drives per device group that can be requested by all batch requests running concurrently at the host. NQS supports eight tape groups (see the `qmgr`(8) man page).
`user_limit`	Sets the maximum number of batch requests that any one user is allowed to run concurrently at the host.

Some global limits are enforced only on certain machines. For more information on what limits are enforced, see Table 4, page 77.

5.10.1 Displaying the Current Global Limits

To display the current global limits, use the following `qmgr` command:

```
show global_parameters
```

The resulting display resembles the following:

```
  Global batch group run-limit: 20
  Global batch run-limit:       30
  Global batch user run-limit: 20
  Global MPP Processor Element limit:   unspecified
  Global memory limit:          unlimited
  Global pipe limit:            20
  Global quick-file limit:      unlimited
  Global tape-drive a limit:    unspecified
  Global tape-drive b limit:    unspecified
  Global tape-drive c limit:    unspecified
  Global tape-drive d limit:    unspecified
  Global tape-drive e limit:    unspecified
  Global tape-drive f limit:    unspecified
  Global tape-drive g limit:    unspecified
  Global tape-drive h limit:    unspecified
```

You also can see these limits in the `show parameters` display. See Section 6.4.1, page 142, for an example. Some parameters are not enforced if the operating system does not support the feature, such as MPP processor elements.

5.10.2 Example of Setting Limits

To set global limits as they are displayed in the previous display, use the following `qmgr` commands:

```
set global group_limit = 20
set global batch_limit = 30
set global user_limit = 20
set global pipe_limit = 20
```

5.11 Defining Per-request and Per-process Limits

To set per-process and per-request limits on batch queues, use the following `qmgr` commands:

`set per_process` *option = limit queuename*
`set per_request` *option = limit queuename*

Per-request limits apply to the request as a whole; they limit the sum of the resources used by all processes started by the request. *Per-process limits* apply to individual processes started by a request (including the parent shell and each command executed). For example, the per-request memory size limit is a limit on the sum total of memory used by all processes started by the request; the per-process memory size limit is the maximum amount of memory that each process can use.

Per-process and per-request limits on a queue are compared with the limits set on a request. If the request's limits are lower, it can enter the queue. If the limits are higher, the request cannot enter the queue. If requests have improperly set limits and enter queues with lower limits than the request actually needs, the request may abort when it attempts to exceed limits imposed by the queue.

Table 4, page 77, shows the limits that can be enforced on NQS requests. This table uses the following notation:

Symbol	Meaning
Y	Limit is supported and enforced on this platform
N	Limit is **not** supported or enforced on this platform
PP	Limit is a per-process limit

PR	Limit is a per-request limit

Note: The limits included in Table 4 can be set by using the `qsub` command option listed in the table or by using the NQE GUI `Job Limits` submenu of the `Configure` menu on the `Submit` window. For detailed information about the command options, see the `qsub`(1) man page. For detailed information about using the NQE GUI `Job Limits` submenu of the `Configuration` menu, see the *NQE User's Guide*, publication SG–2148.

Table 4. Platform support of NQS limits

			Run-time enforcement on executing NQS					
Description	Limit type	`qsub` qualifier	UNICOS/mk and UNICOS	Solaris	AIX	DEC	HP-UX	IRIX
Core file size	PP	`-lc`	Y	Y	Y	Y	N	Y
Data segment size	PP	`-ld`	N	Y	Y	Y	N	Y
Permanent file size	PP	`-lf`	Y	Y	Y	Y	N	Y
	PR	`-lF`	Y	N	N	N	N	N
Memory size	PP	`-lm`	Y	N	N	N	N	Y
	PR	`-lM`	Y	N	N	N	N	Y[2]
MPP memory size	PP	`-l p_mpp_m`	Y	N	N	N	N	N
	PR	`-l mpp_m`	Y	N	N	N	N	N
Nice value	PP	`-ln`	Y	Y	Y	Y	Y	Y
Quick file size (not on UNICOS/mk)	PP	`-lq`	Y	N	N	N	N	N
	PR	`-lQ`	Y	N	N	N	N	N
Stack segment size	PP	`-ls`	N	Y	Y	Y	N	Y
CPU time	PP	`-lt`	Y	Y	Y	Y	N	Y
	PR	`-lT`	Y	Y[1]	Y[1]	Y[1]	N	Y[2]

1 The per-request limit is enforced as if it were per-process.

			Run-time enforcement on executing NQS					
Description	Limit type	`qsub` qualifier	UNICOS/mk and UNICOS	Solaris	AIX	DEC	HP-UX	IRIX
Tape drive limit	PR	`-lU`	Y	N	N	N	N	N
Temporary file size	PP	`-lv`	N	N	N	N	N	N
	PR	`-lV`	Y	N	N	N	N	N
Working set size	PP	`-lw`	N	Y	Y	Y	N	Y
MPP processing elements (T3D systems) or MPP application processing elements (T3E systems) or Number of processors (IRIX systems)	PR	`-l mpp_p`	Y	N	N	N	N	Y[2]
MPP residency time (T3D systems) or CPU time for application PEs (T3E systems)	PP	`-l p_mpp_t`	Y	N	N	N	N	N
	PR	`-l mpp_t`	Y	N	N	N	N	N
Shared memory size	PR	`-l shm_limit`	Y	N	N	N	N	N
Shared memory segment size	PR	`-l shm_segments`	Y	N	N	N	N	N

The following options to the `qmgr` commands `set per_process` and `set per_request` set per-process and per-request limits on NQS queues.

2 Optionally enabled. See Section 5.11.4, page 82.

Option	Description
`corefile_limit`	Sets the maximum size (in bytes) of a core file that may be created by a process in the specified batch queue.
`cpu_limit`	Sets the per-process or per-request maximum CPU time limit (in seconds) for the specified batch queue. On CRAY T3E systems, this CPU time limit applies to command PEs (commands and single-PE applications); for CRAY T3E applications PEs (multi-PE applications), maximum CPU time is set by using the `set per_process` and `set per_request mpp_limit` command. The value can be specified as *seconds*, as *minutes:seconds* or as *hours:minutes:seconds*.
`data_limit`	Sets the maximum size (in bytes) for the data segment of a process in the specified batch queue. This defines how far a program may extend its break with the `sbrk` () system call.
`memory_limit`	Sets the per-process or per-request maximum memory size limit for the specified batch queue.
`mpp_memory_limit`	Sets the per-process or per-request MPP application PE memory size requirements for the specified batch queue.
`mpp_pe_limit`	Sets the MPP processing element (PE) limit for a batch queue against which the per-request MPP PE limit for a request is compared.
`mpp_time_limit`	Sets the CRAY T3D MPP per-process or per-request wall-clock residency time limit or sets the CRAY T3E MPP per-process or per-request CPU time limit for application PEs (multi-PE applications) for the subsequent acceptance of a batch request into the named batch queue.
`permfile_limit`	Sets the per-process or per-request maximum permanent file size limit (in bytes) for the specified batch queue. The per-process request limit is the maximum size of files created by all processes on any type of directory. The

	per-request limit is the maximum amount of disk space used by all processes that make up an NQS request.
`quickfile_limit`	Sets the per-request maximum quickfile space for a batch queue against which the per-request quickfile space limit for a request is compared.
`shm_limit`	Sets the maximum per-request shared memory size limit for a batch queue against which the per-request shared memory size limit for a request is compared.
`shm_segment`	Sets the maximum per-request shared memory segment limit for a batch queue against which the per-request shared memory segment limit for a request is compared.
`stack_limit`	Sets the per-process maximum size (in bytes) for a stack segment in the specified batch queue. This defines how far a program's stack segment may be extended automatically by the system.
`tape_limit`	Sets the per-request maximum tape devices of the specified group for the specified batch queue.
`tempfile_limit`	Sets the per-request maximum temporary file space for an NQS batch queue; not enforced on Cray NQS but available for compatibility with other versions of NQS.
`working_set_limit`	Sets the maximum size (in bytes) of the working set of a process in the specified batch queue. This imposes a limit on the amount of physical memory to be given to a process; if memory is limited, the system may take memory from processes that are exceeding their declared working set size.

5.11.1 Displaying Per-process and Per-request Limits

To check the per-process and per-request limits assigned to a queue, use the `cqstatl -f` or `qstat -f` command; for example:

```
cqstatl -f queuename
```

The limits appear under the characters `<REQUEST LIMITS>`. There is one column for per-process limits and one for per-request limits. See the next section for a partial example, or Section 6.4.4, page 153, for a full example display.

5.11.2 Example of Setting Per-process and Per-request Limits

The following `qmgr` commands set per-process limits for CPU time, permanent file space, and memory:

```
set per_process cpu_limit = 10:0 bqueue15
set per_process permfile_limit = 1Mb bqueue15
```

When you set the limits to 10 minutes, as in this example, that value appears in the `cqstatl` or `qstat` command display as 600 seconds.

The following `qmgr` commands set per-request limits for CPU time and permanent file size:

```
set per_request cpu_limit = 10:20 bqueue15
set per_request permfile_limit = 1Mb bqueue15
```

These commands result in the following `CPU Time Limit` and `Memory Size` fields in the `cqstatl` output:

```
<RESOURCES>
                                PROCESS LIMIT   REQUEST LIMIT
        CPU Time Limit              <600sec>        <620sec>
        Memory Size                 <256mw>         <256mw>...
```

Some parameters are not enforced if the operating system does not support the feature; see Table 4, page 77, for a list of limit enforcements.

5.11.3 How Queue Limits Affect Requests

Request limits are set to the lower of the user-requested limit or the queue limit you set when you created the queue. If a user does not specify a limit, the request inherits the limit set on the queue.

For example, if a user specifies a per-request memory limit, NQS tries to send it to a queue that allows the specified limit. If the same user does not specify a per-process memory limit, the request inherits the per-process memory limit of the queue to which it is sent (even though it was not specified on the command line).

5.11.4 Enabling per Request Limits on Memory Size, CPU Time, and Number of Processors on IRIX Systems

Per request limits on memory size, CPU time, and number of processors must be enabled on IRIX systems. These limits are not turned on by default. To enable checking of these limits, add `NQE_NQS_JL_INTERVAL =`*"seconds"* to the `nqeinfo`(5) file. The *"seconds"* value specifies the number of seconds between limits checking operations. Values of 15 to 120 seconds are recommend, if limits are enabled. Low interval values produce more accurate checking. High interval values generate less overhead.

5.12 Defining Job Scheduling Parameters

To set the weighting factors that will determine the intraqueue priority for runnable requests, use the following `qmgr` commands:

```
set sched_factor share
set sched_factor cpu
set sched_factor memory
set sched_factor time
set sched_factor mpp_cpu
set sched_factor mpp_pe
set sched_factor user_priority
```

All requests in a specific queue are ranked according to four criteria; the following weighting factors determine the importance of each criteria:

Factor	Description
share	Fair share (UNICOS and UNICOS/mk systems only)
cpu	Per-request CPU limit requested
memory	Per-request memory limit requested
time	Time the request has been waiting in the queue for initiation
mpp_cpu	Per-request MPP application PE CPU limit requested
mpp_pe	Per-request number of MPP application PEs requested
user_priority	Per-request user specified priority requested

Each value must be a positive integer less than 1,000,000, including 0. To turn off a weighting factor, specify 0.

The values of the weighting factors are significant only in relation to each other. As the value of one factor increases relative to the others, it has more impact in choosing a request from the suitable requests. If all weighting factors are 0, the request is selected from a specific queue on a first-in, first-out (FIFO) basis.

For all supported platforms except UNICOS/mk, the NQS request scheduler uses the following algorithm to select a request for initiation from all of the eligible requests in a specific queue:

```
intraqueue_priority = ((time_in_queue/ max_time_in_queue) * normalized_time_wt) +
((1.0 - (cpu_time  / max_cpu_time) ) * normalized_cpu_wt) +
((1.0 - (memory_sz / max_memory_sz)) * normalized_mem_wt) +
((1.0 - (share_pri / max_share_pri)) * normalized_share_wt) +
((1.0 - (mpp_cpu_time / max_mpp_cpu_time)) * _mpp_cpu_wt) +
((1.0 - (mpp_pe / max_mpp_pe)) * normalized_mpp_pe_wt) +
((user_priority / max_user_priority) * normalized_user_pri_wt)
```

For UNICOS/mk systems, the NQS request scheduler uses the following algorithm to select a request for initiation from all of the eligible requests in a specific queue:

```
intraqueue_priority=
((time_in_queue/ max_time_in_queue)  * rel_time_wt) +
((1.0 - (cpu_time  / max_cpu_time) ) * rel_cpu_wt) +
((1.0 - (memory_sz / max_memory_sz)) * rel_mem_wt) +
((share_pri / max_share_pri)) * rel_share_wt) +
((1.0 - (mpp_cpu_time / max_mpp_cpu_time)) * rel_mpp_cpu_wt) +
((1.0 - (mpp_pe / max_mpp_pe)) * rel_mpp_pe_wt) +
((user_priority / max_user_priority) * rel_user_pri_wt
```

Each of the job-scheduling resources is scaled from 0 to 1 relative to the other queued jobs. The scheduling weighting factors are also normalized. The normalized *share_wt*, *time_wt*, *cpu_wt*, and *mem_wt* parameters are determined by the `qmgr` command `set sched_factor`. They determine the amount of influence of the fair-share usage, time-in-queue, total requested CPU time, and total requested memory values, respectively, on the computation of the intraqueue priority. The algorithm is designed so that, when all parameters are set to the same value, all of the weighting factors will have an equal effect. The *share_wt* value is valid only for UNICOS and UNICOS/mk systems.

The *cpu_requested* and the *mem_requested* values are the per-request CPU and memory limits from the batch request. These values and the *time-in-queue* value are from the batch request structure. The *share_priority* value is the fair-share usage value and is valid only for UNICOS and UNICOS/mk systems. On UNICOS systems, the fair-share usage value is obtained by accessing the lnode information through the `limits`(2) system call if the user is active at that time; otherwise, the UDB is used to obtain the value. On UNICOS/mk systems, the *share_priority* is obtained from the political scheduling daemon.

On UNICOS/mk or IRIX systems, the number of requested application PEs (*mpp_pe*) and the application PE CPU time (*max_mpp_pe*) weighting factors allow you to further tune the NQS job initiation (intraqueue) priority formula.

If a user specifies a priority by using the `General Options` selection of the `Configure` menu on the NQE GUI `Submit` window, the priority is displayed by the `cqstatl` or `qstat` command while the request resides in a pipe queue, and it continues to be in the request queue if the request is routed to another machine. The value specified by the user is not used in the calculation of the intraqueue priority; it is still valid on NQE systems because it allows compatibility with public-domain versions of NQS.

For UNICOS systems, the `qsub -p` *priority* and `cqsub -p` *priority* commands specify the user-requested intraqueue priority for all systems or the Unified Resource Manager (URM) priority increment for UNICOS systems running URM. The *priority* is an integer between 0 and 63. If you do not use this option, the default is 1.

If you are running URM, the *priority* is passed to URM during request registration. URM adds this value as an increment value to the priority that it calculates for the request.

To change the intraqueue priority of queued job requests, use the `qmgr schedule request` commands. Scheduling a request changes the position of a request within a queue. The request must still meet the global, complex, queue, and user limits criteria before it is initiated, except when it is specified in the `schedule request now` command. Requests in queues that have a higher interqueue priority usually are initiated before requests in lower-priority queues, regardless of whether they have been scheduled.

user_priority is the scheduling weighting factor for user specified priority.

In order to get the previously supported behavior of the `qsub -p` command user specified priority for job initiation scheduling, you should set the *user_priority* weighting factor to nonzero and set all of the other weighting factors to 0. Then, only user specified priority will be used in determining a job's intraqueue priority for job initiation. The `qmgr modify request` command can still be used to modify a job's user priority.

The user specified priority is displayed in request full status displays of the `qstat`(1), `cqstatl`(1), and the NQE GUI. The column name displays as follows:

```
User Priority/URM Priority Increment:   22
```

The `qsub -p` submission option is used for both user priority and URM priority. If NQS is configured for URM job scheduling, then any `-p` submission option is handled as an URM minirank priority increment by the `nqsdaemon`. Otherwise, the `-p` submission option is handled as a user specified priority, which is used with the NQS job initiation formula when calculating intraqueue priority for the job. Either NQS or URM is making job ordering decisions, but not both at the same time.

5.12.1 Example Commands for Scheduler Weighting Factors

Perhaps the most effective way to use intraqueue priority is to set the scheduling factors so a request is kept waiting in relation to how long it will take to run.

For example, if there are two requests with a disparate need for CPU time, they can greatly affect each other. If request `big` needs 900 seconds of CPU time and request `small` needs only 10 seconds, letting `big` run before `small` is significant for the user who submits `small`. If `big` will take 900 seconds to run, the effect on `big` of letting `small` run first is about 2% of `big`'s expected run time. However, if `big` runs before `small`, the effect on `small` is about 6000% of `small`'s expected run time. Thus, it makes more sense to give `small` the higher priority.

However, it is also significant how long a request has been waiting to run. If `big` has been set aside in favor of smaller requests for several days, it too suffers a disproportionate waiting time.

For UNICOS and UNICOS/mk systems, you can set a job-scheduling factor that seeks to schedule according to the considerations just described. You can schedule by the requested CPU time and the time the request has been waiting in the queue. The following settings for the `set sched_factor` commands equally weight the amount of time the request has been waiting and the amount of CPU time requested (the `qmgr` command `set sched_factor share` is valid only on UNICOS systems):

```
set sched_factor share = 0
set sched_factor cpu = 100
set sched_factor memory = 0
set sched_factor time = 100
set sched_factor mpp cpu = 0
set sched_factor mpp pe = 0
set sched_factor user specified priority = 0
```

The value of 100 is not significant; what is significant is that CPU time and time waiting in the queue are equal.

In this scenario, the more CPU time the request needs, the lower its scheduling priority. After it waits longer than its requested CPU time, it has a higher scheduling priority. This stops a large request from being excluded indefinitely by many small requests.

5.12.2 Example Display of Scheduler Weighting Factors

The following example shows the `qmgr` display from a `show parameters` command for a machine that has set a strategy that equally weights requested CPU time and time waiting in a queue:

```
Job Initiation Weighting Factors:
   Fair Share is not enabled.
   Fair Share Priority     = 0
   Requested CPU Time      = 1
   Requested Memory        = 0
   Time-in-Queue           = 1
   Requested MPP CPU Time  = 0
   Requested MPP PEs       = 0
   User Specified Priority = 0
```

5.13 Defining Political Scheduling Daemon Threshold

NQS uses various user, queue, complex and global configurable limits for CRAY T3E application PEs when deciding whether a particular job should be initiated. Since this information is not directly related to the actual availability of PEs on a CRAY T3E system, NQS can initiate jobs which then wait in a Global Resource Manager (GRM) queue, waiting for sufficient application PEs to become available.

You can configure NQS to query the CRAY T3E political scheduling daemon (`psched`) to receive the current size of the maximum contiguous block of application PEs. A threshold to be used before contacting the `psched` daemon is configurable within the `nqeinfo`(5) file. This new variable is `NQE_PSCHED_THRESHOLD`. The valid values for this variable are as follows:

- -1 (default value)
- 0 through 100

This value represents a percentage amount. The default of -1 disables the checking of the application PE size. This variable is not in the default `nqeinfo` file. You can use the `nqeconfig`(8) command to add this variable to the `nqeinfo` file.

The threshold percentage is applied against the number of application PEs that NQS thinks are available, using NQS requested PE counts of running jobs and the NQS global PE limit. If a job requests that threshold amount or a number of application PEs greater than the threshold amount, then the `nsqdaemon`(8) calls the `psched` daemon to get the current maximum contiguous application PE size. If the job's requested application PEs are at or below this size, the job is initiated. Otherwise, the job is not initiated and the job's status display substatus is set to `mp`. The `mp` substatus indicates that the job was not initiated because there were not enough application PEs available.

Here is an example given this scenario:

The `NQE_PSCHED_THRESHOLD` variable is added to the `nqeinfo`(5) file and is set to 80.

The NQS global application PE limit is 240.

Currently running jobs submitted through NQS have requested 200 application PEs.

NQS assumes that 40 applications PEs are still available.

The application PE space is fragmented and only 37 application PEs are available in a contiguous block.

The next job selected to be initiated requests 36 application PEs. The 80 percent threshold is applied against the 40 PEs that NQS thinks are available. The job is asking for application PEs above the threshold of 32 PEs, so the `psched` daemon is contacted. The returned application PE contiguous job size of 37 allows this job to be initiated.

If the job had requested 38 application PEs, the job would not be initiated. A status display for this job would show a substatus of `mp`, indicating that insufficient applications PEs were available to run the job.

If the `nqsdaemon` is unable to get application PE size information from the `psched` daemon for any reason, the job is not held back but is initiated. Messages are written to the NQS logfile describing the error.

5.14 Unified Resource Manager (URM) for UNICOS Systems

The UNICOS Unified Resource Manager (URM) is a job scheduler that provides a high-level method of controlling the allocation of system resources to run jobs. When you enable URM scheduling within NQS, NQS registers certain jobs with URM. Which jobs are registered depends upon which job-scheduling type the `set job scheduling` command has specified. The jobs registered with URM are in a `qmgr` scheduling pool and show, in the `qstat` display, a major status of `Q` and a substatus of `us`.

URM advises NQS when to initiate jobs that have been placed in the scheduling pool. When URM scheduling is in effect, the intraqueue priority has no meaning and is displayed as `---`. All `qmgr schedule` commands, except `qmgr schedule now`, have no effect when URM is recommending which jobs to initiate. The `qmgr schedule now` command initiates a job immediately without consulting URM as to whether the current machine load can tolerate the submission.

For more information about URM, see *UNICOS Resource Administration*, publication SG–2302.

5.15 Miser Scheduler for IRIX Systems

The IRIX 6.5 release introduces a new scheduler called the Miser scheduler. The Miser scheduler (also referred to as Miser) is a predictive scheduler. As a predictive scheduler, Miser evaluates the number of CPUs and amount of memory a batch job will require. Using this information, Miser consults a schedule that it maintains to track the utilization of CPU and memory resources on the system. Miser determines the time when a batch job can be started and have its requirement for CPU and memory usage met. The batch job is then scheduled to begin execution based on this time.

Miser is beneficial because it reserves the resources that a batch job requests. As a result, when the batch job begins to execute, it does not need to compete with other processes for CPU and memory resources. A disadvantage of using Miser is that a job may have to wait for its reservation period before it can begin executing. Additionally, batch jobs must provide the Miser scheduler with a list of resources that they will require. Currently, those resources are the number of CPUs, the amount of memory required, and the length of execution time required.

For a request to be successfully submitted to the Miser scheduler on an NQE execution host, the following criteria must be met:

- The job scheduling class must equal `Miser Normal`. For more information on `Miser Normal`, see Section 5.15.1, page 90.
- The request specifies a Miser resource reservation option. For more information on the Miser resource reservation option, see the *NQE User's Guide*, publication SG–2148.
- The destination batch queue has been directed to forward requests to the Miser scheduler. For more information on directing batch queues to forward requests to the Miser scheduler, see Section 5.15.2, page 90.

- Miser is running on the execution host.
- The Miser queue exists and is accessible on the execution host .
- The request can be scheduled to begin execution before the scheduling window expires. The schedule for a request will depend upon the resources requests (CPU, memory, and time). For more information, see Section 5.15.2, page 90.

5.15.1 Setting the Job Scheduling Type

A new job scheduling type has been created to allow NQS to inter-operate with the Miser scheduler on IRIX systems. The new scheduling type is `Miser Normal`. The `Miser Normal` scheduling type is derived from the `Nqs Normal` scheduling type. Under `Miser Normal` scheduling, the order of the batch jobs in the NQS queues is determined by the queue ordering algorithms that operate under `Nqs Normal` scheduling.

The difference between `Miser Normal` and `Nqs Normal` scheduling is realized when a job is being analyzed to determine if it should be executed. During this analysis, the request is scanned to determine if Miser resource options have been specified. If Miser options are specified, the NQS batch queue is directed to forward requests to the Miser scheduler and NQS queries the Miser scheduler. If no Miser options are specified, the request is started as a normal NQS request (the request will be processed by the operating system default scheduler).

When querying the Miser scheduler, NQS tries to determine if the request can be started by Miser, given the resources requested, before the defined scheduling window expires. If the request cannot be scheduled, NQS continues to hold the request in the batch queue.

For more information on `Miser Normal` scheduling and for the syntax to set the job scheduling type, see the `qmgr`(8) man page.

5.15.2 Directing Batch Queues to Forward Requests to the Miser Scheduler

To configure NQS to operate with the IRIX Miser scheduler you must define the batch queue to indicate that it is capable of forwarding requests to a Miser scheduling queue. You can use the `qmgr`(8) command to set the batch queue to forward requests to the Miser scheduling queue as follows:

```
qmgr> set miser [batch queue] [miser queue] [H:M.S|INT[s|m|h]
```

The options for the `qmgr`(8) command are as follows:

`batch queue`	Name of the NQS batch queue
`miser queue`	Name of the Miser scheduler resource queue
`H:M.S|INT[s|m|h`	Time for the scheduling window defined in hours: minutes.seconds or an integer value with the suffix: `s` for seconds, `m` for minutes, and `h` for hours.

For the `qmgr` command to execute successfully, the batch queue must exist, Miser must be running, and the Miser resource queue must exist.

The scheduling window determines how long a request can wait in the Miser scheduler resource queue before it begins executing; that is, before its resource reservation period arrives. This scheduling window can be used to control the job mix that is forwarded to the Miser resource queue. For example, three NQS batch queues might be defined as follows:

- One queue is defined for jobs that require 32 or more CPUs.
- A second queue may be defined for jobs that require between 8 and 32 CPUs.
- A third queue may be defined for jobs that require less than 8 CPUs.

All of these batch queues can be designated to send requests to the same Miser resource queue.

You can define the scheduling windows so that the batch queues that forward requests that require larger amounts of resources are given a larger scheduling window. You can also define scheduling windows so that batch queues that require fewer resources are given smaller scheduling windows. Defining the scheduling windows in this manner results in a configuration where requests that are lightweight are throttled back so that they do not interfere with the scheduling of requests that require more resources.

An example of the `set miser` command is as follows:

```
qmgr> set miser batnam1 chemistry 20m
```

This example shows that the NQS batch queue `batnam1` will forward requests to the Miser resource queue `chemistry`. The scheduling window specified is 20 minutes. When NQS attempts to execute a request in queue `batnam1`, it will query the Miser scheduler to determine when the request will be given the resources it has requested using the `qsub -X` command. If the query indicates that the request will begin executing before the scheduling window expires, the request is submitted to Miser. Otherwise, NQS will continue to queue the request.

To remove the relationship between a NQS batch queue and Miser run queue, subsitute the keyword `none` for the Miser resource queue name. The value for the scheduling window is irrelevant in this case. The following example shows how you can remove the relationship between the NQS batch queue `batnam1` and the Miser resource queue `chemistry`:

```
qmgr> set miser batnam1 none 0
```

5.16 NQS Periodic Checkpointing

On UNICOS, UNICOS/mk, and IRIX systems, NQS periodic checkpointing provides transparent automatic checkpointing of NQS requests. The checkpoint files restart NQS requests from the time the checkpoint file was created. To checkpoint the requests at intervals that are site-controlled, use the `qmgr set periodic_checkpoint` commands. For more information, see the `qmgr`(8) man page.

Periodic checkpointing initiates a checkpoint for all NQS requests based on the amount of CPU time that is used or the amount of wall-clock time that has elapsed. An NQS administrator can tailor the following periodic checkpoint options:

- Enable and disable periodic checkpointing for all NQS requests
- Enable and disable periodic checkpoint intervals based on the CPU time
- Enable and disable periodic checkpoint intervals based on the wall-clock time
- Exclude short running requests
- Exclude large memory requests
- Exclude large secondary data segments (SDS) requests (UNICOS systems only)
- Set an interval for periodic checkpoints based on the CPU and wall-clock time
- Set the maximum number of concurrent periodic checkpoints
- Define an interval to check eligible requests

Users can enable and disable periodic checkpointing during the life of a job by using the `qalter` command in jobs. For more information, see the `qalter`(1) man page.

5.16.1 Periodic Checkpoint File Restrictions

The periodic checkpoint/restart file cannot be a network file system (NFS) file, but a process with open NFS files can be checkpointed and restarted.

If a process is checkpointed with an open NFS file, and the file is on a file system that was manually mounted, that file system also must be manually mounted when the process is restarted; otherwise, the restart will fail.

5.16.2 Periodic Checkpoint Modes

You can set your periodic checkpoint environment to one of the following modes:

- Compatibility mode (default)
- User-initiated mode
- CPU time mode
- Wall-clock mode
- CPU and wall-clock mode

This section describes how to set each periodic checkpoint mode of operation by using the NQS `qmgr`(8) and `qalter`(1) commands.

5.16.2.1 Compatibility Mode

To set your periodic checkpoint environment to compatibility mode, keep the periodic checkpointing turned off. This is the default environment for periodic checkpoint. To disable periodic checkpointing, use the following command:

```
set periodic_checkpoint status off
```

If the `qalter`(1) command is used in a request, the settings are retained while periodic checkpointing is disabled and used if periodic checkpointing is enabled later. The jobs are checkpointed at NQS shutdown.

5.16.2.2 User-initiated Mode

To set your periodic checkpoint environment to user-initiated mode, you must enable periodic checkpointing with the CPU and wall-clock intervals disabled. This mode periodically checkpoints a request that contains `qalter`(1) commands to enable checkpointing for the request. The following example shows how you can use periodic checkpoint commands to set the user-initiated mode:

```
set periodic_checkpoint status on
set periodic_checkpoint time_status off
set periodic_checkpoint cpu_status off
set periodic_checkpoint cpu_interval = 15
set periodic_checkpoint time_interval = 90
set periodic_checkpoint max_mem_limit = 64MW
set periodic_checkpoint max_sds_limit = 48MW
set periodic_checkpoint min_cpu_limit = 600
set periodic_checkpoint scan_interval = 60
set periodic_checkpoint concurrent_checkpoints = 1
```

In this example, individual requests can be periodically checkpointed. The NQS system status is enabled, but the wall-clock and CPU time status are disabled. NQS lets requests that were modified by the `qalter`(1) command be enabled for periodic checkpointing. If a `qalter` command enables the CPU-based periodic checkpointing but does not define an interval, the request receives the default of 15 minutes. If a `qalter` command enables the wall-clock based periodic checkpointing but does not define an interval, the request receives the system default value of 90 minutes.

If a request has a limit of more than 64 Mwords of memory, more than 48 Mwords of SDS space, or has a CPU time limit of less than 600 seconds, it is excluded from periodic checkpointing. NQS scans requests every 60 seconds to determine whether a running request is eligible for periodic checkpointing. Only one periodic checkpoint request can be run at a time. The jobs are checkpointed at NQS shutdown.

5.16.2.3 CPU Time Mode

To set your periodic checkpoint environment to CPU time mode, you must enable periodic checkpointing to create the checkpoint files after a specific amount of CPU time has been used by the request.

On IRIX systems, job limit checking must be enable per CPU time mode periodic checkpoint.

The following example shows how you can use the periodic checkpoint commands to set the CPU time mode:

```
set periodic_checkpoint status on
set periodic_checkpoint time_status off
set periodic_checkpoint cpu_status on
set periodic_checkpoint cpu_interval = 30
set periodic_checkpoint time_interval = 60
set periodic_checkpoint max_mem_limit = 32MW
set periodic_checkpoint max_sds_limit = 32MW
set periodic_checkpoint min_cpu_limit = 900
set periodic_checkpoint scan_interval = 60
set periodic_checkpoint concurrent_checkpoints = 1
```

In this example, NQS system-wide periodic checkpointing is enabled, and CPU-time-based checkpointing is enabled. The wall-clock-time checkpointing is disabled. NQS checkpoints requests every 30 CPU minutes.

If a request has a limit of more than 32 Mwords of memory or more than 32 Mwords of SDS space, or has a CPU time limit of less than 900 seconds, it is excluded from periodic checkpointing. NQS scans requests every 60 seconds to determine whether a running request is eligible for periodic checkpointing. No more than one periodic checkpoint request can be running. The jobs are checkpointed at NQS shutdown.

5.16.2.4 Wall-clock Mode

To set your periodic checkpoint environment to wall-clock mode, you must enable periodic checkpointing to create the checkpoint files after a specified amount of wall-clock time has elapsed for a request.

The following example shows how you can use the periodic checkpointing commands to set the wall-clock mode:

```
set periodic_checkpoint status on
set periodic_checkpoint time_status on
set periodic_checkpoint cpu_status off
set periodic_checkpoint cpu_interval = 30
set periodic_checkpoint time_interval = 60
set periodic_checkpoint max_mem_limit = 64MW
set periodic_checkpoint max_sds_limit = 64MW
set periodic_checkpoint min_cpu_limit = 900
set periodic_checkpoint scan_interval = 120
set periodic_checkpoint concurrent_checkpoints = 2
```

In this example, NQS system-wide periodic checkpointing is enabled, and checkpointing based on wall-clock time is enabled. The CPU-time-based checkpointing is disabled. NQS checkpoints requests every 60 wall-clock minutes.

If a request has a limit of more than 64 Mwords of memory or more than 64 Mwords of SDS space, or has a CPU time limit of less than 900 seconds, it is excluded from periodic checkpointing. NQS scans requests every 120 seconds to determine whether a running request is eligible for periodic checkpointing. Two periodic checkpoint requests can run simultaneously. The jobs are checkpointed at NQS shutdown.

5.16.2.5 CPU and Wall-clock Mode

To set your periodic checkpoint environment to CPU and wall-clock mode, you must enable periodic checkpointing to create the checkpoint files after a specified amount of CPU or wall-clock time has elapsed for a request, whichever occurs first. The following example shows how you can use the periodic checkpoint commands to set the CPU and wall-clock mode:

```
set periodic_checkpoint status on
set periodic_checkpoint time_status on
set periodic_checkpoint cpu_status on
set periodic_checkpoint cpu_interval = 20
set periodic_checkpoint time_interval = 45
set periodic_checkpoint max_mem_limit = 64MW
set periodic_checkpoint max_sds_limit = 64MW
set periodic_checkpoint min_cpu_limit = 1200
set periodic_checkpoint scan_interval = 90
set periodic_checkpoint concurrent_checkpoints = 3
```

In this example, NQS system-wide periodic checkpointing, CPU-time-based checkpointing, and wall-clock-time-based checkpointing are all enabled. NQS checkpoints requests every 45 wall-clock minutes or every 20 CPU minutes, whichever occurs first.

If a request has a limit of more than 64 Mwords of memory or more than 64 Mwords of SDS space, or has a CPU time limit of less than 1200 seconds, it is excluded from periodic checkpointing. NQS scans requests every 90 seconds to determine whether a running request is eligible for periodic checkpointing. Three periodic checkpoint requests can run simultaneously. The jobs are checkpointed at NQS shutdown.

5.16.3 Periodic Checkpoint Behavior in NQS

The following sections describe how periodic checkpointing affects NQS events.

5.16.3.1 NQS System Shutdown

Periodic checkpoints are not initiated during a system shutdown. The checkpoints that are in progress continue to run until they are completed.

5.16.3.2 Request Completion

No checkpointing occurs when a request is terminating. However, a checkpoint event can be initiated and a request can terminate during the grace period. In this case, NQS request completion cleans up the checkpoint end. If this request will be requeued, the previous valid checkpoint file is retained.

5.16.3.3 `qchkpnt` Command

A `qchkpnt`(1) command is similar to a periodic checkpoint event. If another checkpoint event is in progress, a `qchkpnt` command is rejected. A `qchkpnt` event is counted toward a periodic checkpoint event, and the timers are reset for the next periodic event after a `qchkpnt` completes.

5.16.3.4 Abort

A periodic checkpoint that is in progress during an abort event is terminated, and the checkpoint file is deleted.

5.16.3.5 Modify Request

If you modify a request, its CPU or memory limit can change. This change can make the request ineligible for periodic checkpointing if its CPU time is set to a value less than the value of the `set periodic_checkpoint min_cpu_limit` command, its memory limit is set to a value greater than the value of the `set periodic_checkpoint max_mem_limit` command, or its SDS request limit is set to a value greater than the value of the `set periodic_checkpoint max_sds_limit` command.

5.16.3.6 Hold Request

If a request is being held, it will be checkpointed, terminated, and requeued. If another checkpoint event is in progress, the requests that are being held are rejected. After a request has been held, it is no longer running or eligible for periodic checkpointing.

5.16.3.7 Preempt Request

If a request is being preempted or has completed preemption, no checkpoint events for that request can occur. If a request is currently in a preempted state or in the process of being preempted, no periodic checkpointing is tried.

5.16.3.8 Release Request

If a request is released, it requeues a previously held request and makes it eligible for initiation. The request is started from the last checkpoint file that was created. Any changes that are related to periodic checkpointing are retained, and those values are used when the request is spawned.

5.16.3.9 Rerun Request

If a request is rerun, it is terminated and requeued, and any checkpoint files are deleted. The request is rerun as though it were a new request. Periodic checkpointing is not affected. If the `qalter`(1) command changed any periodic checkpoint options, they revert back to the system-wide default values that `qmgr`(8) defined.

5.16.3.10 Restore Request

If a request is restored, it resumes a request that was previously preempted and makes it eligible for execution. If changes related to periodic checkpointing were made while the request was preempted, they are retained, and those values are used when the request begins to execute.

5.16.3.11 Resume Request

If a request is resumed, it resumes a request that was previously suspended and makes it eligible for execution. If changes related to periodic checkpointing were made while the request was suspended, they are retained, and those values are used when the request begins to execute.

5.16.3.12 Suspend Request

If a request is being suspended or has completed suspension, checkpoint events for a request cannot occur. If a request is currently in a suspended state or in the process of being suspended, no periodic checkpoint is tried.

5.17 Setting the Log File

NQS maintains a log file in which time-stamped messages of NQS activity are recorded. These messages are in text format. To define the file specification of the log file, use the following command:

`set log_file` *name*

The *name* specified may be either an absolute path name or a path name relative to the spool directory. If you specify the same log file name as the existing log file, new information is appended to that file.

To truncate the current log file, use the following `qmgr` command:

```
reset logfile
```

The log file is maintained throughout every shutdown and recovery sequence; new messages are appended to the end of the file. If the log file name is changed, the old file is preserved and is then available for further processing.

NQS can be configured to copy the existing log file automatically to a new file and then reset the existing log file to its beginning. This process is called *segmenting* the log file.

To specify the directory where the segmented log file is kept, use the following command:

`set segment directory` *dirname*

The *dirname* is either the absolute specification of the directory in which segmented files are to be placed, or it is relative to the spool directory (as indicated by the value of `NQE_NQS_SPOOL` in the `nqeinfo` file). Segmentation will not take place if *directory* is not set or is set to `none`. You should not specify `/dev/null` as the log file name. The segmented NQS log file names reflect the date and time of the earliest information contained in the segment.

To set the log file segmentation at NQS start-up, use the following `qmgr` commands:

```
set segment on_init on
set segment on_init off
```

To segment the log file when it reaches a certain size or age, use the following `qmgr` commands:

```
set segment size = bytes
set segment time_interval = minutes
```

To control the detail level of the information recorded in the log file, use the following `qmgr` commands:

```
set message_types on event_type(s)
set message_types off event_type(s)
```

Warning, fatal, and unrecoverable messages are always recorded in the log file. You can turn off informational, efficiency, caution, and trace messages for specific types of events. The following is a list of the event types; the brackets indicate the portion of the command that does not need to be typed when it is entered:

`al[l]` (except flow)	`db_r[eads]`	`op[er]`	`rec[overy]`
`ac[ccounting]`	`db_w[rites]`	`ou[tput]`	`req[uest]`
`ch[eckpoint]`	`f[low]`	`packet_c[ontents]`	`rou[uting]`
`com[mand_flow]`	`network_m[isc]`	`packet_f[low]`	`s[cheduling]`
`con[fig]`	`network_r[eads]`	`proto_c[ontents]`	`user1`
`db_m[isc]`	`network_w[rites]`	`proto_f[low]`	`user2`
			`user3`
			`user4`
			`user5`

Note: Use the `set message_types on flow` command only for a short time interval to help with problem analysis because it can cause severe performance degradation.

To control the amount of information in the header of each log message, use the following `qmgr` commands:

```
set message_header long
set message_header short
```

Long message headers can help with quicker problem resolution.

To control the level of detail of information in the log file, set the debug level through the `qmgr` command `set debug`. For more information, see Section 5.18, page 102.

5.17.1 Displaying Information on the Current Log File

To display information about the current log file, use the following `qmgr` command:

```
show parameters
```

The settings are shown next to the `Log_file`, `MESSAGE_Header`, `MESSAGE_Types`, and various `SEgment` display entries. See Section 6.4.1, page 142, for an example display.

5.17.2 Log File Command Examples

To set the log file name, the message header information level, the recorded message types, and the segmentation interval for the log file (the file is relative to the NQS `spool` directory), use the following `qmgr` commands:

```
set log_file active.log
set message_header long
set message_types on all
set segment on_init on
set segment directory = nqs.log
set segment time_interval = 1440
```

Segmented log files are placed in the `nqs.log` directory under the NQS spool directory, which is set as described in Section 5.17, page 99.

5.18 Setting the Debug Level

To control the level of information recorded in the log file, use the following `qmgr` command to set the debug level for NQS:

`set debug` *level*

Currently, two sets of debug information are supported:

Level	Meaning
0	No debug information is recorded.
1-3	When you specify each higher number (level), the amount of debugging information that is recorded increases, respectively.

To avoid excessive disk usage, set a nonzero debug level only when extra information is required to analyze a problem. The default on an installed NQS system is 0.

To display the current debug level, use the following `qmgr` command:

```
show parameters
```

The setting is shown next to the words `Debug level` on the display. See Section 6.4.1, page 142, for an example display.

5.19 Establishing NQS Accounting Procedures

The accounting information that NQS produces is part of daemon accounting. This information deals with job initiation, termination, and conditions, such as rerun, restart, and preemption. The `set accounting off` and `set accounting on` commands in the `qmgr`(8) subsystem let you specify whether this information will be produced.

The accounting file location is `$NQE_SPOOL/spool/private/root/nqsacct` or `$NQE_NQS_SPOOL/private/root/nqsacct`, where `$NQE_SPOOL` is the spool location defined at installation time. Both variables appear in the `nqeinfo` file.

5.19.1 Enabling and Disabling Accounting

The `qmgr` commands `set accounting off` and `set accounting on` are used to turn NQS daemon accounting off or on.

On UNICOS and UNICOS/mk systems, NQS daemon accounting information is set using the standard UNICOS and UNICOS/mk accounting system. For detailed information about UNICOS accounting, see *UNICOS System Administration*, publication SG–2113 or *UNICOS/mk Resource Administration*, publication SG–2602.

On UNIX systems, NQS simply writes accounting records to the file `private/root/nqsacct` (as indicated by the value of `NQE_NQS_SPOOL` in the `nqeinfo` file) under the NQS spool directory. The file format is ASCII and has the following fields: User ID, Job ID, Sequence Number, Time, Type, Subtype, Host, Request name, and Queue. For job termination records, the user, system, child user, and child system time for NQS shepherd processes are also recorded. The same information can be logged to the NQS log file by using the `qmgr` command `set message_type on accounting`.

The following `qmgr` command is used to turn NQS daemon accounting off:

```
set accounting off
```

The current state of NQS daemon accounting can be displayed by using the `show parameters` command.

The following `qmgr` command is used to turn NQS daemon accounting on:

```
set accounting on
```

Note: This is with respect to NQS. For UNICOS and UNICOS/mk systems, it is up to the UNICOS or UNICOS/mk accounting system to actually enable NQS daemon accounting.

The current state of NQS daemon accounting can be displayed by using the `qmgr` command `show parameters`.

5.19.2 Accounting Records

Accounting records are written to the NQS log and/or the NQS accounting file. Table 5 describes the record types and subtypes. The formats of the Type 1 and Type 2 records that are written follow the table.

Table 5. Accounting records

Type	Subtype	Format	Description
NQ_INFO	NQ_INFO_ACCTON	Type 1	Written when accounting is enabled
NQ_INFO	NQ_INFO_ACCTOFF	Type 1	Written when accounting is disabled
NQ_RECV	NQ_RECV_NEW	Type 1	Written when new request is received
NQ_RECV	NQ_RECV_LOCAL	Type 1	Written when new request is received from local pipe queue
NQ_RECV	NQ_RECV_REMOTE	Type 1	Written when new request is received from remote pipe queue
NQ_INIT	NQ_INIT_START	Type 1	Written when request is started on NQS execution server
NQ_INIT	NQ_INIT_RESTART	Type 1	Written when request is restarted on NQS execution server
NQ_INIT	NQ_INIT_RERUN	Type 1	Written when request is rerun on NQS execution server
NQ_TERM	NQ_TERM_EXIT	Type 2	Written when request terminates normally
NQ_TERM	NQ_TERM_REQUEUE	Type 2	Written when request terminates normally and will be requeued
NQ_TERM	NQ_TERM_PREEMPT	Type 2	Written when request terminates normally because it was preempted
NQ_TERM	NQ_TERM_HOLD	Type 2	Written when request terminates normally because it was held
NQ_TERM	NQ_TERM_OPRERUN	Type 2	Written when request terminates normally because the operator reran the request
NQ_TERM	NQ_TERM_RERUN	Type 2	Written when request terminates normally and will be rerun
NQ_DISP	NQ_DISP_NORM	Type 1	Written when request is disposed normally
NQ_DISP	NQ_DISP_BOOT	Type 1	Written when request cannot be requeued at boot time

Type	Subtype	Format	Description
`NQ_SENT`	`NQ_SENT_INIT`	Type 2	Written when request starts being sent to NQS
`NQ_SENT`	`NQ_SENT_TERM`	Type 2	Written when request completes being sent to NQS
`NQ_SPOOL`	`NQ_SPOOL_INIT`	Type 1	Written when request output starts being transferred
`NQ_SPOOL`	`NQ_SPOOL_TERM`	Type 1	Written when request output completes being transferred

NQE writes two formats of accounting records; the formats are described below. In both formats, all records are ASCII text and fields are separated by spaces.

The Type 1 record format is as follows:

Format	Description
uid	UNIX user ID number for this request
jobid	Process ID of the NQE shepherd for this request
sequence_number	NQE request sequence number
time_of_day	Time of day the record is written; such as, `Tue Oct 10 08:49:17 1995`
type	Record type
subtype	Record subtype
host_name	Host name on which the record was written
request_name	NQE request name
queue_name	NQE queue name
utime	User time of NQE shepherd for this job
stime	System time of NQE shepherd for this job
cutime	Sum of user CPU time for all processes in this job
cstime	Sum of system CPU time for all processes in job

project_id	Project ID for all processes in this job (IRIX systems only)

Note: Total job time = *utime* + *stime* + *cutime* + *cstime*

Total overhead = *utime* + *stime*

Total user time = *cutime* + *cstime*

The Type 2 record format is as follows:

Format	Description
uid	UNIX user ID number for this request
jobid	Process ID of the NQE shepherd for this request
sequence_number	NQE request sequence number
time_of_day	Time of day the record is written; such as, `Tue Oct 10 08:49:17 1995`
type	Record type
subtype	Record subtype
host_name	Host name on which the record was written
request_name	NQE request name
queue_name	NQE queue name
project_id	Project ID for all processes in this job (IRIX systems only)

The units used for *utime*, *stime*, *cutime*, and *cstime* vary by machine and operating system level. The unit is measured in ticks per second. The unit is defined on each system by the value of `sysconf (_SC_CLK_TCK)`, which can vary by machine and may be run-time alterable.

5.20 Setting the Validation Strategy for Remote Access

To provide security against unauthorized remote access of the local NQS system, you can specify that you want to validate incoming client requests. The type of validation you select is performed for all actions: submitting requests, monitoring requests and queues, and deleting or signaling requests.

The administrator of each NQS server in the group of NQE nodes in the NQE cluster can set a validation type. To avoid confusing users, all NQS servers in the group of NQE nodes in the NQE cluster should use the same validation type. The default validation type for an NQS server when first installed is validation file checking.

You can specify the following validation types for the NQS host:

Validation type	Description
No validation	No validation is performed by NQS. If the name of the user under which a command is issued (the *target user*) is a valid user at the NQS server, the command is successful.
Validation files	A validation file is examined to determine whether there is an entry that allows the user under which the command is issued to access the server. The validation file can be either an `.rhosts` or `.nqshosts` file in the user's home directory at the NQS server. (If you do not use `.rhosts` files at your site, you can use `.nqshosts`). Individual users create these files. **Note:** If a machine name has any aliases, the `.rhosts` files must contain all forms of the machine name.
Password checking	The NQS user is required to supply a password for the user name at the NQS server at which the request is to be run. A password may be no longer than 8 characters. The password is checked only at the NQS server. The password is used for validating the user name under which a user command will be executed (either through the NQE GUI functions or through the following commands: `cqsub`, `cqstatl`, `cqdel`, `qsub`, `qstat`, and `qdel`). The password is encrypted and passed over the network, and NQS checks the password file to ensure that the password is valid.
Password checking and validation files	In this combined method, a user may supply a password when issuing a command as is done in password checking. If a password is supplied, the

local NQS server validates it; if the password is incorrect, NQS rejects the action the client command requested. If a password is not supplied, validation file checking is performed.

Note: For client commands, the user is prompted for a password if the `-P` option is specified on the command line or if the `NQS_PASSWORD_NEEDED` environment variable is set.

5.20.1 User Validation Commands

To set the validation type, use the following `qmgr` commands:

```
set validation [password] [file]
```

The default validation is by `file`. You can specify one or both of the parameters, as follows:

Parameter	Definition
`password`	Password checking only.
`file`	Validation file checking only.
`password file`	Password checking if a password is supplied; otherwise, validation file checking is performed.

To turn off validation, use the following command:

```
set no_validation
```

The following field in the `qmgr` command `show parameters` display indicates the validation policy is by `file`:

```
Validation type = validation files
```

See Section 6.4.1, page 142, for a full example display.

5.20.2 Example of Setting User Validation

To set the method of validation to password checking only, use the following `qmgr` command:

```
set validation password
```

5.21 Defining Other System Parameters

This section describes miscellaneous NQS configuration settings.

5.21.1 Setting Retry Limits

If the NQS system cannot send a request from a local pipe queue to its destinations (for example, because the network connection to the system supporting the destinations has failed), it will try resending the request at regular intervals. To define how often the server will retry sending a request, use two `qmgr` commands, as follows:

- To define the interval (in minutes) between successive attempts to send the request, use the following `qmgr` command:

  `set default destination_retry wait` *interval*

  The initial value for a newly installed NQS system is 5 minutes.

- To avoid continuous retries, use the following `qmgr` command to define the maximum elapsed time (in hours) for which a request can be retried. If this time is exceeded, the request submission fails and a mail message is sent to the user who submitted the request.

  `set default destination_retry time` *time*

  The initial value for a newly installed NQS system is 72 hours.

5.21.1.1 Displaying the Current Retry Limits

The following fields in the `qmgr` command `show parameters` display indicate the values for retry limits:

```
Default destination_retry time = 72 hours
Default destination_retry wait = 5 minutes
```

See Section 6.4.1, page 142, for a full example display.

5.21.1.2 Example of Setting Retry Limits

In this example, you want to retry requests every 10 minutes. However, you want to set a limit of 48 hours on the time during which the retries can occur; after this time, NQS will abandon sending the request and inform the user that the request failed. To do this, you must enter the following `qmgr` commands:

```
set default destination_retry wait 10
set default destination_retry time 48
```

5.21.2 Defining the Sender of NQS Mail

NQS can send different types of mail messages to users. For example, a message can be sent when the request completes execution if the `qsub -me` option was used in the request submission. Messages are also sent when errors occur. The initial setting uses the user name `root` in the `From` field of the message.

To change the name in the `From` field to another name, use the following `qmgr` command:

`set mail` *name*

The *name* argument must be the name of a valid user at the system running the server.

5.21.2.1 Displaying the Current Sender of Mail

The following field in the `show parameters` display indicates that `root` is the sender of NQS mail:

```
Mail account = root
```

See Section 6.4.1, page 142, for a full example display.

5.21.2.2 Example of Setting NQS Mail User

If you have a special user account as the NQS manager, you may want users to identify mail messages from that user account rather than just indicating that they are from `root`. For example, if there is a user called `nqsman`, enter the following `qmgr` command:

```
set mail nqsman
```

Mail then seems to come from a user called `nqsman`.

5.21.3 Locking and Unlocking Daemon Processes

The UNIX system swaps processes in and out of memory as required. Therefore, more processes can be run simultaneously than when all the processes are in memory all of the time. Although this does slightly increase the overhead on the system performance to manage the swapping, it is not usually useful to lock a process into memory.

By default, the NQS daemon `nqsdaemon` is not locked into memory (it can be swapped in and out as required). If you want to lock it into memory, use the following `qmgr` command:

```
lock local_daemon
```

Note: The ability to lock `nqsdaemon` into memory is supported only for systems running a derivative of UNIX System V.

To unlock the process from memory, use the following `qmgr` command:

```
unlock local_daemon
```

5.21.3.1 Displaying the Current Locking State

The following field in the `show parameters` display indicates that the NQS daemon is not currently locked in memory:

```
NQS daemon is not locked into memory.
```

See Section 6.4.1, page 142, for a full example display.

5.22 Using a Script File As Input to the `qmgr` Utility

If you are defining many systems or queues, perform the configuration manually or create a script file of `qmgr` commands to use as input to `qmgr` instead of entering the information at the `qmgr` prompt.

After you complete an initial configuration, use the `qmgr snap` command to generate a script file automatically. This file contains the `qmgr` commands that will generate the current NQS configuration; see Section 5.22.1, page 113 for more information.

To edit the script file, use any standard text editor. The `qmgr` commands in the file are the same as those you would enter at the `qmgr` prompt.

To pipe the file into the `qmgr` utility, use the following command line:

```
cat filename | qmgr
```

Each line in the file is executed as if it were entered at the `qmgr` prompt. Messages are displayed by `qmgr` as usual in response to the commands. If a command produces an error, the error message is displayed, and execution continues to the next line of the file.

To record the output that `qmgr` produces (the size of the output display usually is larger than the size of your screen display), use the following command to redirect the standard output to a file:

`cat` *filename* `| qmgr >` *output-filename*

5.22.1 Creating a `qmgr` Script File of the Current Configuration

The `qmgr` command `snap` creates a file containing the `qmgr` commands that are necessary to redefine the current NQS configuration. This file can then be input to `qmgr` at another NQS system to build a duplicate copy of the configuration there.

The format of the `snap` command is as follows:

`snap [file =` *pathname*`]`

The *pathname* argument is the path name of the file that will receive the `qmgr` commands. If you omit this argument, the `qmgr` commands are written to the file specified by the `qmgr` command `set snapfile`. If no file was specified, the `snap` command fails.

The format of the `set snapfile` command is as follows:

`set snapfile` *pathname*

The *pathname* is relative to the NQS spool directory, or it can be an absolute path name. The current file (if any) is shown on the display produced by the `qmgr` command `show parameters`. The name appears after the characters `snap_file`. The initial value of the file name is `/dev/null`. See Section 6.4.1, page 142, for an example display.

To input a snap file to `qmgr`, use the `cat`(1) command, as follows:

`cat` *pathname* `| qmgr`

The *pathname* is an absolute path name or relative to the current directory.

In the following example, the default file that snap uses is first set to `nqssnap` in the NQS spool directory, and then snap is used to write the file. The `more`(1) command displays the first few lines of the snap file to show what is written to the file.

```
snow% qmgr
Qmgr: set snapfile nqssnap
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
Qmgr: snap
Qmgr: quit
snow% more nqssnap
#
#Create the nqs network information concerning Mid, Service,
#and Aliases.
#
add mid rain rain-hy rain-ec
add output_agent fta rain
add mid squall squall-hy squall-ec
add output_agent fta squall
--More--
```

The script file produced by `snap` contains comments that divide the commands into functional groups.

The next screen shows how this script file can be used as input to `qmgr` at another NQS system:

```
$ cat /nqebase/database/spool/snapfile | qmgr > outfile
```

5.23 Output Validation

The NQS server returns output from requests to the client host or to another host (depending on the location that the user specified in the request's output options) by either writing output to a common file space or using `fta` or `rcp`. The default is `fta` using `ftp` protocol. Return output to the DFS system is restricted to writing output to a common file space or using `fta`; `rcp` does not support DFS.

Output option	Output location
No location specified or *filename* specified	The user's working directory at submission on the client workstation.
host and *filename* specified	Specified server and file name. The directory is the user's home directory on *host*, unless otherwise specified.

user, host and/or domain, and *pathname* specified	Specified user, host, and path name. The *pathname* can be a simple file name or an absolute path. If the *pathname* is a simple file name, the file is placed in the user's home directory on the specified host. The *domain* specifies an FTA domain name that uses network peer-to-peer authorization (NPPA) rather than a password. If users do not want to use NPPA, they can provide a *password*.

NQS tries to return output to the correct user and directory according to what the user specified. The NQS server uses the following methods to determine how output will be returned to the client (listed in order of precedence):

Note: When using the `qsub`(1) command, numbers 2 and 3 do not apply, and the order of precedence is dependent upon the output agents defined.

1. If the output destination host is the local NQS server (`NQS_SERVER`), output is placed locally in the directory from which the request was submitted or in a specified local location.
2. If the output destination host is not the local NQS server, NQS uses FTA. NQS always tries to locate a password for the FTA transfer, which may be embedded in the `cqsub -o` command line or in the user's `.netrc` file, or it may be specified when using the NQE GUI to submit a request. FTA transfers the file according to the rules configured for the *domain* specified. If the user omits *domain*, FTA uses its default domain, which is `inet` unless you have reconfigured FTA. The `inet` domain uses `ftp` protocol and requires a password unless you have configured NPPA. If you have configured NPPA, no password is required.
3. If the FTA transfer fails, NQS tries to send the output using `rcp`. `rcp` uses the `.rhosts` file on the destination host.
4. If all of these methods fail, NQS places the output in the home directory of the user on the execution host.

5.24 NQS and Multilevel Security on UNICOS and UNICOS/mk Systems

This section explains the differences for NQS when used on UNICOS or UNICOS/mk systems that run the multilevel security feature (MLS). It is assumed that you are familiar with the UNICOS or UNICOS/mk documentation about general system administration and the multilevel security feature on the UNICOS or UNICOS/mk system.

Note: If you are running NQE 3.3 on a UNICOS 10.0 system or UNICOS/mk 2.0.2, this section applies to your system since the MLS features are always available at these release levels.

If you are running UNICOS 9.0, this section applies to your system if you are running the MLS feature.

When NQS runs on a UNICOS or UNICOS/mk system that runs the multilevel security feature, the following differences exist:

Note: If the MLS feature on UNICOS or security enhancements on UNICOS/mk are enabled on your system, the job output files are labeled with the job execution label. For jobs that are submitted locally, the return of the job output files may fail if the job submission directory label does not match the job execution label. For example, if a job is submitted from a level 0 directory, and the job is executed at a requested level 2, the job output files cannot be written to the level 0 directory. If the home directory of the UNICOS user under whom the job ran is not a level 2 directory, does not have a wildcard label, or is not a multilevel directory, the job output files cannot be returned to that directory either. The job output files will be stored in the NQS failed directory.

If the MLS feature on UNICOS or security enhancements on UNICOS/mk are enabled on your system and you submitted a job remotely, the Internet Protocol Security Options (IPSO) type and label range that are defined in the network access list (NAL) entry for the remote host affect the job output file return.

- When job output is returned, the output files are labeled at the job execution label for both local and remote destinations.
- When a job is restarted, the session security attributes are reverified; if these attributes are not in a valid range, based on the user's user database (UDB) entry and the socket (or system) security attributes when the job was queued, the job is deleted. If the job is deleted, a message is written to the `syslog` file.
- A label requested by using the `-L` and `-C` options of the `qsub` or `cqsub` command or the NQE GUI must dominate the job submission label.
- Mail is sent at the appropriate job label. Mail is not sent at the `nqsdaemon` label, but at either the job submission or job execution label, as applicable. If the job has not yet been initiated, mail sent to the job owner is sent at the job submission label. If the job has been initiated or has completed

execution, mail is sent at the job execution label. To read this mail, users must have their current active label set appropriately.

- Session security attributes are constrained by the socket security attributes when the job was queued and the user's UDB entry. For local submissions, the system minimum and maximum label range is used. For more information on the socket security attributes, see your UNICOS or UNICOS/mk system administration documentation.
- The workstation access list (WAL) is checked for remote job activity, such as job acceptance, job status, and deleting jobs. If the execution host has a WAL entry for the origin host that restricts access to NQS services for the caller, remote job activity to a remote host can fail. For more information on the WAL, see your UNICOS or UNICOS/mk system administration documentation.
- For sites running the MLS `SECURE_MAC` feature, all NQS user commands are labeled as trusted processes so that they can write to a `syshigh` labeled log file. The NQS user commands write messages to the NQS log file.
- On an upgrade installation from a system not running the multilevel security feature to a system that is running the multilevel security feature or an upgrade installation from UNICOS 9.0 to UNICOS 10.0, the jobs are held and are not initiated at NQS startup. The NQS administrator can then either release the job (which will then run at the job owner's UDB default label) or delete the job.

For information on implementing the multilevel security feature, see your UNICOS or UNICOS/mk system administration documentation.

5.24.1 NQS Multilevel Directories (MLDs)

If you are upgrading your NQS configuration to use multilevel directories, you must convert your wildcard directories to multilevel directories (MLDs). To convert between existing wildcard-labeled directories and MLDs, use the `cvtmldir` command. (For more information, see the `cvtmldir`(8) man page.) This conversion must be performed in single-user mode after installing NQE.

The `cvtmldir` command requires absolute path names; you must rename the affected spool directories before calling `cvtmldir` to convert from wildcard directories to multilevel directories. In the following examples, the `NQS_SPOOL/private/root/control` directory is being converted, but you also must convert the following NQS directories, the locations of which are defined in your `nqeinfo` file:

- `NQS_SPOOL/private/requests`
- `NQS_SPOOL/private/root/chkpnt`
- `NQS_SPOOL/private/root/control`
- `NQS_SPOOL/private/root/data`
- `NQS_SPOOL/private/root/failed`
- `NQS_SPOOL/private/root/interproc`
- `NQS_SPOOL/private/root/output`
- `NQS_SPOOL/private/root/reconnect`

To convert from wildcard directories to MLDs, use the following procedure (you must be `root`):

1. Ensure that the `nqsdaemon` is **not** running while the conversion is occurring. For more information on how to shut down NQS, see the `qstop`(8) man page. For more information on `nqsdaemon`, see the `nqsdaemon`(8) man page.
2. Activate the `secadm` category by entering the `setucat secadm` command if needed.
3. Change to the parent directory; in this example, to convert the `NQS_SPOOL/private/root/control` directory, change to the `NQS_SPOOL/private/root` directory.
4. Rename the `control` directory to `control.temp` by entering the following command:

   ```
   /bin/mv ./control ./control.temp
   ```

5. Enter the following command to create a multilevel symbolic link called `NQS_SPOOL/private/root/control`, which points to a directory named `NQS_SPOOL/private/root/control.mld`. The files in the `control.temp` directory are linked or copied, when necessary, into the new `control.mld` directory. The files are not deleted from the `control.temp` directory.

   ```
   /etc/cvtmldir -m $NQS_SPOOL/private/
   root/control.temp $NQS_SPOOL/private/root/control
   ```

6. Enter the following command to create a relative path name for the multilevel symbolic link output, allowing access to the files and directories in `NQS_SPOOL/private/root` while running with the caller's privileges:

   ```
   /etc/unlink NQS_SPOOL/private/root/control
   /bin/ln -m ./control.mld ./control
   ```

You should ensure that the directories were converted successfully. The `control.temp` directory and its files should not be deleted until NQS has been restarted and the jobs are successfully requeued or restarted. Repeat the preceding steps for each NQS directory that is listed earlier in this section.

5.24.2 PALs

Note: PALs are required for UNICOS 10.0 and UNICOS/mk systems but not for UNICOS 9.0 systems.

You must assign the privilege assignment lists (PALs) by running the `privcmd`(8) command. For more information on PALs and the multilevel security feature, see your UNICOS or UNICOS/mk system administration documentation. Multilevel security on a UNICOS or UNICOS/mk system provides a default set of PALs.

To install PALs on each UNICOS or UNICOS/mk multilevel security server, execute the following command:

`/etc/privcmd -b /`*nqebase*`/etc/nqs.db`

5.24.3 Security Policies

This section describes the following multilevel security policies that are available on a UNICOS or UNICOS/mk system, which are supported on NQS.

Note: To determine if these policies are supported on your system, see your UNICOS or UNICOS/mk system administration documentation.

- Mandatory access controls (MACs)
- Discretionary access controls (DACs)
- Identification and authentication (I&A)
- System management
- Auditing

5.24.3.1 MAC Security Policy

Mandatory access controls (MACs) are rules that control access based directly on a comparison of the subject's clearance and the object's classification.

The security policy controls read and write operations to prohibit unauthorized disclosure of any system or user information. The *security policy* is defined as the set of rules and practices by which a system regulates the disclosure of information.

NQS provides enforcement of the MAC rules for status, signal, and delete requests (local and remote). To audit the delete requests, use the `SLG_NQS` audit record for both successful and failed requests. To audit the MAC override for NQS managers and operators, use the `SLG_TRUST` (trusted process) audit record. For more information on auditing, see the description of the multilevel secuity feature in your UNICOS or UNICOS/mk system administration documentation.

The MAC rules for status requests require that the caller's active security label dominate the submission label of a job before information about that job can be displayed (that is, the caller's label must be greater than or equal to the job label).

The MAC rules for signal and delete requests require that the caller's active security label equal the label of the job. If the job is queued, the caller's active label must equal that of the job submission label. If the job is executing, the caller's active label must equal that of the job execution label.

If you want to enforce the MAC rules, set the `NQE_NQS_MAC_COMMAND` configuration parameter to 1 in the `nqeinfo` file before NQS is started; otherwise, MAC checking is not performed. The setting of this parameter does not affect the NQS managers and operators, who can automatically bypass the MAC rules and administer NQS without additional restrictions.

If MAC rules enforcement is enabled, users from a system that is not running the multilevel security feature can check the status of and delete their jobs only if the jobs have a level 0, no compartments label (UNICOS 9.0 systems only). Depending on the network access list (NAL) configuration on the execution host, these users may not be allowed to submit jobs that request a higher label. For more information on the NAL configuration, see your UNICOS or UNICOS/mk system administration documentation and the `spnet`(8) man page.

NQS lets you specify a job execution label by using the NQE GUI or the `-C` and `-L` options of the `qsub` and `cqsub` commands. Descriptions of the execution labels follow, assuming that the user's UDB security attributes allow the specified execution label:

- If no `-C` or `-L` command–line options are specified, or if the job execution label is not specified using the NQE GUI:
  - For a local job, the execution label equals the job submission label.
  - For a remote job, the execution label equals the user's default label if it is within the session bounds (based on the socket minimum and maximum values and the user's UDB minimum and maximum values); if it is not within the session bounds, the execution label equals the session minimum label.
- If the `-C` command–line option is specified, or if the job execution label is not specified using the NQE GUI:
  - For a local job, the execution label equals the job submission level and the requested compartment set.
  - For a remote job, the execution label equals the socket active level and the requested compartment set.
- If the `-L` command–line option is specified, or if the job execution label is specified using the NQE GUI:
  - For a local job, the execution label equals the requested level and the submission compartment set.
  - For a remote job, the execution label equals the requested level and the socket active compartment set.
- If both the `-C` and `-L` command–line options are specified, or if the job execution label is specified using the NQE GUI:
  - For a local job, the execution label equals the requested level and the requested compartment set.
  - For a remote job, the execution label equals the requested level and the requested compartment set.

A requested label must dominate the job submission label. For example, if a user has an active label of level 2 and compartments `sysadm` and `secadm`, and submits a job by using the `qsub -L 4 -C sysadm` command, this requested label does not dominate the submission label. The requested compartment set must be a superset of the submission compartment set. If the user's UDB entry allows authorized compartments of `secadm`, `sysadm`, and `sysops`, the `qsub`

`-L 4 -C sysadm,secadm` command and the `qsub -L 4 -C sysadm,secadm,sysops` command would dominate.

For a remote job, the socket security information is stored during job acceptance in the tail of the control file, as the new record type `s`, and also within the `rawreq` extension structure. If the receiving host is configured for strict B1 compliance, the stored socket minimum level and maximum level are set to the socket active level, and the stored socket valid compartment set is set to the socket active compartment set. This constrains the session security minimum and maximum to the socket active label, and, therefore, also contains the job execution label to the socket active label.

If a remote job is queued when the system is not enabled for multilevel security and the system is then configured to be enabled for multilevel security, the `s` record values are set based on the system security values, and the submission label is set to the user's default label.

If you have installed PALs on your system, any user with an active `secadm`, `sysadm`, `sysops`, or `system` category can start `nqsdaemon`; this procedure is audited. The `qst` PAL privilege text identifies which categories of users can start `nqsdaemon`. The shutdown procedure is not changed; the caller must be an NQS administrator, who does not need an active category. For more information on `nqsdaemon`, see the `nqsdaemon`(8) man page.

The following changes occur in NQS with the MAC security policy:

- The following directories are either labeled as multilevel directories (MLDs) or are wildcard directories as defined in your `nqeinfo` file:
  - `NQS_SPOOL/private/root/control`
  - `NQS_SPOOL/private/root/data`
  - `NQS_SPOOL/private/root/failed`
  - `NQS_SPOOL/private/root/interproc`
  - `NQS_SPOOL/private/root/output`
  - `NQS_SPOOL/private/root/chkpnt`
  - `NQS_SPOOL/private/requests`
  - `NQS_SPOOL/private/reconnect`
- If UNICOS or UNICOS/mk is configured to support the `syslow` and `syshigh` MAC labels, the log file, console file, and daemons file are labeled

system high (`syshigh`); all other NQS database files are labeled system low (`syslow`).

- The client processes access the NQS protocol named pipe through the privilege mechanism (or the pipe is labeled as wildcarded).
- Enforcement of the MAC rules for status and delete operations is configurable; NQS administrators bypass this enforcement.
- The MAC rules for writing to job output files are enforced; NQS administrators bypass this enforcement. For more information on writing messages to job output files, see the `qmsg`(1) man page.
- Output files are labeled at the job execution label.
- Mail is sent at the job label using the following conditions:
  - If the job is queued, mail is sent at the job submission label.
  - If the job has been initiated or has completed, the mail is sent at the job execution label.
- The socket security attributes at job acceptance are used to determine the session security attributes for jobs that are submitted remotely; this constrains the user's UDB security attributes.
- Session security attributes are reverified at job restart; if constraints have changed, the job may be deleted instead of restarted.

5.24.3.2 DAC Security Policy

Discretionary access controls (DACs) are rules that control and limit access to an object, based on an identified individual's need to know and without intervention by a security officer for each individual assignment.

This is accomplished by using standard mode permission bits and an access control list (ACL); the ACL and mode bits allow the owner of a file to control `r/w/x` access to that file. The owner of the file can create or modify an ACL that contains the identifiers and the `r/w/x` permissions for those individuals and groups that will be allowed to access the file.

The mandatory policy restrictions established by the security administrator always govern object access.

For more information on how ACLs are used and for examples that show how to create and maintain ACLs, see the `spacl`(1) man page and your UNICOS or UNICOS/mk system administration documentation.

When DAC is used on a secure system, the following changes occur in NQS:

- You must explicitly add `root` to the queue access lists (this is also true on non-multilevel security systems).
- Any current NQS manager or user who is authorized through the PAL (see Section 5.24.3.4) can add NQS administrators.

5.24.3.3 I&A Security Policy

The identification and authentication (I&A) feature describes the login and password features used on a UNICOS or UNICOS/mk system that is running multilevel security. When I&A is used on a secure system, the following changes occur in NQS:

- Centralized authentication is used to authenticate remote and alternative users (this is also true on UNICOS 9.0 non-multilevel security systems).
- When the NQS validation type is `file`, and the system is configured for strict B1 compliance, the effect on the alternative user is as follows:
  - The remote user must be the same as the local user.
  - The remote host must be in the `/etc/hosts.equiv` file.
  - The remote host must be in the local user's `.rhosts/.nqshosts` file.
  - To provide alternative user capability for NQS when it is configured for file validation on a system that has multilevel security enabled, you must add the `NETW_RCMD_COMPAT` configuration parameter to the `SECURE_NET_OPTIONS` macro in the system `config.h` file before building and installing the kernel.
- The WAL is checked for remote job queuing, remote status, and remote signal/deletion events to see whether the user is allowed access to NQS.

5.24.3.4 System Management

Note: For UNICOS 10.0 and UNICOS/mk systems, sites can run with `PRIV_SU` and PALs or PAL-only Trusted UNICOS or PAL-only Cray ML-Safe.

The ability to separate and define different administrative roles and tasks is part of the security-related system management policies and procedures. This section describes the authorizations required to perform each administrative task or function for each system management mechanism.

- All NQS user commands are labeled as trusted processes because they may write to the `syshigh` labeled log file; this is when `SECURE_MAC` is turned on. The `privcmd`(8) command is used to assign PALs, and MAC and DAC labels.
- When using the super-user mechanism (`PRIV_SU`), only `root` can change the user validation type.
- When using the PAL privilege mechanism, the `chgm` privilege text (for an active `secadm` or `sysadm` category) is used to determine who can add, delete, and set managers, and to change the user validation type. To change the user validation type, the caller must be an NQS manager.

5.24.3.5 Auditing

The site can define the security auditing policy for a UNICOS or UNICOS/mk system on which the multilevel security feature is enabled. The site policy should be determined before the system is up and running, and it should be applied consistently at all times. Consistent and proper use of the the multilevel security auditing features will help ensure site security.

For information on producing the NQS security log records in the multilevel security feature, see your UNICOS or UNICOS/mk system administration documentation.

The audit records specific to NQS are `SLG_NQS` and `SLG_NQSCF`. Whenever an NQS delete request is made, the `SLG_NQS` record is produced. These entries are logged through the `slgentry`(2) system call. Whenever a security-related change is made to the configuration of NQS, the `SLG_NQSCF` record is produced. These entries are logged through the `slgentry`(2) system call.

To enable auditing, you can use either the `/etc/spaudit -e` *record_type* (*record_type* is either `nqs` or `nqscf`) command or the UNICOS Installation/Configuration Menu System (on UNICOS systems) or the system configuration files (on UNICOS/mk systems). For more information on auditing the multilevel security feature on a UNICOS or UNICOS/mk system, see your UNICOS or UNICOS/mk system administration documentation and the `spaudit`(8) man page. The following security-related events are audited by the NQS audit records, or the `SLG_LOGN` or `SLG_TRUST` audit records:

- User authentication success/failure (`SLG_LOGN`)
- Failure to set job process label (`SLG_LOGN`)
- Job deletion (`SLG_NQS`)

- Queue access list insertions and deletions (`SLG_NQSCF`)
- User validation type changes (`SLG_NQSCF`)
- Add, delete, and set managers and operators (`SLG_NQSCF`)
- Attempt to execute `qmgr` subcommands by a nonprivileged user (`SLG_NQSCF`)
- Bypass of the MAC rules enforcement by NQS administrators (`SLG_TRUST`)

5.24.4 NQS Multilevel Security Configuration

The configurable multilevel security features for NQS on UNICOS or UNICOS/mk systems, which are described in Section 5.24.3, page 119, are disabled by default. To enable these features, you must make the following configuration changes.

Note: To determine if these policies are supported on your system, see your UNICOS or UNICOS/mk system administration documentation.

1. Enforce the MAC rules for status and delete requests by setting the `NQE_NQS_MAC_COMMAND` variable to 1 in the `nqeinfo` file.
2. Use MLDs instead of wildcard-labeled spool directories by setting the `NQE_NQS_MAC_DIRECTORY` variable to 1 in the `nqeinfo` file.

   If the NQS spool directories already exist, see Section 5.24.1, page 117, for conversion instructions.

   Note: To execute the `mlmkdir`(8) command, which is used to create MLDs, special authorization is required. To create or convert to MLDs in the NQS spool area, you must be `root` and must have the following additional authorizations, depending on your system configuration: PAL-based (Active `secadm` category) and `PRIV_SU` (No additional privilege).

3. Set the `NQE_NQS_PRIV_FIFO` variable to 1 in the `nqeinfo` file. This action enforces the use of privilege through PALs for client processes, such as `qsub`, by writing over the NQS protocol named pipe and the NQS log pipe.

   Before starting NQS, see *NQE Installation*, publication SG–5236, appendix A, *Preparing a Node to Run NQE*. For more information, see the `privcmd`(8) man page.

Warning: If your system is configured to enforce the `syshigh` and `syslow` security labels, the NQS spool directories and the NQS log file and console file must be on a file system that has `syshigh` and `syslow` in its label range.

On UNICOS systems, ensure that the UNICOS installation tool menu is set as follows:

```
Configure system -> Multilevel security (MLS) configuration ->
System options -> Enforce system high/low security labels -> ON
```

5.24.4.1 Converting to Wildcard Labeled Directories

Note: To use the `mlmkdir`(8) command, which is used to create MLDs, special authorization is required. To create or convert to MLDs in the NQS spool area, you must be `root` and must have the following additional authorizations, depending on your system configuration:

PAL-based	Active `secadm` category
`PRIV_SU`	No additional authorization

To convert from MLDs to wildcard-labeled directories, use the following procedure:

1. Ensure that the `nqsdaemon` is **not** running while the conversion is occurring. For more information on how to shut down NQS, see the `qstop`(8) man page. For more information on `nqsdaemon`, see the `nqsdaemon`(8) man page.

2. As `root`, activate the `secadm` category, if needed, by entering the `setucat secadm` command.

3. Change to the directory that you want to convert; in this example, change to the `NQS_SPOOL/private/root` directory.

4. Remove the multilevel symbolic link control, so that a name can be used with the `cvtmldir`(8) command to create the directory that will be labeled as a wildcard directory by entering the following commands:

   ```
   /bin/ln -m ./control.mld ./control.temp
   /etc/unlink ./control
   ```

   The `control.temp` directory is now a symbolic link pointing to `.control/.mld`.

5. Create a multilevel symbolic link called `NQS_SPOOL/private/root/control`, which points to a directory named `NQS_SPOOL/private/root/control.mld`, by entering the following command:

   ```
   /etc/cvtmldir -w $NQS_SPOOL/private
   /root/control.temp $NQS_SPOOL/private/root/control
   ```

   The files in the `control.mld` directory are linked or copied, when necessary, into the new `control` directory. The files are not deleted from the `control.mld` directory.

6. Set the `NQE_NQS_MAC_DIRECTORY` parameter value to 0 in the `nqeinfo` file. You must restart NQS with these values.

The administrator should ensure that the directories were converted successfully. You should not delete the `control.temp` directory and its files until NQS has been restarted and the jobs are requeued or restarted successfully. You must repeat the preceding steps for each NQS directory that is listed at the beginning of Section 5.24.1, page 117.

5.25 NQS Local Changes for UNICOS and UNICOS/mk Systems

On UNICOS and UNICOS/mk systems, you may make local changes to NQS. For Cray PVP systems, you may make changes to NQS by using user exits and by making source-code modifications. For UNICOS/mk systems, you may make changes to NQS only by using user exits.

5.25.1 NQS User Exits for UNICOS and UNICOS/mk Systems

For UNICOS and UNICOS/mk systems that support the user exit feature, the NQS user exits (which are located in the `/usr/src/lib/libuex` directory) let you introduce local code at defined points (compatible from release to release) to customize NQS without having access to the source.

The user exits let sites tailor various functions of NQS, including queue destination selection, `qsub`(1) option preprocessing, job startup and termination processing, and NQS startup and shutdown processing.

NQS user exits are used for the following functions:

- NQS daemon packet. Lets sites add functionality when a packet arrives.

- NQS destination ordering. Lets sites control the order of pipe queue destinations within NQS.
- NQS fair-share priority calculation for the *share_pri* value.
- NQS job selection. Lets sites determine whether the request chosen by NQS should be started, and lets them customize the order in which requests are started by NQS.
- NQS job initiation. Allows a user exit before job initiation.
- NQS job termination. Allows a user exit after job termination.
- NQS `qmgr`(8) command. Allows additional functionality when a `qmgr` command will be processed.
- `qsub` directives. Allows user exits before the first `#QSUB` directive, on each `#QSUB` directive, and after the last `#QSUB` directive.
- NQS startup. Lets sites perform processing during the NQS startup.
- NQS shutdown. Lets sites perform processing during the NQS shutdown.
- Job submission. NQS uses the centralized user password identification and authentication (I&A) routines. The user exits that are a part of the new validation routines allow sites to implement their own NQS user identification and validation algorithms. For more information on the I&A user exits in the multilevel security feature, see your UNICOS or UNICOS/mk system administration documentation.

To use the user exits, follow these steps:

Warning: The NQE `/opt` header files are not available in a chroot environment. If you create an NQS user exit which references header files in /*nqebase*/`nqe/.../` and then build UNICOS, the `libuex.a` library build described in the steps below will fail. You must keep your UNICOS build and your build for NQS user exits separate.

1. Copy the user exit template file (or the site's customized file) to `/usr/src/lib/libuex/local` (for example, copy `nqs_uex_jobselect.template` to `local/nqs_uex_jobselect.c`).
2. Ensure that the path names to the NQS header files (`.h`) are correct.
3. Execute the `nmakefile` file by using the `nmake install` command in the `/usr/src/lib/libuex` directory to build the `libuex.a` library. This

library must be rebuilt whenever a user exit is modified, added to, or deleted from `libuex/local`.

4. Execute the NQS `nmakefile` file by using the `/etc/build_nqs` script in the /*nqebase*/`$NQE_VERSION` directory to rebuild NQS.

To disable a user exit, you must remove (or rename) your local user exit file (for example, `local/nqs_uex_jobselect.c`) and repeat steps 3 and 4 to rebuild `libuex.a` and NQS with the default user exit stub. For examples on how to code the NQS user exits, see the `/usr/src/lib/libuex` directory.

The NQS user exits are described as follows:

User exit	Description
`nqs_uex_dorder`	Lets sites change the destination list of a pipe client just before it tries to send a request to that list of destinations. To configure the pipe queue destinations, use the `qmgr create pipe_queue` and `qmgr add destinations` commands. The order of the pipe queue destinations determines the order in which they are contacted. If the configured list of destinations is empty, this user exit is not called. A return code of 0 is the only valid return code and indicates successful completion of this user exit; any other return code is treated as if it were a 0.
`nqs_uex_jobinit`	Lets sites provide additional functionality when a job is initiated. A site can use this exit to set additional environment variables. A return code of 0 is the only valid return code and indicates successful completion of this user exit; any other return code is treated as if it were a 0.
`nqs_uex_jobselect`	Determines whether the request chosen by NQS should be started. You can customize the order in which NQS starts requests. A return code of 0 indicates that normal NQS scheduling should occur. A return code of 1 indicates that the job should not be started. A

	return code of 2 indicates that the job should be started.
`nqs_uex_jobterm`	Lets sites provide additional functionality when a job is terminated. A return code of 0 is the only valid return code; any other return code is treated as if it were a 0.
`nqs_uex_packet`	Lets sites provide additional functionality when specific NQS packets arrive in the daemon. A user exit is required in the NQS daemon each time a packet arrives. A return code of 0 indicates that NQS should process this packet. A return code of 1 indicates that NQS should ignore this packet.
`nqs_uex_qmgr`	Lets sites provide additional functionality when a `qmgr` subcommand is entered, providing control of the operator functions and the customization of requests. A return code of 0 indicates that NQS should process this command. A return code of 1 indicates that NQS should ignore this command.
`nqs_uex_qsub_after`	Lets sites provide additional functionality after all `qsub` options have been processed. This user exit executes under the user's process. A return code of 0 indicates that NQS should continue processing this request. A return code of 1 indicates that NQS should discard this request.
`nqs_uex_qsub_before`	Lets sites provide additional functionality before `qsub` handles the first `QSUB` directive. This user exit executes under the user's process. A return code of 0 indicates that NQS should continue processing this request. A return code of 1 indicates that NQS should discard this request.
`nqs_uex_qsub_each`	Lets sites interrogate or change each `QSUB` directive that is processed by `QSUB`. This user exit executes under the user's process.

	A return code of 0 indicates that NQS should continue processing this request. A return code of 1 indicates that NQS should discard this request.
`nqs_uex_shrpri`	Lets sites provide their own calculation for the *share_pri* value when the fair-share scheduler is active. The *share_pri* value is used in the priority calculation for job scheduling. A return code of 0 indicates that the *share_pri* value generated by NQS should be used. A nonzero return code indicates that the *share_pri* value from this user exit should be used.
`nqs_uex_shutdown`	Lets sites do additional processing when the NQS daemon is terminating. A return code of 0 is the only valid return code; any other return code is treated as if it were a 0.
`nqs_uex_start`	Lets sites do additional processing when the NQS daemon is initializing. A return code of 0 indicates that NQS should continue the startup; a return code of 1 indicates that NQS should abort the startup.

5.25.2 NQS Source-code Modification for UNICOS Systems

For Cray PVP systems, you may make changes to NQS by making source-code modifications. The source code is provided when you receive NQS; it is located in /*nqebase*/`src`.

If you wish to modify NQS by changing the source-code, make a backup copy of the content of the /*nqebase* directory, and then modify the source as desired.

Rebuild NQS by using the `/etc/build_nqs` script in the /*nqebase*/`$NQE_VERSION` directory.

Operating NQS [6]

This chapter describes how a user defined as an NQS operator can monitor and control NQS on the local execution server. Most of these actions are performed using the commands of the `qmgr` utility. A user defined as an NQS manager can perform all of the actions described in this chapter.

This chapter discusses the following topics:

- Overview of operator actions
- Starting and stopping NQS on the local execution server
- Monitoring the local NQS server, including displaying system parameters, monitoring the status of queues and requests, and displaying the list of machines currently defined in the network
- Controlling the availability of local NQS queues, including starting and enabling queues
- Manipulating batch requests on the local execution server

6.1 Overview of Operator Actions

An NQS operator can perform the following actions on the local NQS server (these actions may also be performed by an NQS manager):

- Controlling the availability of NQS:
  - Starting up and shutting down the system
  - Enabling and disabling NQS queues
  - Starting and stopping NQS queues
- Monitoring NQS by using various `qmgr show` commands and by using the `cqstatl` or `qstat` utility to display the following:
  - The current values of system parameters
  - The status of NQS queues
  - A detailed description of the characteristics of NQS queues
  - The status of individual requests in NQS queues

You must use the `cqstatl` or `qstat` command to display information about NQS queues.

- Controlling the processing of batch requests that users submit to the system:
  - Moving requests between queues or within queues
  - Holding and releasing requests in various ways
  - Modifying characteristics of a request
  - Deleting requests

6.2 Starting NQS

The `nqeinit` script is responsible for starting all NQE components. It calls `qstart` to start the NQS component. You can also start an individual process by executing the `qstart` command (as follows), which then automatically starts the other NQS processes.

```
% qstart options
```

Note: For customers who plan to install PALs on UNICOS systems that run only the NQE subset (NQS and FTA components), the `qstart -c` option should be used to direct the NQS console file into a directory that does not require `root` to write to it.

See the `qstart`(8) man page for a complete description of the syntax of the `qstart` command.

The `qstart` command is a shell script that starts `nqsdaemon` and passes `qmgr` startup commands to the daemon. By default, `qstart` performs the following functions:

- Starts `nqsdaemon`
- Initializes the system's NQS environment with commands defined in the `qmgr` input file

On startup, the `nqeinit` and `qstart` commands check whether the NQS database has been created. If it has not, the commands create it. During the creation of the NQS database, default values are taken from the following file:

```
/nqebase/etc/nqs_config
```

When NQS is being configured, this process occurs only once, unless you delete the NQS database structure. After the database is created, `nqs_config` is not used again.

During each startup of NQS, /*nqebase*/`etc` is examined for a file called `NQS.startup`. If the file exists, NQS uses it as input to `qmgr` as NQS is being started. If it does not exist, NQS is brought up and the `qmgr start all` command is issued. You can see a log of the startup activity in the NQS log directory (`$NQE_SPOOL/log`).

The directory structure under `$NQE_SPOOL/log` is as follows:

```
# cd $NQE_SPOOL
# ls
# ftaqueue   log    nlbdir   spool
# ls log
nqscon.26553.out  nqslog      qstart.26553.out
```

Except for the `nqslog` file, the files have the UNIX process ID appended, which makes the files unique across invocations of `qstart` and `qstop`.

After the initial boot of NQS on a system, you should make subsequent database changes interactively using `qmgr`. After changes are made, you can use the `qmgr snap file` command to save the current configuration. To update the standard configuration file, you can replace the `nqs_config` file with the snap file.

Note: In order to use the `qmgr snap file` command, you must have NQS manager privilege.

You do not need an `NQS.startup` file to preserve changes made to `qmgr`. Any `qmgr` changes are written to the NQS database and preserved across shutdowns and startups. You should make a copy of your changes to either the `nqs_config` or `NQS.startup` file, in case your database is ever corrupted.

After `nqsdaemon` is initialized, `qstart` determines whether the NQS spool database (usually in `$NQE_SPOOL/spool`) exists. If `qstart` finds that the database is missing, the /*nqebase*/`etc/nqs_config` configuration file is sent as input to `qmgr`. This file should contain the complete NQS database configuration, including the machine IDs, queues, and global limits. You can copy the output from a `qmgr` snap file to /*nqebase*/`etc/nqs_config` when you are doing an upgrade and an NQS database must be constructed. NQS executes the /*nqebase*/`etc/NQS.startup` file each time it starts up, whether or not an NQS database exists.

The changes that you make to the /*nqebase*/`etc/nqs_config` file are not propagated automatically into the active NQS spool database. Usually, the database is in `$NQE_SPOOL/spool` when the system is booted; therefore, the configuration is already loaded (except when the system is initially installed or when a clean `$NQE_SPOOL/spool` is being used).

To ensure that the NQS daemon is running, use the following command:

```
# /nqebase/bin/qping -v
# nqs-2550 qping: INFO
#   The NQS daemon is running.
```

The following is a sample `NQS.startup` file, which the site can configure. A copy of the file, as it is shown in this example, is contained in /*nqebase*/`examples`.

```
# USMID %Z%%M%  %I%     %G% %U%
#
#   NQS.startup - Qmgr commands to start up NQS.
#
#   Example of site-configurable script that resides in /nqebase/etc/config;
#   /nqebase/etc/qstart will invoke /nqebase/etc/NQS.startup unless the
#   -i option is specified.
#
#
#   Set general parameters.
#
#   Fewer timeouts on qstat, qmgr, etc., if nqsdaemon locked in memory.
#
lock local_daemon
#
#   Site may not run accounting during nonprime time to save disk space.
#
set accounting on
#
#   Could have put checkpoint directory in /tmp/nqs/chkpnt
#   for a benchmark; so make sure it is in usual place
#   for prime time.
#
set checkpoint_directory =(/usr/spool/nqe/spool/private/root/chkpnt)
#
#   Could ensure that NQS will not fill up /usr/spool/nqe by
#   setting the log_file to /tmp/nqs/nqslog.
#
set log_file =/usr/spool/nqe/log/nqslog
#
#   Debug level may have been set to 3 (verbose) for testing;
#   reduce log file size for prime time by reducing level.
#
set debug 0
#
#   There are many NQS message types that can be turned on and
#   off separately. Set some message types on when wanting
#   more information logged for areas of interest or set all
#   message types on when tracking problems. Note that
#   "set message_type on all" will turn on all but "flow"
#   messages. Using "set message_type on flow" should only be
#   used very selectively in controlled situations for tracking
#   a very difficult problem.
```

```
#
set message_type off all
#
#   NQS message headers in the log file can now be set to short
#   (same as always) or long, which includes information on where
#   the message is being issued from. Use long when tracking
#   problems.
#
set message_header short
#
#   NQS log files can be automatically segmented based on time, size,
#   or at each startup of NQS. The directory to which NQS will
#   segment the log file is specified with the "set segment directory"
#   command.
#
set segment directory /usr/spool/nqe/log
#
#   NQS can now automatically segment the NQS log file each time NQS
#   is started.
#
set segment on_init on
#
#   Name "batch" is not magic; corresponds to hypothetical pipe_queue.
#   See nqs/example/qmgr.example.
#
set default batch_request queue batch
#
#   Number of hours during which a pipe queue can be unreachable;
#   a routing request that exceeds this limit is marked as failed.
#
set default destination_retry time 72
#
#   Number of minutes to wait before trying a pipe queue destination
#   that was unreachable at the time of the last attempt.
#
set default destination_retry wait 5
#
#   Turn on the default return of the user's job log file.
#
set default job_log on
#
#   Assumes NQS account exists on the Cray Research system
#   and belongs to group root. If -me or -mb on qsub; this is
#   who mail will be from.
#
set mail nqs
#
#
#   Now set global parameters.
#
set global batch_limit = 20
set global user_limit = 20
set global group_limit = 15
#
#   Consider this number an "oversubscription"; it does not
#   have to be the size of the machine.
#
set global memory_limit = 256Mw
#
#   Consider this number an "oversubscription"; it does not
#   have to be the size of the SSD.
#
set global quickfile_limit = 1Gw   # Y-MP Only
#
#   The numbers in each tape group must reflect
#   local site configuration.
#
set global tape_limit a = 10
set global tape_limit b = 5
set global tape_limit c = 5
set global tape_limit d = 0
set global tape_limit e = 0
set global tape_limit f = 0
set global tape_limit g = 0
set global tape_limit h = 0
#
#   Maximum number of pipe requests allowed to run concurrently.
#
set global pipe_limit = 5
#
#   Capture configuration status at startup; written to
#   /usr/spool/nqe/log/qstart.out unless otherwise specified.
#   These commands are entirely optional, but handy
#   for reference if problems occur.
#
sho queues
sho parameters
sho mids
sho man
#
#    Now NQS will begin scheduling jobs.
#
start all_queues
```

6.3 NQS Shutdown and Restart

The `nqestop` script is responsible for stopping all NQE components. It calls `qstop` to stop the NQS component. You can also stop an individual NQS process by executing the `qstop`(8) command as follows:

`% qstop` *options*

See the `qstop`(8) man page for a complete description of the syntax of the `qstop` command.

⚠ **Caution:** When you stop NQS by using the `qstop` command, you should be aware that you have not stopped NQE. The Network Load Balancer collector daemon uses `logdaemon` services . The `qstop` command does not stop NLB collectors; use `nqestop`(8) to stop all of NQE.

When the `qmgr shutdown` command shuts down NQS, all processes that make up a restartable running batch request on the local host are killed. On UNICOS, UNICOS/mk, and IRIX systems, an image of the request is saved in a file in the checkpoint directory by NQS using the `chkpnt`(2) system call on UNICOS and UNICOS/mk systems and the `cpr`(1) command on IRIX systems.

For UNICOS, UNICOS/mk, and IRIX systems, for a batch request to be considered restartable, it must meet the recoverability criteria of job and process recovery, as described in *General UNICOS System Administration*, publication SG–2301 or in the *Checkpoint and Restart Operation Guide*, publication 007–3236–001, a Silicon Graphics publication.

When NQS is restarted, checkpointed requests are recovered from the images in their respective restart files. They resume processing from the point of suspension before the shutdown. After a request is restarted successfully, the restart file is kept until the request completes (in which case, the file is removed) or the request is checkpointed again (in which case, the file is overwritten).

The following is a sample `NQS.shutdown` file, which the site can configure. A copy of the file, as it is shown in this example, is contained in /*nqebase*/`examples`.

```
# USMID %Z%%M%  %I%     %G% %U%
#
#   NQS.shutdown - Qmgr commands to shut down NQS.
#
#   Example of site-configurable script that resides in /nqebase/etc/config;
#   /nqebase/etc/qstop invokes /etc/config/NQS.shutdown, unless the
#   -i option is specified.
#
stop all
#
#   The "set accounting off" line has been removed from this example
#   as it was found that setting accounting off at NQS.shutdown time
#   turned NQS accounting off before accounting records had been given
#   their appropriate terminating status and this caused problems with
#   the post-processing accounting routine summaries of the accounting
#   data.
#
#   The 60-second grace period means shutdown will take longer than 60
#   seconds. The nqsdaemon sends a SIGSHUTDN signal to the processes
#   of all running requests and then waits for the number of seconds
#   specified by the grace period. After the grace period, nqsdaemon
#   attempts to checkpoint all running requests that are restartable;
#   i.e., qsub option -nc NOT specified. nqsdaemon will send SIGKILL to
#   processes of requests that were not checkpointed, including those
#   for which the checkpoint failed. All checkpointed requests and
#   rerunnable requests (i.e., qsub option -nr NOT specified) will be
#   requeued to be restarted or rerun when NQS is next initiated.
#
shutdown 60
```

6.4 Monitoring NQS

Several `qmgr` commands are available to monitor NQS; these commands all begin with `show`.

You also can use the NQE GUI `Status` display and the `cqstatl` or the `qstat` command to gather valuable information about requests and queues.

6.4.1 Displaying System Parameters

To display the current values for system parameters, use the following `qmgr` command:

```
show parameters
```

An example of the display follows:

```
Qmgr: sho parameters
sho para

  Checkpoint directory =
/nqebase/version/pendulum/database/spool/private/root/chkpnt
  Debug level = 1
  Default batch_request queue = nqenlb
  Default destination_retry time = 72 hours
  Default destination_retry wait = 5 minutes
  Default return of a request's job log is OFF
  Global batch group run-limit: unspecified
  Global batch run-limit:       5
  Global batch user run-limit: 2
  Global MPP Processor Element limit:   unspecified
  Global memory limit:          unlimited
  Global pipe limit:            5
  Global quick-file limit:      unspecified
  Global tape-drive a limit:    unspecified
  Global tape-drive b limit:    unspecified
  Global tape-drive c limit:    unspecified
  Global tape-drive d limit:    unspecified
  Global tape-drive e limit:    unspecified
  Global tape-drive f limit:    unspecified
  Global tape-drive g limit:    unspecified
  Global tape-drive h limit:    unspecified
  Job Initiation Weighting Factors:
 Fair Share Priority = 0    (Fair Share is not enabled)
    Requested CPU Time  = 0
    Requested Memory    = 0
    Time-in-Queue       = 1
    Requested MPP CPU Time  = 0
    Requested MPP PEs       = 0
    User Specified Priority = 0
Job Scheduling:   Configured = nqs normal   Active = nqs normal
  Log_file = /nqebase/version/pendulum/database/spool/../log/nqslog
  MESSAGE_Header = Short
```

```
MESSAGE_Types:
    Accounting         OFF    CHeckpoint         OFF    COMmand_flow       OFF
    CONfig             OFF    DB_Misc            OFF    DB_Reads           OFF
    DB_Writes          OFF    Flow               OFF    NETWORK_Misc       OFF
    NETWORK_Reads      OFF    NETWORK_Writes     OFF    OPer               OFF
    OUtput             OFF    PACKET_Contents    OFF    PACKET_Flow        OFF
    PROTOCOL_Contents  OFF    PROTOCOL_Flow      OFF    RECovery           OFF
    REQuest            OFF    ROuting            OFF    Scheduling         OFF
    USER1              OFF    USER2              OFF    USER3              OFF
    USER4              OFF    USER5              OFF
  Mail account = root
  Netdaemon = /nqebase/bin/netdaemon
  Network client = /nqebase/bin/netclient
  Network retry time = 31 seconds
  Network server = /nqebase/bin/netserver
  NQS daemon accounting is OFF
NQS daemon is not locked in memory
  Periodic_checkpoint concurrent_checkpoints = 1
  Periodic_checkpoint cpu_interval = 60
  Periodic_checkpoint cpu_status on
  Periodic_checkpoint max_mem_limit = 32mw
  Periodic_checkpoint max_sds_limit = 64mw
  Periodic_checkpoint min_cpu_limit = 60
  Periodic_checkpoint scan_interval = 60
  Periodic_checkpoint status off
  Periodic_checkpoint time_interval = 180
  Periodic_checkpoint time_status off
  SEgment Directory = NONE
  SEgment On_init = OFf
  SEgment Size = 0 bytes
  SEgment Time_interval = 0 minutes
  Sequence number for next request = 0
  Snapfile = /nqebase/version/pendulum/database/spool/snapfile
  Validation type = validation files
```

This information also is displayed by the following `qmgr` command (along with the list of managers and operators and any user access restrictions on queues):

```
show all
```

The following `qmgr` command displays only the global limits (a subset of the output displayed in the preceding command):

```
show global_parameters
```

All entries in the `show parameters` display, except for the `Sequence number for next request`, can be configured by `qmgr` commands.

> **Note:** Some parameters are not enforced if the operating system does not support the feature, such as MPP processing elements or checkpointing.

The following list explains each entry and shows the `qmgr` command (in parentheses) that is used to change the entry. Most of these commands can be used only by an NQS manager; commands that can be issued by an NQS operator are prefixed by a dagger (†).

Display entry	Description
`Debug level`	The level of information written to the log file (`set debug`).
`Default batch request queue`	The queue to which requests are submitted if the user does not specify a queue (`set [no_]default batch_request queue`).
`Default destination retry time`	The maximum time that NQS tries to send a request to a pipe queue (`set default destination_retry time`).
`Default destination retry wait`	The interval between successive attempts to retry sending a request (`set default destination_retry wait`).
`Default return of request’s job log`	Determines whether the default action is to send a job log to users when the request is complete (`set default job log on/off`).
`Global batch group run-limit`	The maximum number of batch requests that all users of a group can run concurrently (†`set global group_limit`).

`Global batch run-limit`

The maximum number of batch requests that can run simultaneously at a host (†`set global batch_limit`).

`Global batch user run-limit`

The maximum number of batch requests any one user can run concurrently (†`set global user_limit`).

`Global MPP Processor Element limit`

The maximum number of MPP processing elements available to all batch requests running concurrently (†`set global mpp_pe_limit`). Valid only on the Cray MPP systems and optionally enabled on IRIX systems (see Section 5.11.4, page 82).

`Global memory limit`

The maximum memory that can be allocated to all batch requests running concurrently (†`set global memory_limit`).

`Global pipe limit`

The maximum number of requests that can be routed simultaneously at the host (†`setglobal pipe_limit`).

`Global quick-file limit`

The maximum quickfile space that can be allocated to all batch requests running concurrently on NQS on UNICOS systems (†`set global quickfile_limit`).

`Global tape-drive` *x* `limit`

The maximum number of the specified tape drives that can be allocated to all batch requests running concurrently (†`set global tape_limit`).

`Job Initiation Weighting Factoring`

The weight of the specified factor in selecting the next request initiated in a queue (`set sched_factor cpu|memory|share|time|mpp_cpu|mpp_pe|user_priority`)

`Job Scheduling` *option(s)*

The job scheduling type used by `nqsdaemon` (`set job_scheduling`).

`Log file`

The path name of the log file (`set log_file`).

`MESSAGE_Header`

The NQS message header (`set message_header long|short`).

`MESSAGE_Types`

The type of NQS messages sent to message destinations such as the NQS log file or the user's terminal (`set message_types off|on`).

`Mail account`

The name that appears in the `From:` field of mail sent to users by NQS (`set mail`).

`Netdaemon`

The name of the TCP/IP network daemon (`set [no_]network daemon`).

`Network client`

The name of the network client process (`set network client`).

`Network retry time`

The maximum number of seconds a network function can fail before being marked as completely failed (`set network retry_time`).

`Network server`

The name of the network server process (`set network server`).

`NQS daemon accounting`

Determines whether NQS daemon accounting is enabled (`set accounting off|on`).

`NQS daemon is/is not locked in memory`

The state of the `nqsdaemon` process; that is, whether it is locked into memory or not (†`lock local daemon` and †`unlock local daemon`).

`Periodic_checkpoint concurrent_checkpoints`

The maximum number of checkpoints that can occur simultaneously during a periodic checkpoint scan interval (valid only on UNICOS, UNICOS/mk, and IRIX systems) (`set periodic_checkpoint concurrent_checkpoints`).

`Periodic_checkpoint cpu_interval`

The default CPU time interval for periodic checkpointing (valid only on UNICOS, UNICOS/mk, and IRIX systems) (`set periodic_checkpoint cpu_interval`).

`Periodic_checkpoint cpu_status on`

Determines whether NQS periodic checkpoints are initiated based on the CPU time used by a request (valid only on UNICOS, UNICOS/mk, and IRIX systems) (`set periodic_checkpoint cpu_status off|on`).

`Periodic_checkpoint max_mem_limit`

The maximum memory limit for a request that can be periodically checkpointed (valid only on UNICOS, UNICOS/mk, and IRIX systems) (`set periodic_checkpoint max_mem_limit`).

`Periodic_checkpoint max_sds_limit`

The maximum SDS limit for a request that can be periodically checkpointed (valid only on UNICOS systems) (`set periodic_checkpoint max_sds_limit`).

`Periodic_checkpoint min_cpu_limit`

The minimum CPU limit for a request that can be periodically checkpointed (valid only on UNICOS, UNICOS/mk, and IRIX systems) (`set periodic_checkpoint max_mem_limit`).

`Periodic_checkpoint scan_interval`

The time interval in which NQS scans running requests to find those eligible for periodic checkpointing (valid only on UNICOS, UNICOS/mk, and IRIX systems) (`set periodic_checkpoint scan_interval`).

`Periodic_checkpoint status off`

Determines whether NQS will examine running requests to see whether they can be checkpointed and then schedule them for checkpointing (`on`). If the status is `off`, no requests are periodically checkpointed (valid only on UNICOS, UNICOS/mk, and IRIX systems) (`set periodic_checkpoint status off|on`).

`Periodic_checkpoint time_interval`

The default wall-clock time interval for periodic checkpointing (valid only on UNICOS, UNICOS/mk, and IRIX systems) (`set periodic_checkpoint cpu_interval`).

`Periodic_checkpoint time_status off`

Determines whether NQS periodic checkpoints are initiated based on wall-clock time used by a request (valid only on UNICOS, UNICOS/mk, and IRIX systems) (`set periodic_checkpoint time_status off|on`).

`SEgment Directory`

The name of the directory containing log file segments (`set segment directory`).

`SEgment ON_init`

Determines whether the log file is segmented at initialization (`set segment on_init off|on`).

`SEgment Size`

The maximum size of an NQS log file before it is segmented (`set segment size`).

`SEgment Time_interval`

The maximum time that can elapse before the NQS log file is segmented (`set segment time_interval`).

`Sequence number for next request`

The next sequence number that will be assigned to a batch request.

`Snap_file`

The default name of the file that will receive output from the `snap` command (`set snapfile`).

`Validation type`

The type of user validation that will be performed on user requests (`set [no_]validation`).

6.4.2 Displaying a List of Managers and Operators

To display a list of the currently defined NQS managers and operators for this server, use the following `qmgr` command:

```
show managers
```

In the resulting display, users who are managers are indicated by a `:m` suffix; operators are indicated by a `:o` suffix, as follows:

```
Qmgr: show managers
show managers

   root:m
   snowy:m
   fred:o
   xyz:o
```

The list always includes `root` (as a manager).

6.4.3 Displaying a Summary Status of NQS Queues

To display a brief summary status of all currently defined batch and pipe queues, use the following `qmgr` command:

```
show queue
```

An example of the display follows; it is identical to that produced by the `cqstatl -s` or `qstat -s` command:

```
Qmgr: show queue
show queue
-----------------------------
NQS BATCH QUEUE SUMMARY
-----------------------------
QUEUE NAME              LIM TOT ENA STS QUE RUN
----------------------- --- --- --- --- --- ---
nqebatch                  5   0 yes  on   0   0
----------------------- --- --- --- --- --- ---
pendulum                  5   0           0   0
----------------------- --- --- --- --- --- ---
----------------------------
NQS PIPE QUEUE SUMMARY
----------------------------
QUEUE NAME              LIM TOT ENA STS QUE ROU
----------------------- --- --- --- --- --- ---
nqenlb                    1   0 yes  on   0   0
----------------------- --- --- --- --- --- ---
pendulum                  5   0           0   0
----------------------- --- --- --- --- --- ---
```

To display a more detailed summary, use the following `qmgr` command:

```
show long queue
```

An example of the display follows; it is identical to that produced by the `cqstatl` or `qstat` command.

```
Qmgr: show long queue
show long queue
-----------------------------
NQS BATCH QUEUE SUMMARY
-----------------------------
QUEUE NAME             LIM TOT ENA STS QUE RUN  WAI HLD ARR EXI
---------------------- --- --- --- --- --- ---  --- --- --- ---
nqebatch                 5   0 yes  on   0   0    0   0   0   0
---------------------- --- --- --- --- --- ---  --- --- --- ---
latte                    5   0           0   0    0   0   0   0
---------------------- --- --- --- --- --- ---  --- --- --- ---
----------------------------
NQS PIPE QUEUE SUMMARY
----------------------------
QUEUE NAME             LIM TOT ENA STS QUE ROU  WAI HLD ARR DEP  DESTINATIONS
---------------------- --- --- --- --- --- ---  --- --- --- ---  -------------
nqenlb                   1   0 yes  on   0   0    0   0   0   0
cool                     1   0 yes  on   0   0    0   0   0   0  batch@cool
---------------------- --- --- --- --- --- ---  --- --- --- ---  -------------
latte                    5   0           0   0    0   0   0   0
---------------------- --- --- --- --- --- ---  --- --- --- ---  -------------
```

The columns in these summary displays are described as follows:

Column name	Description
`QUEUE NAME`	Indicates the name of the queue.
`LIM`	Indicates the maximum number of requests that can be processed in this queue at any one time. This is the run limit that the NQS manager defines for the queue. For batch queues, `LIM` indicates the maximum number of requests that can execute in this queue at one time. After this limit is reached, additional requests will be queued until a request already in the queue completes execution. For pipe queues, `LIM` indicates the maximum number of requests that can be routed at one time.
`TOT`	Indicates the total number of requests currently in the queue.

`ENA`	Indicates whether the queue was enabled to accept requests. If the queue is disabled (`ENA` is `no`), the queue will not accept requests.
`STS`	Indicates whether the queue has been started. If a queue was not started (`STS` is `off`), the queue will accept requests, but it will not process them.
`QUE`	The number of requests that are waiting to be processed.
`ROU`	The number of pipe queue requests that are currently being routed to another queue.
`RUN`	The number of batch queue requests that are currently running.
`WAI`	The number of requests that are waiting to be processed at a specific time.
`HLD`	The number of requests the NQS operator or manager has put into the hold state.
`ARR`	The number of requests currently arriving from other queues.
`DEP`	The number of requests currently terminating their processing.
`DESTINATIONS`	The destinations that were defined for the queue by the NQS manager.

The last line of the display shows the total figures for the use of the queues by users at the NQS host (`host1` in the example), except for the `LIM` column, which shows the global pipe limit for this system.

When you use a `cqstatl` command rather than `qmgr show` commands, you can limit the display to NQS pipe queues by using the `-p` option, or limit the display to NQS batch queues by using the `-b` option.

6.4.4 Displaying All the Characteristics Defined for an NQS Queue

To display details of all the characteristics defined for NQS queues, use the `cqstatl -f` or the `qstat -f` command. You can restrict the display to specific queues by using the *queues* option.

You must separate queue names with a space character.

An example of a full display for the batch queue `nqebatch` follows:

```
% cqstatl -f nqebatch
-------------------------------------
NQS BATCH QUEUE: nqebatch@latte        Status:          ENABLED/INACTIVE
-------------------------------------
                                              Priority:       30
<ENTRIES>
        Total:          0
        Running:        0       Queued:         0       Waiting:        0
        Holding:        0       Arriving:       0       Exiting:        0
<RUN LIMITS>
        Queue:          5       User: unspecified       Group: unspecified
<COMPLEX MEMBERSHIP><RESOURCE USAGE>
                                   LIMIT                ALLOCATED
        Memory Size             unspecified                    0kw
        Quick File Space        unspecified                    0kw
        MPP Processor Elements  unspecified                    0
<REQUEST LIMITS>
                                PER-PROCESS             PER-REQUEST
        type a Tape Drives                              unspecified
        type b Tape Drives                              unspecified
        type c Tape Drives                              unspecified
        type d Tape Drives                              unspecified
        type e Tape Drives                              unspecified
        type f Tape Drives                              unspecified
        type g Tape Drives                              unspecified
        type h Tape Drives                              unspecified
        Core File Size          unspecified
        Data Size               unspecified
        Permanent File Space    unspecified             unspecified
        Memory Size             unspecified             unspecified
        Nice Increment                0
        Quick File Space        unspecified             unspecified
        Stack Size              unspecified
        CPU Time Limit           unlimited               unlimited
        Temporary File Space    unspecified             unspecified
        Working Set Limit       unspecified
        MPP Processor Elements                          unspecified
        MPP Time Limit          unspecified             unspecified
        Shared Memory Limit                               unspecified
        Shared Memory Segments                            unspecified
        MPP Memory Size          unspecified             unspecified

<ACCESS>
        Route: Unrestricted            Users: Unrestricted
<CUMULATIVE TIME>
        System Time:    904.22 secs     User Time:      59.14 secs
```

Some parameters are not enforced if the operating system does not support the feature, such as MPP processor elements.

An example of a full display for the pipe queue `nqenlb` follows:

```
% cqstatl -f nqenlb
----------------------------------
NQS PIPE QUEUE: nqenlb@latte       Status:      ENABLED/INACTIVE
----------------------------------
                                          Priority:       63
<ENTRIES>
     Total:          0

     Running:        0       Queued:         0       Waiting:        0

     Holding:        0       Arriving:       0       Departing:      0

<DESTINATIONS>

<SERVER>
        /nqebase/bin/pipeclient CRI_DS

<ACCESS>
        Route: Unrestricted            Users: Unrestricted

<CUMULATIVE TIME>
        System Time:    269.92 secs     User Time:      95.21 secs
```

The `cqstatl` or `qstat` command also can be used to display details of characteristics defined for NQS queues on other hosts. The difference is that you must give the name of the remote host where the queues are located. For example, for the `cqstatl` command, you could include the `-h` option to specify the remote host, as follows:

```
cqstatl -d nqs -h target_host -f queues
```

You also can change your `NQS_SERVER` environment variable to specify the remote host.

6.4.5 Displaying Summary Status of User Requests in NQS Queues

You can display a summary of the requests currently in the queue if you include the *queue-name* parameter in the `qmgr` commands `show queue` and `show long queue`:

```
show queue queue-name
show long queue queue-name
```

If no requests are currently in the queue, the following message is displayed:

```
nqs-2224 qmgr: CAUTION
no requests in queue queue-name
```

You can restrict the display to the requests in the queue that were submitted by a specific user by including the name of that user after the *queue-name* argument:

```
show queue queue-name username
show long queue queue-name username
```

An example of the display for a batch queue follows:

```
Qmgr: show long queue nqebatch
show long queue nqebatch
-------------------------------
NQS BATCH REQUEST SUMMARY
-------------------------------
IDENTIFIER NAME  USER     LOCATION/QUEUE   JID  PRTY REQMEM REQTIM ST
---------- ----- -------- --------------- ---- ---- ------ ------ ---
39.latte   STDIN mary     nqebatch@latte   10644  --- 262144     ** R
```

The columns in this display have the following descriptions:

Column name	Description
`IDENTIFIER`	The request identifier (as displayed when the request was submitted).

`NAME`	The name of the script file.
`USER`	The user name under which the request will be executed at the NQS system.
`LOCATION/QUEUE`	The NQS queue in which the request currently resides.
`JID`	The job identifier for the request at the NQS system.
`PRTY`	The nice value of the request.
`REQMEM`	The number of kilowords of memory the request is using.
`REQTIM`	The number of seconds of CPU time remaining to the request.
`ST`	An indication of the current state of the request. This can be composed of a major and a minor status value. Major status values are as follows:

`A`	Arriving
`C`	Checkpointed
`D`	Departing
`E`	Exiting
`H`	Held
`Q`	Queued
`R`	Running
`S`	Suspended
`U`	Unknown state
`W`	Waiting

See the `cqstatl`(1) and `qstat`(1) man pages for a list of the minor values.

The `show queue` display has much of this information, but it does not contain the `USER`, `QUEUE`, `JID`, and `PRTY` fields.

The following example display is a summary of all requests in an NQS pipe queue called `squall` at a system called `host1`:

```
Qmgr: **show long queue nqenlb**
show long queue nqenlb
------------------------------
NQS PIPE REQUEST SUMMARY
------------------------------
IDENTIFIER    NAME    OWNER    USER     LOCATION/QUEUE        PRTY ST
------------- ------- -------- -------- -------------------- ---- ---
40.latte      STDIN   1201     mary     nqenlb@latte            1 Q
```

The columns in this display have the following meanings:

Column name	Description
`IDENTIFIER`	The request identifier (as displayed when the request was submitted).
`NAME`	The name of the script file, or `stdin` if the request was created from standard input.
`OWNER`	The user name under which the request was submitted.
`USER`	The user name under which the request will be executed.
`LOCATION/QUEUE`	The queue in which the request currently resides.
`PRTY`	The intraqueue priority of the request.
`ST`	An indication of the current state of the request (in this case, `Q` means the request is queued and ready to be routed to a batch queue).

The `show queue` display has much of this information in it, but it does not contain the `OWNER`, `USER`, `QUEUE`, and `PRTY` fields.

A list of all requests that run in all NQS queues can be displayed by `root` or an NQS manager (otherwise, the display shows only the submitting user's requests). Use the `cqstatl` or `qstat` command with the `-a` option; for example:

```
cqstatl -a
```

Note: For information about displaying requests in the NQE database, see Chapter 9, page 231.

6.4.6 Displaying the Mid Database

To display a list of NQS systems that are currently defined in the machine ID (*mid*) database, use the following `qmgr` command:

```
show mid
```

An example of the display follows:

```
Qmgr: sho mid
sho mid
   MID      PRINCIPAL NAME   AGENTS    ALIASES
 --------   --------------   ------    -------
 10111298   latte            nqs       latte.cray.com
 10653450   pendulum         nqs
 10671237   gust             nqs       gust.cray.com
```

This display shows that three systems are currently defined; the descriptions of these entries are explained in Section 5.3, page 50.

The following `qmgr` command displays information on a specific *mid* or host:

`show mid` *hostname* | *mid*

An example of such a display follows:

```
Qmgr: show mid latte
show mid latte
   MID      PRINCIPAL NAME   AGENTS    ALIASES
 --------   --------------   ------    -------
 10111298   latte            nqs       latte.cray.com
```

6.5 Controlling the Availability of NQS Queues

Before an NQS queue can receive and process requests submitted by a user, it must meet the following requirements:

- The queue must be enabled so that it can receive requests. If a queue is disabled, it can still process requests already in the queue, but it cannot receive any new requests.

- The queue must be started so that it can process requests that are waiting in the queue. If a queue is stopped, it can still receive new requests from users, but it will not try to process them.

The NQS operator (or manager) can control these states, thereby controlling the availability of a queue.

For example, you can disable all queues before shutting down the NQS system and prevent users from submitting any additional requests. The requests that are already in the queues will be processed.

Another example might be when no destination for a pipe queue will be available for some time (for example, the destination host may not be functioning). You can stop the queue rather than having NQS repeatedly try to send requests and deplete system resources. There is a system limit to the amount of time a request can wait in a queue.

6.5.1 Enabling and Disabling Queues

To enable a queue and allow requests to be placed in it, use the following `qmgr` command:

`enable queue` *queue-name*

If the queue is already enabled, no action is performed.

To disable a queue and prevent any additional requests from being placed in it, use the following `qmgr` command:

`disable queue` *queue-name*

If the queue is already disabled, no action is performed. If requests are already in the queue, these can still be processed (if the queue has been started).

An NQS manager also can prevent or enable individual users' access to a queue (see Section 5.8, page 69).

6.5.1.1 Example of Enabling and Disabling Queues

In the following example, you want to prevent users from sending requests to the NQS queue called `express` for a few hours. To disable the queue, use the following `qmgr` command:

```
disable queue express
```

When you want to enable the queue, use the following `qmgr` command:

```
enable queue express
```

6.5.2 Starting and Stopping Queues

To start an NQS queue and allow it to process any requests waiting in it, use the following `qmgr` command:

`start queue` *queue-name*

If the queue is already started, no action is performed.

To start all NQS queues, use the following `qmgr` command:

```
start all_queues
```

To stop a queue and prevent any further requests in it from being processed, use the following `qmgr` command:

`stop queue` *queue-name*

If the queue is already stopped, no action is performed. A request that had already begun processing before the command was issued is allowed to complete processing. All other requests in the queue, and any new requests placed in the queue, are not processed until the queue is started.

To stop all NQS queues, use the following `qmgr` command:

```
stop all_queues
```

You can also suspend the processing of a specific request in a queue (instead of the entire queue); see Section 6.6.4, page 165.

6.5.2.1 Example of Starting and Stopping Queues

In the following example, you can stop all NQS queues except the queue called `cray1-std`, by first stopping all queues and then restarting `cray1-std`, as follows:

```
stop all_queues
start queue cray1-std
```

The following `qmgr` command starts all of the queues:

```
start all_queues
```

6.6 Manipulating Requests

You can perform certain actions by using `qmgr` commands on requests that were submitted by users. To delete a request, you must have either operator privileges on NQS or be the owner of the request.

6.6.1 Moving Requests between NQS Queues

To move a request from one NQS queue to another, use the following `qmgr` command:

`move request =` *request queue_name*

A request can be moved only if it has not yet started processing (that is, has not yet started being transferred to another queue).

The *request* argument is the request identifier. It corresponds to the value displayed under the `IDENTIFIER` column of the summary request display (see Section 6.4.5, page 156). The *queue_name* argument is the name of the queue to hold the request.

To move more than one request, use the following `qmgr` command:

`move request = (`*requests*`)` *queue_name*

You must separate request identifiers with a space or a comma and enclose them all in parentheses.

To move all requests that have not started processing in one queue to another queue, use the following `qmgr` command:

`move queue` *from_queue_name to_queue_name*

6.6.1.1 Example of Moving Requests between Queues

The following example moves the two requests with request identifiers `123` and `135` from the current queue to another queue called `fast`:

```
move request = (123,135) fast
```

You do not have to specify the queue in which these requests currently reside.

6.6.2 Moving Requests within a Queue

The following `qmgr` command schedules queued requests manually, changing their position in a queue:

`schedule request` *request(s)* [*order*]

The *request* argument is one or more request identifiers of the request to be rescheduled. If you specify more than one request, you must enclose them all in parentheses and separate them with a space character or a comma.

The *order* argument can be one of the following:

- `first`, before all other requests in the queue, including restarted or rerun requests
- `next`, after all other `qmgr` scheduled requests in the queue
- `now`, immediately initiating the request and bypassing all NQS limit checks
- `system`, moving a previously scheduled request back to system-controlled scheduling

You can use the `schedule` command only on queued requests.

If a queue has a higher interqueue priority, the requests in that queue are usually initiated before those in queues with a lower interqueue priority.

6.6.2.1 Example of Moving Requests within a Queue

The following example repositions the requests `100`, `101`, and `102` to the order `101`, `102`, and then `100`:

```
schedule request 100 next
schedule request 101 now
schedule request 102 first
```

Request `101` is scheduled immediately, before all other queued requests.

Request `102` is scheduled to go before all other requests scheduled by `qmgr` (but not before those scheduled with the `schedule now` command, which take precedence).

Request `100` is scheduled to go after all requests scheduled by `qmgr`, but before system-scheduled requests.

6.6.3 Holding and Releasing Requests

To hold specific queued requests so that they are not processed, use the following `qmgr` command:

`hold request` *request(s)* [*grace-period*]

The *request* argument is one or more request identifiers of the requests to be held. If you specify more than one request, you must enclose them all in parentheses and separate them with a space character or a comma.

The *grace-period* argument is the number of seconds between when the command is issued and when the request is actually held; the default is 60 seconds.

Held requests become ineligible for processing; they are removed from the run queue and their NQS resources are released. UNICOS, UNICOS/mk, and IRIX requests that are running can be held; they are checkpointed until they are released. On other platforms, running requests cannot be held because they cannot be checkpointed.

You can hold more than one request at a time. Until it is released, you cannot process a held request (for instance, move it to another queue).

To release requests that are held, use the following `qmgr` command:

`release request` *request(s*)

The `qmgr` command `release request` does not affect any request that is in a state other than being held. No grace period exists for releasing a request.

6.6.3.1 Example of Holding and Releasing Requests

In the following example, the request with identifier `123` is immediately held to prevent it from being routed to a destination by using the following `qmgr` command:

```
hold request 123 0
```

When you want to begin processing the request, release the request by using the following `qmgr` command:

```
release request 123
```

6.6.4 Suspending, Resuming, and Rerunning Executing Requests

To suspend specific requests that are running and make their processes ineligible for execution, use the following `qmgr` command:

`suspend request` *request(s)*

The *request* argument is one or more request identifiers of the requests to be held. If you specify more than one request, you must enclose them all in parentheses and separate them with a space character or a comma.

The resources of the request are not released with the `suspend` command.

You can suspend more than one request at a time.

When you suspend a request, its processes are sent a `SIGSTOP` signal; when you resume processing of the request, a `SIGCONT` signal is sent. On UNICOS and UNICOS/mk systems, NQS uses the `suspend`(2) system call.

You cannot process a suspended request (for instance, route it to another queue) until it is resumed.

To resume requests that are suspended, use the following `qmgr` command:

`resume request` *request(s)*

The `resume request` command does not affect any request that is in a state other than suspension.

To abort and immediately requeue a *running* request, use the following `qmgr` command:

`rerun request` *request(s)*

This command kills processes of the request and the request is returned to the queued state at its current priority. If a request is not running or cannot be rerun, no action is taken. Users can specify that a request cannot be rerun by using the NQE GUI `Submit` display, the `cqsub` or `qsub` command with the `-nr` option, or the `qalter -r` command.

6.6.4.1 Example of Suspending and Resuming Requests

In the following example, the request with identifier `222` is suspended and its processes become ineligible for execution:

```
suspend request 222
```

When you want to begin processing the request, you must release the request by using the following `qmgr` command:

```
resume request 222
```

6.6.5 Modifying the Characteristics of Requests in Queues

The following `qmgr` command changes characteristics of a request in a queue:

`modify request` *request option=limit*

The *request* argument is the request identifier of the request to be modified.

The *limit* argument overrides the limit specified when the request was submitted and overrides the default limit defined for the batch queue in which the request is to be executed. NQS considers the new *limit* for selecting the batch queue in which the request will execute.

The *option* can set the following per-process and per-request attributes; the actual name of the *option* is given in parentheses:

Attribute	Description
CPU time limit	Per-request and per-process CPU time that can be used; on CRAY T3E MPP systems, this limit applies to command PEs (commands and single-PE applications) (`rtime_limit` and `ptime_limit`).
priority	NQS user specified priority of a request. To enable user specified priority for job initiation scheduling, you should set the *user_priority* weighting factor to nonzero and set all of the other weighting factors to 0. Then, only user specified priority will be used in determining a job's intraqueue priority for job initiation.
Memory limit	Per-request and per-process memory that can be used (`rmemory_limit` and `pmemory_limit`).
Nice value	Execution priority of the processes that form the batch request; specified by a nice increment between 1 and 19 (`nice_limit`).
Per-request permanent file space limit	Per-request permanent file space that can be used (`rpermfile_limit`).

Per-process permanent file space limit	Per-process permanent file space that can be used (`ppermfile_limit`).
Per-request MPP processing elements (CRAY T3D systems) or MPP application processing elements (CRAY T3E systems) or number of processors (IRIX systems) limit; optionally enabled on IRIX systems, see Section 5.11.4, page 82	Per-request Cray MPP processing elements (PEs) limit (`-l mpp_p`).
Per-process Cray MPP residency time limit	For CRAY T3D MPP systems, sets the per-process wallclock residency time limit, or for CRAY T3E MPP systems, sets the per-process CPU time limit for application PEs (multi-PE applications) for subsequent acceptance of a batch request into the specified batch queue (`-l p_mpp_time_limit`) .
Per-request Cray MPP residency time limit	For CRAY T3D MPP systems, sets the per-request wallclock residency time limit, or for CRAY T3E MPP systems, sets the per-request CPU time limit for application PEs (multi-PE applications) for subsequent acceptance of a batch request into the specified batch queue (`-l r_mpp_time_limit`) .
Per-request quick-file size limit	(UNICOS systems with SSDs) The user’s per-process secondary data segments (SDS) limit, which is set to the value of the user’s user database (UDB) entry for per-process SDS limits (`-lQ`).
Per-request shared memory size limit	Per-request shared memory size limit (`-l shm_limit`) .
Per-request shared memory segment limit	Per-request shared memory segment limit (`-l shm_segment`).
MPP memory size	Per-process and Per-request MPP application PE memory size limits (`-l p_mpp_m,mpp_m`).

You can add a suffix to the memory limit with one of the following characters (these are case sensitive) which denote the units of size used. A word is 8 bytes on UNICOS, UNICOS/mk, and 64-bit IRIX systems, and a word is 2 bytes on other supported systems.

Suffix	Units
`b`	Bytes (default)
`w`	Words
`kb`	Kilobytes (2^{10} bytes)
`kw`	Kilowords (2^{10} words)
`mb`	Megabytes (2^{20} bytes)
`mw`	Megawords (2^{20} words)
`gb`	Gigabytes (2^{30} bytes)
`gw`	Gigawords (2^{30} words)

6.6.5.1 Example of Modifying a Queued Request

In this example, a request with identifier `1340` is waiting to execute in an NQS queue. The submitter of the request wants to increase the memory allocated to the request to 500 Mwords. To do this, you can enter the following `qmgr` command:

```
modify request 1340 rmemory_limit=500mw
```

6.6.6 Deleting Requests in NQS Queues

The following `qmgr` commands delete a request that is in a NQS queue:

- `abort request`, which sends a `SIGKILL` signal to each of the processes of a running request.
- `delete request`, which deletes requests that are not currently running.

The request is removed from the queue and cannot be retrieved. The original script file of the request is unaffected by this command.

To delete all requests in a particular NQS queue, use the following `qmgr` command:

- `abort queue`, which sends a `SIGKILL` signal to all running requests in the specified queue
- `purge queue`, which purges all waiting, held, and queued requests in the specified queue but allows running requests to complete

To delete requests using the NQE GUI `Status` window, select the request to be deleted by clicking on the request line; then select `Delete` from the `Actions` menu.

You can also use the `cqdel` or the `qdel` command to delete requests; an example of the syntax follows:

`cqdel -u` *username requestids*

The *username* argument is the name of the user who submitted the request.

6.6.6.1 Example of Deleting a Queued Request

To delete all requests waiting in the queue `bqueue15`, use the following `qmgr` command:

```
purge queue bqueue15
```

6.6.7 Deleting or Signaling Requests in Remote NQS Queues

After a request is in an NQS queue at a remote system, you can use the NQE GUI `Status` window or the `cqdel` or `qdel` command to delete or signal the request (the `qmgr` subcommands affect only requests in local NQS queues). You must authenticate yourself by entering the correct password or having an entry in your `.rhosts` or `.nqshosts` file and in the `.rhosts` or `.nqshosts` file of the job's owner. You do not have any special privileges to delete other users' requests on remote NQS systems, even if you are an NQS operator or manager on both the local and the remote host.

Note: If you are using the NQE database, you only need system access to the NQE database from your remote host. For more information, see Chapter 9, page 231, and the `nqedbmgr`(8) man page.

To delete requests using the NQE GUI `Status` window, select the request to be deleted by clicking on the request line; then select `Delete Job` from the `Actions` menu. For more information about using the `cqdel` and `qdel` commands to delete or signal requests on NQS, see the `cqdel`(1) and `qdel`(1) man pages.

6.7 Recovering Jobs Terminated Because of Hardware Problems

When a process within a job has been terminated by a `SIGUME` or `SIGRPE` signal or a `SIGPEFAILURE` signal (UNICOS/mk systems only), NQS requeues the job rather than deleting it if either of the following is true:

- The job is rerunnable
- The job is restartable and has a restart file

Applications running on a CRAY T3E system are killed when a PE assigned to the application goes down. NQS is now notified when a job is terminated by a `SIGPEFAILURE` signal. NQS will requeue the job and either restart or rerun the job, as applicable.

Periodic checkpointing should be enabled so that restart files will be available for restarting rather than rerunning jobs which were terminated by a downed PE `SIGPEFAILURE` signal.

By default, each NQS job is both rerunnable and restartable. These defaults can be changed through use of the `qsub -nr` and `-nc` options and through use of the `qalter -r n` and `-c n` options. The job's owner can specify the `qsub` options and use the `qalter` command to modify the job rerun and/or restart attributes. An administrator can also use the `qalter` command to modify any job's rerun and/or restart attributes.

If NQS requeues a job because the job was terminated by either the `SIGUME` or `SIGRPE` signals, the following message is written into the system `syslog`, the NQS log file, and the job's log file:

```
Request <1.subzero>: Request received SIGRPE or SIGUME
signal; request requeued.
```

If NQS requeues a job because the job was terminated by the `SIGPEFAILURE` signal, the following message is written into the system `syslog`, the NQS log file, and the job's log file:

```
Request <1.subzero>: Request received SIGPEFAILURE signal; request requeued.
```

As a requeued job, the job will be reinitiated after it is selected by the NQS scheduler. The `qmgr schedule request now` command can be used to force the job to be reinitiated immediately.

The following actions can be expected when a job is terminated by either a `SIGRPE` or `SIGUME` signal or a `SIGPEFAILURE` signal (UNICOS/mk systems only):

- For a job that has default rerun and restart attributes, the job is requeued and rerun.
- For a job that has default rerun and restart attributes and has a restart file associated with it, the job is requeued and restarted from the restart file.
- For a job that has the no-rerun attribute and has no restart file, the job is deleted.
- For a job that has the no-rerun attribute but does have a restart file, the job is requeued and restarted from the restart file.
- For a job that has the no-restart attribute and uses the default rerun attribute, the job is requeued and rerun.
- For a job that has the no-rerun and no-restart attributes, the job is deleted.

NLB Administration [7]

The Network Load Balancer (NLB) is comprised of servers, collectors, and clients. The NQE GUI `Status` function uses collectors that communicate with the NLB server to obtain batch request status information.

This chapter describes NLB administration. The following topics are discussed:

- Overview of NLB
- Overview of NLB configuration
- Editing NLB configuration file
- Configuring ACLs for the NLB
- Starting the NLB server
- Specifying the NLB server location
- Creating redundant NLB servers
- Starting NLB collectors
- Managing the NLB server, including shutdown

The *NQE User's Guide*, publication SG–2148, describes the NQE GUI `Load` window, which allows you to monitor machine load and system usage across the complex.

For information about the NQE database and NQE scheduler, see Chapter 9, page 231.

7.1 NLB Overview

The Network Load Balancer (NLB) server contains the NLB database for NQE information. It stores this information in the form of *objects*. A *policy* is the mechanism by which the NLB server selects destinations based on the information in the NLB database. (Policies are described in Section 8.1, page 193.)

Each object in the NLB database is uniquely identified by a combination of its *type* (the definition of the object) and its *name*. All objects of a specific type have unique names to differentiate them; these object names are case insensitive. The following example describes one object in the NLB database:

```
object NLB (host1)
```

The object type is `NLB` and the name of the object is `host1`.

Each object contains a set of *attributes*. An attribute consists of unique *attribute names*, which identify it and its corresponding *attribute value*. Attribute names are defined when the object type is defined; attribute names are case insensitive. The object type and attribute names are defined in the *name map*.

The following is an example of an object with one of its attributes:

```
object NLB (host1) {
      NLB_HARDWARE = "Sun4c"
}
```

The NLB object named `host1` contains an attribute called `NLB_HARDWARE`, which is set to `Sun4c`. For a complete list of attributes for an object type, use the following command:

```
nlbconfig -mdump -t object_type
```

Although the NLB database can store any defined object, within NQE only the following object types are defined and used:

Object type	Description
`NLB`	Data sent by the collector processes containing information about the NQE server that may be used for destination selection. Attributes of this object include such things as CPU load.
`NJS` and `NJS_ALIVE`	Data sent by collector processes describing NQS requests on the NQE server. This is used by the NQE GUI `Status` function. `NJS_ALIVE` tells you the last time the NLB hear from a collector; it is used by the NQE GUI `Load` function to indicate an NQE server is no longer talking to the NLB. It may also be used for destination selection.
`FTA_CONFIG` and `FTA_DOMAIN`	FTA configuration objects. For information on defining these objects, see Section 13.2.10, page 335.

`NLB_JOB`	Data sent to the NLB about the request, which the NLB uses to choose a destination for the request.
`C_OBJ`	Configuration for the NLB itself.

7.2 Overview of NLB Configuration

The NLB has the following configuration requirements:

- The NLB server can be run from any account unless it is configured to use a privileged TCP/IP port number.
- The server requires a working directory and free space for data storage. This directory holds configuration files used by the server; it also stores a permanent copy of the NLB database. To run 750 requests, 250 Kbytes should be ample space.
- There are two configuration files, as follows:
  - `config`, which defines some constants of operation and the master ACL used to control administrative access to the server. It is read at startup time.
  - `policies`, which contains the definitions of policies. This file also is read at startup, but the servers reads it again when the administrator issues the `nlbconfig -pol` command. This file is described in Section 8.2, page 194. NQS is shipped with a policy named `nqs`, which selects batch queues based on which system has the highest average idle CPU load.

The location of these files is defined in the `NLB_SERVER_DIR` attribute of the `nqeinfo` file.

7.2.1 Server Database Integrity

The NLB server database is held in memory, but it is periodically backed up to disk if it has been modified. The update interval is configurable; the default is 30 seconds. See the `SKULK` field in the NLB configuration file, described in the following section. The NLB database also is written out to disk on server shutdown. When the server restarts, it reads the NLB database from disk.

Access control lists (ACLs) are read and written in the same manner as the NLB server database. (For a description of ACLs, see Section 7.9.5, page 186.)

The restart sequence, in combination with collectors, provides a high degree of consistency between NLB database contents and the information sent to the server.

7.3 Editing the NLB Configuration File

The NLB configuration file sets the configurations that cannot be changed while the NLB server is running. To change these settings, edit the NLB configuration file and then restart the NLB.

The NLB configuration file, `$NLB_SERVER_DIR/config` (where `NLB_SERVER_DIR` is defined in the `nqeinfo` file), is an ASCII text file that has the following format:

KEY:value

The *KEY* field defines the parameter being controlled; *value* gives the setting of the parameter. The interpretation of *value* depends on the *KEY*. Any text after a # symbol is a comment; spaces or tabs can be used anywhere in the file. There are defaults for all fields.

KEY	Description of value
`OBJECTS:`	Number of different object types the server will allow. The default is 32.
`PORT:`	TCP/IP port number or service name on which the server should listen for connections. The default is `nlb`.
`SKULK:`	The interval (in seconds) between NLB database write operations to disk. The server checks for NLB database modifications every *value* seconds, and it writes out those object lists or ACLs that have changes. The default is 30 seconds.
`M_ACL:`	The record to add to the master ACL. This determines who is allowed access to the ACL objects. This value contains three fields: the user name, the host name, and the privileges to be granted. The host name can be used to allow the user access from any location. If no `M_ACL` record is present, the ACL defaults to the user who started the server on the same host as the server.
`D_ACL:`	Record to add to the default ACL for a new object. This value has the same format as the `M_ACL` record. If it is not specified, there

is no default ACL, and any user is allowed access to any object except ACL objects. Read access is allowed for the ACL for the `C_OBJ` object type, but no access is allowed for any other ACLs.

`DBASE:`	Path name of the directory used for object storage. You can override it by using the `nlbserver -d` option.
`OBJ:`	Controls the storage class for an object, which you can configure by using the `nlbconfig` command.

Two fields are required: the object ID and the storage class. Storage class is one of the following:

`persistent`	Objects are written to disk at the rate controlled by `SKULK`.
`permanent`	Objects are written to disk when the server is shut down.
`transient`	Objects are never written to disk.

The following is an example of a configuration file:

```
#   NLB server configuration file - master and default ACLs
#
#   OBJECTS:        - defines maximum number of different object types
#   SKULK:          - interval in seconds between database update
#   PORT:           - TCP service name or port number to listen on
#   M_ACL:          - record for master ACL (can be repeated)
#   D_ACL:          - record for default ACL (can be repeated)
#
#   If M_ACL is not defined then it defaults to the user who
#   started the server on the same machine. The master ACL
#   controls the editing of other ACLs.
#
#   If D_ACL is not defined the default ACL is empty which
#   means objects are not restricted. If there is an ACL for an
#   individual object type it overrides the default ACL
#

OBJECTS:        32                # number of different objects
PORT:           nlb               # tcp service name
SKULK:          30

OBJ:            500     permanent
OBJ:            501     permanent

#               User    Host            Privs
#               ----    ----            -----

M_ACL:          root    *               all
M_ACL:          fred    *               all
```

7.4 Configuring ACLs for the NLB

Access to data in the NLB is controlled through Access Control Lists (ACLs). Each object type may have its own ACL. If no ACL is defined for an object type, then the default ACL (D_ACL) is used for objects of that object type. If there is no ACL for an object type and there is not an ACL, then access to any object of that object type is unrestricted. The ability to access the ACLs is controlled by the master ACL (M_ACL).

The M_ACL and the D_ACL are controlled by the config file in NLB_SERVER_DIR (as defined in the nqeinfo file) and may not be changed without restarting the NLB.

The following values are supplied with the initial installation of NQE:

```
#    User    Host        Privs
#    ----    ----        -----
#
M_ACL:        root    *               all
M_ACL:        WORLD   *               r
D_ACL:        root    *               all
D_ACL:        OWNER   *               all
D_ACL:        WORLD   *               r
```

With the initial configuration, only root is able to change ACLs; everyone else, indicated by the keyword WORLD, has read-only access. Access may be granted for specific hosts, but an asterisk (*) is a wildcard indicating root access, regardless of the NQE server. If no ACLs are defined, the default ACL is used, and either root or an object's owner may change the object. Everyone else may view the object but not change it (read-only access). The types of privileges are: d (delete the object), u (update (change) the object), and r (read the object).

For information about editing the NLB configuration file, see Section 7.3, page 176.

For information about changing ACLs for different object types, see Section 7.9.7, page 189.

7.5 Starting the NLB Server

To start the NLB server, use the following command:

```
nlbserver
```

The nlbserver command automatically runs in the background; you will receive the system prompt before the command completes execution.

7.6 Specifying the NLB Server Location

All programs that communicate with the NLB server (the collector, the NQE GUI Status and Load functions, and the nlbconfig and nlbpolicy clients) can use the same set of mechanisms to locate the server. These mechanisms are as follows; they are listed in order of precedence:

1. The command line
2. The `NLB_SERVER` environment variable
3. The value set in the `nqeinfo` file at installation
4. The `.Xdefaults` file

In each of these cases, the syntax is as follows. For one server, the syntax is:

"*hostname*[:*service_name*]"

If you have multiple servers, specify multiple *hostname*[:*service_name*] pairs; commas are not allowed, use a space to separate the servers. The syntax is as follows:

"*hostname*[:*service_name*] *hostname*[:*service_name*] *hostname*[:*service_name*]"

The *hostname* is the Internet host name of the server host. The optional *service_name* can be either a name that is mapped to a port number or an ASCII representation of the port number. The service name is the string contained in the configuration file (see Section 7.3, page 176) as a value for the keyword `PORT`. The default value of *service_name* is `nlb`. Use quotation marks on the command line or when setting the `NLB_SERVER` environment variable. The following are examples of `NLB_SERVER` syntax:

```
"cool:nlb_server"
"ice:10001"
"hot:604 wind rain gust:705"
```

If you place the environment variable `NLB_SERVER` in the system startup file on a host, it will be available for all NLB commands. Specifying a server on a command line will override the environment variable.

In the case of the collector, data is sent to all servers listed. For all other programs the servers are tried in turn until one responds. If a server exits, the clients will automatically switch to the next specified server.

Note: If you will send data to multiple NLB servers and will use job dependency, read Section 12.2, page 314.

7.7 Creating Redundant NLB Servers

Running redundant NLB servers ensures that all of the NQE nodes can still communicate with an NLB server, even if a network failure occurs.

When you install NQE collector nodes, specify all of the hosts on which you will run NLB servers so that collectors can send data to all NLB nodes.

You can set the `NLB_SERVER` environment variable to point to several hosts. The first host specified is the first one queried for information. If the first host does not respond, NQE tries the second host, and so on. For more information on setting environment variables, see Chapter 3, page 35.

Warning: The job dependency feature does not provide synchronization of events over multiple NLB servers. If you will use job dependency and redundant NLB servers, you must read the information about job dependency in Chapter 12, page 313.

7.8 Starting NLB Collectors

An NLB collector should be configured to run on each NQE node on which NQS is configured to run. The collector extracts various statistics from the system at regular intervals and sends them to the server.

Note: If you will send data to multiple NLB servers and will use job dependency, read Section 12.2, page 314.

A number of attributes are collected (see Section 8.5, page 204). For those attributes that collect dynamic statistics (such as `NLB_PHYSMEM`), both the current value and a rolling average are collected. The rolling average is the value of the attribute over the last *n* seconds; *n* is specified with the `ccollect -r` option.

You can start the collector from any directory. To start the collector, use the following command:

`ccollect` *option*

The `ccollect` command automatically runs in the background; you will receive the system prompt before the command completes execution.

The *option* can have the following meanings:

Option	Description
`-a`	Specifies that you want to use `awk`(1) to parse request (`qstat`) information instead of getting the information directly from the NQS database, which is the default source of information. To customize how data is parsed, use this option in conjunction with the `-f`, `-p`, and `-q` options.
`-c`	Specifies whether NLB will have a permanent TCP/IP connection to the server. The default is to establish a new connection for each update. Even though new connections mean that the server uses more CPU time, they reduce the number of connections the server needs to have open.
`-C` *filename*	Specifies the full or relative path name of a file that contains custom objects that are read by the collector and then sent to the NLB server. If the path is relative, the directory search order is the current working directory and then the /*nqebase*/`etc` directory (which is `$NQE_ETC`, as defined in the `nqeinfo` file), or the `$NQE_ETC/config` directory. You can specify this option multiple times. You also can set this option by using the `NQE_CUSTOM_FILE_LIST` variable either in the `nqeinfo` file or in the environment. If the variable exists in both places, the value in the environment takes precedence. The variable mechanism syntax lets you specify multiple files by separating them with either a space or a comma. The search order is the same as above. Files used with the `-C` option are read when the collector performs an update, and the values for the attributes are sent to the NLB server.
`-d` *file_system*	Reports the kilobytes of free disk space to the NLB server. The `-d` option adds the following attributes to the `name_map` file: `NLB_TMPNAME` Directory name

`NLB_FREETMP`	Free space in the file system (in kilobytes)
`NLB_A_FREETMP`	The rolling average of `NLB_FREETMP`

`-f` *parsefile*	Specifies the name of a file used to parse `qstat` information. The default is `njs_qstat.awk`. When you use this option, you also must use `-a`.
`-i` *interval*	Specifies the interval (in seconds) at which the collector gathers machine load information. The default is 15 seconds.
`-I` *interval*	Specifies the interval (in seconds) at which the collector gathers information about NQS batch requests. The default is 30 seconds.
`-J`	Starts a collector that collects only job status statistics.
`-L`	Starts a collector that collects only machine load statistics.
`-o` *filename*	Used at collector startup time only. Specifies an object definition file that contains extra attributes to be sent to the server from this collector. Specifying additional attributes allows a site to use its own attributes in displays and policies (for example, a site might write a policy that uses a weighted value based on the number of CPUs).
`-p "`*parse_command*`"`	Specifies a pattern-scanning command used to parse the output of a `qstat` command. When you use this option, also you must use `-a`.
`-q "`*qstat_command*`"`	Specifies a `qstat` command to be used in gathering network job status statistics. This option is used if the `qstat` command is not in the collector's `PATH` environment variable. This must be a full path name. When you use this option, you also must use `-a`.
`-Q` *queue*	(Required) Specifies the host's `NLB_QUEUE_NAME` attribute in the NLB database, which defines the queue name NQS uses when it sends load-balanced requests to this machine. If you are

	using the default NQE queue configuration, this queue is `nqebatch`.
`-r` *interval*	Specifies the interval (in seconds) over which the rolling averages of attributes are calculated. This value is turned into a multiple of the update interval. A value of 0 turns off the rolling average attributes. Rolling average attributes are provided to prevent load-balancing decisions from being influenced by sudden spikes in the various measurements. However, the longer the time period over which the averages are taken, the longer it will take the load balancer to note changes in a machine's current load. For more information on averaging data, see Section 8.3.5, page 198.
`-s` *server*	Specifies the name of the NLB server to which to send information. If you want to specify that multiple servers will be updated from this collector, you either can specify the `-s` option more than once, or you can specify the option once but place the server names in quotation marks. If you omit the option, the `NLB_SERVER` environment variable is used to locate the server.

The default mechanism for gathering batch request information is to read it directly from the NQS database. To customize the method by which data is collected and stored, use the `-a`, `-f`, `-p`, and `-q` options.

If you use the `-a` option, batch request information is parsed by using the `qstat -af | awk -f njs_qstat.awk` command.

To change the path name of the `qstat` command, use the `ccollect -q` option. You may want to do this if `qstat` is not in the collector's `PATH` environment variable. You must specify a full path name. If you use this option, you also must use the `-a` option.

To change the `awk` command used to parse the `qstat` output, use the `-p` option. For example, you may want to use `nawk`(1). If you use this option, you also must use the `-a` option.

To change the file name used to parse `qstat` information, use the `-f` option. You can customize the script `njs_qstat.awk` to parse the data the collector gathers so that it suits your site's needs. For example, you can modify the script so that two different collectors report information to two different NLB servers, thus keeping user communities separated.

To customize NLB collectors, see Section 8.8, page 226.

7.9 Managing the NLB Server

The `nlbconfig` facility lets you maintain and administer the NLB server. It provides the following capabilities:

- Shutdown an NLB server
- Instruct an NLB server to reread its policy file (the file name is `policies`)
- Display, write, and modify name map information
- Display, write, and modify object information
- Display, write, and modify ACL information
- Change storage class of objects

The following sections describe the `nlbconfig` command.

7.9.1 Specifying the Server on the Command Line

To specify a server on the command line, use the following command:

`nlbconfig -s` *server*

The *server* is the name of the host on which the NLB server resides. The server also can be specified in the `nqeinfo` file or with the `NLB_SERVER` environment variable (as described in Section 7.6, page 179). Specifying `-s` *server* overrides the value of `NLB_SERVER`. If you do not specify a server name and it is not set by any of the methods listed in Section 7.6, page 179, the nlbconfig command will fail with the following message:

```
nlbconfig:  cannot determine server location
```

7.9.2 Verifying the NLB Server Is Running

To ensure that the NLB server is running, use the following command:

```
nlbconfig -ping
```

The response you receive tells you whether the NLB server is running. If the server is running, the response tells you the name of the server that was contacted when you issued the command.

7.9.3 Shutting down the Server

To shut down the server, use the following command:

```
nlbconfig -kill
```

7.9.4 Rereading the `policies` File

To reread the `policies` file and make any new information in it available to the NLB, use the following command:

```
nlbconfig -pol
```

Note: The `policies` file that NLB reads when you issue an `nlbconfig -pol` command is in the location defined in the `nqeinfo` file `NLB_SERVER_DIR/policies`. Do not confuse this file with the `policies` file located in the `/`*nqebase*`/etc` directory, which is only used when the NLB directory is created. If you make changes to the policies file, make them to the `NLB_SERVER_DIR/policies` file.

7.9.5 Configuring Name Maps, Objects, and ACLs

The `nlbconfig` command uses similar options to operate on name maps, objects, and access control lists (ACLs).

You can list and dump the contents of the name map, object, and ACL files the server is currently using. When you list information, you receive basic information about the type of objects in the server. When you dump information, the current values of the objects are formatted and displayed. The following options perform these operations:

Option	Description
`-mlist`	Lists names, types, and storage classes in the name map file

`-olist`	Lists names of objects
`-alist`	Lists names of an ACL list
`-mdump`	Dumps values in name map file
`-odump`	Dumps values in the object file
`-adump`	Dumps values in the ACL list

The following options replace the contents (or part of the contents) of the name map, object, and ACL files.

Option	Description
`-mput`	Downloads a name map file to the server
`-oput`	Downloads the specified object(s) to the server
`-aput`	Downloads the specified ACLs to the server

The following options delete objects or ACLs in the server's files:

Option	Description
`-odel`	Deletes the specified object(s)
`-adel`	Deletes the specified ACLs
`-mdel`	Deletes the name map for the specified object types from the server

The following option adds attributes to objects in the server's files, leaving all the current attributes unchanged:

Option	Description
`-oupdat`	Updates the specified object(s) with additional attributes

The following options change the storage class of an object in the name map file (see page Section 8.5.1, page 204 for more information about storage classes):

Option	Description
`-perm`	Changes the storage class of the specified object to permanent
`-pers`	Changes the storage class of the specified object to persistent

`-tran`	Changes the storage class of the specified object to transient

Options are provided to qualify the options already described. For example, when you want to download a name map file, you use the `-f` option with the `nlbconfig` command to specify the file name of the new file. The following qualifying options are provided (for definitions of objects, object names, and attribute names, see Section 7.1, page 173):

Option	Description
`-f` *file*	Specifies a file name that contains objects or ACL descriptions.
`-t` *type*	Specifies an object type or ACL type.
`-h` *host*	Specifies an ACL host name; the default is `"*"` (denoting all names).
`-n` *name*	Specifies an object name.
`-a` *attribute*	Specifies an object attribute name or a set of ACL privileges.
`-u` *user*	Specifies an ACL user name.

The following list indicates the options that can be qualified:

Option	Qualifier
`-adel`	Object type, user, and host
`-adump`	Object type and output file
`-alist`	Object type and output file
`-aput`	Object type, object attribute, user, host, and output file
`-mdel`	Object type
`-mdump`	Object type and output file
`-mlist`	Object type and output file
`-mput`	Input file
`-odel`	Object type and object name
`-odump`	Object type, object name, object attribute, and file
`-olist`	Object type, object name, and file
`-oput`	Input file
`-oupdat`	Input file

`-perm`	Object type
`-pers`	Object type
`-trans`	Object type

The `-kill`, `-pol`, and `-ping` options do not use qualifiers.

7.9.6 Examples of Configuring Name Maps and Objects

The following example lists the name map for the server defined by the `NLB_SERVER` environment variable:

```
nlbconfig -mlist
```

This command produces the following output:

```
object type     id          storage class
-----------     ----        -------------
NLB             1024        persistent
NJS             500         permanent
C_OBJ           1           persistent
NJS_ALIVE       501         permanent
FTA_CONFIG      400         persistent
FTA_DOMAIN      401         persistent
FTA_DATA        402         persistent
FTA_ACTION      403         persistent
NLB_JOB         1025        persistent
NLB_POL         3           persistent
EVENT           601         transaction
```

The following example deletes the object of type `C_OBJ` with the name `gust` on the server defined (to locate the server, use the mechanisms described in Section 7.6, page 179):

```
nlbconfig -odel -t C_OBJ -n gust
```

7.9.7 Changing ACL Permissions

ACLs control permissions for reading, updating, or deleting objects in the NLB server. ACL permissions are usually changed with the `nlbconfig` command.

The following example adds an ACL to the ACL list that allows user `mario` to read (`r`) all objects of type `C_OBJ` from any host (the default) on the server defined by the `NLB_SERVER` environment variable:

```
nlbconfig -aput -t C_OBJ -u mario -a r
```

The master ACL controls all other ACLs; only users in this ACL can edit other ACL lists with `nlbconfig`. If you receive the following error message, you need to change the master ACL:

```
# nlbconfig -aput -t C_OBJ -u mario -a r
gust: No privilege for requested operation
```

To change the master ACL, edit the `$NLB_SERVER_DIR/config` file and restart the NLB; you cannot change it by using `nlbconfig`.

By default, all users have read permission for ACL information in the server database. The keyword `WORLD` is used to specify all users. If you changed permissions for any users, you must explicitly add `WORLD` read permission again.

To modify data in the server database, the NLB collector account (that is, the account from which the `ccollect` command is issued) must have `u` (update) and `d` (delete) permission.

All users can monitor all requests (through the NQE GUI `Status` window) in the NQE cluster.

To avoid having to issue a long list of commands, you can place the information in a file and then download that file to the server. The following command places the current ACL for NJS objects in a file named `acl_njs`:

```
nlbconfig -s cool -alist -f acl_njs
```

The `acl_njs` file has the following format:

```
type            user            host            priv
----            ----            ----            ----
NJS             luigi           *               dru
NJS             root            *               dru
NJS             mario           *               dru
NJS             WORLD           *               r
```

All jobs in the NQE database will be seen, regardless of how you set you ACL.

The following `nlbconfig` command places the information in the server:

```
nlbconfig -aput -f acl_file
```

Implementing NLB Policies [8]

This chapter describes the NQE Network Load Balancer (NLB) policies and their implementation. The following topics are discussed:

- Defining load-balancing policies
- The `policies` file
- Examples of how policies are applied
- Testing policies
- File formats
- Implementing NQS destination-selection policies
- Using batch request limits in policies
- Storing arbitrary information in the NLB database (extensible collector)

8.1 Defining Load-balancing Policies

A *load-balancing policy* is the mechanism whereby the NLB server determines a list of hosts (with the most desirable first) to be used as the target for a batch request. For example, when a request is submitted to a load-balanced queue, the NLB server applies a policy to the current information it has about hosts in the complex. The server provides NQS with a list of machines sorted by the specifications of the policy.

Policies are filters and sort routines that are applied to destination selection. The information in the NLB database objects is used to determine how to filter and sort the destinations (that is, how to balance the load across the NQE.) Policies consist of a set of Boolean and arithmetic expressions that use kernel statistics collected from the target machines. These expressions are defined in the `policies` file of the server and you can reread the file using the `nlbconfig`(8) command. Multiple policies can exist at any one time.

Some attributes are reserved by the NLB, and others can be defined by a site.

Sites have different requirements for the way they want work to be assigned to machines, and the workload in an NQE complex is often varied; therefore, no single set of rules need be successfully applied. Sites have the ability to determine their own policies based on their unique needs.

8.2 The `policies` File

When the system is installed and initially configured, the NLB reads the `$NQE_ETC/policies` file for the initial configuration and writes it to the `$NLB_SERVER_DIR/policies` file. After that, only the `$NLB_SERVER_DIR/policies` file is read. You should define any local, site-defined policies in the `$NQE_SPOOL/policies` file.

Note: The `policies` file that NLB reads when you issue an `nlbconfig -pol` command is located by default in `$NLB_SERVER_DIR/policies`. Do not confuse this file with the `policies` file located in the `/`*nqebase*`/etc` directory, which is used only on NQE start-up (after an install) if `$NLB_SERVER_DIR/policies` does not exist. If you make changes to the policies file, make them to the file in `/usr/spool/nqe/nlbdir`.

The template for a policy definition is as follows:

```
# name is the mechanism for selecting an individual policy
policy: name

# A host must satisfy all specified constraints to be
# selected as a destination by the policy. There can be
# zero, one or many constraints.
constraint: Boolean expression
constraint: Boolean expression

# Hosts which meet all constraints are then sorted by an
# equation. The operation is applied to the attributes of
# each host in turn, the host producing the largest result
# is returned first.
sort: arithmetic operation

# Optional maximum number of results, prevents NQS from
# attempting to send the request to too many destinations.
count: integer
```

Multiple policies can exist in the file. The first policy is defined as the default policy, which is named `NQS` and is the one used if the client does not specify a policy by name or if it specifies a policy that does not exist.

The following Boolean operators are supported in all expressions in a policy:

Operator	Description
`||`	Logical `OR`
`&&`	Logical `AND`
`^`	Exclusive `or` (`a` or `b` but not `a` and `b`)
`!`	Logical `not`
`==`	Equal to
`!=`	Not equal to
`>=`	Greater than or equal to
`<=`	Less than or equal to
`>`	Greater than
`<`	Less than
`=>`	Implies. For example, `a => b` means that if `a` is true, `b` must also be true for the expression to be true. If `a` is false, the value of `b` is not examined and the expression is false.

Operator precedence is the same as in the C language. The `implies` operator (`=>`) has the lowest precedence; that is, the `=>` operation will be performed after all other operators.

The following functions also are available to you when defining load-balancing policies:

Function	Description
`[`*variable* `== "`*regex*`"]`	Tests that the string *variable* equals the regular expression (*regex*). This is a regular expression as understood by `grep` and `ed`, not the shell file name expansion form. Regular expressions are case insensitive.
`exists (`*attrib*`)`	Tests the existence of an attribute. The expression is true if the attribute exists and false if the attribute does not exist.
age	The number of seconds that have elapsed since this host's attributes were last updated. Used to filter out hosts that are not responding.

All expressions support `()`, `/`, `*`, `+`, and `-` operators. The attributes of the hosts are used as the variables in the policy expressions. NQS attributes specified by using the NQE GUI or by using the `-la` option of the `qsub` or `cqsub` command can also be used in policies.

Attributes provided at startup are described in Table 7, page 208. Additional static attributes can be defined and set using the `nlbconfig` command; these attributes can then be used in policies.

Note: The extensible collector allows you to add your own dynamic attributes. For additional information about the extensible collector, see Section 8.8, page 226.

8.3 Example of How Policies Are Applied

The following section describes how a policy for NQS load balancing is applied. For information about how to specify a policy name, see Section 5.6.4, page 65.

8.3.1 Step 1: The Name

After receiving a request in a load-balanced pipe queue, NQS will send a query to the NLB server. This query includes the name of the policy to be used.

The policy *name* is used to find the correct policy to apply. If the name is not that of a known policy, the default policy is applied. The default policy is the one named `NQS`.

NQS is shipped with a policy named `nqs`, which selects batch queues based on which system has the highest average idle CPU load.

8.3.2 Step 2: The Constraints

The first constraint is taken in and applied to the list of hosts that have sent data to the server by using the `ccollect` process (described in Section 7.8, page 181). It is applied to the objects of object type `NLB`. For a list of these objects, use the following command:

```
nlbconfig -olist -t NLB
```

A constraint is expected to be a Boolean expression, and a `true` result means that the host is kept as a valid target. A `false` result means the host is discarded.

Each subsequent constraint is applied to the hosts that met the previous constraints. This process continues until no more constraints are left, or until no hosts meet a constraint. If there are no acceptable hosts, NLB returns a null list, and NQS retries the request later. If there are no constraints, all hosts are assumed to be acceptable targets for the policy.

Constraints can filter out machines that physically cannot run a request, as in the following example:

```
# Policy for SPARC-specific requests
policy: SPARC
constraint: [ NLB_HARDWARE == "SPARC" ]
```

For the `nqs` policy, any machines that do not have a queue name associated with them are filtered out. It is also important to filter out machines that have not sent data recently. The time period used to evaluate this depends on the time interval between samples sent from the collectors. The following example sets the time period to 300 seconds:

```
constraint: exists(NLB_QUEUE_NAME) && age < 300
```

Constraints can also remove machines that are currently not good choices. For example, the following policy selects only machines that have more than 32768 Kbytes of free memory (32 Mbytes). Memory is measured in units of Kbytes:

```
policy: bigjob
constraint: NLB_FREEMEM > 32768
```

8.3.3 Step 3: The Sort

After the list of hosts has been filtered, the sort expression/operation is used to order them. The expression/operation is applied to each host in turn, and the result for that host is stored in the NLB database as the `NLB_POLICY` attribute. The hosts are then sorted on this number. The host with the largest `NLB_POLICY` value is regarded as the best choice.

8.3.4 Step 4: The Count

A policy can optionally specify a count of the maximum number of targets that should be returned. This is needed in large networks to reduce the number of different systems to which NQS might attempt to send a batch request.

8.3.5 Approaches to Designing Policies: Averaging Data

Policies are based on statistical data gathered from machines by NLB collectors. This data reflects the state of the machine in the recent past, but its current state may be different.

The data available to the server is a dynamic snapshot of the machine state; therefore, the server's picture of the state of machines in the network is always a little out-of-date.

This section describes a special collection of attributes, which, by convention, are named `NLB_A_`*regular attribute name*, such as `NLB_A_IDLECPU`.

Because an instantaneous snapshot of machine load could cause an incorrect view of its true state over time, the collectors provide two sets of measurements: the instantaneous data (which is of most use in the NQE GUI `Load` window displays) and averaged data for use in policies. The averaged data tends to smooth out the short-term peaks and troughs in the instantaneous data and to give a more accurate view of the overall load, as can be seen in Figure 16 and Figure 17.

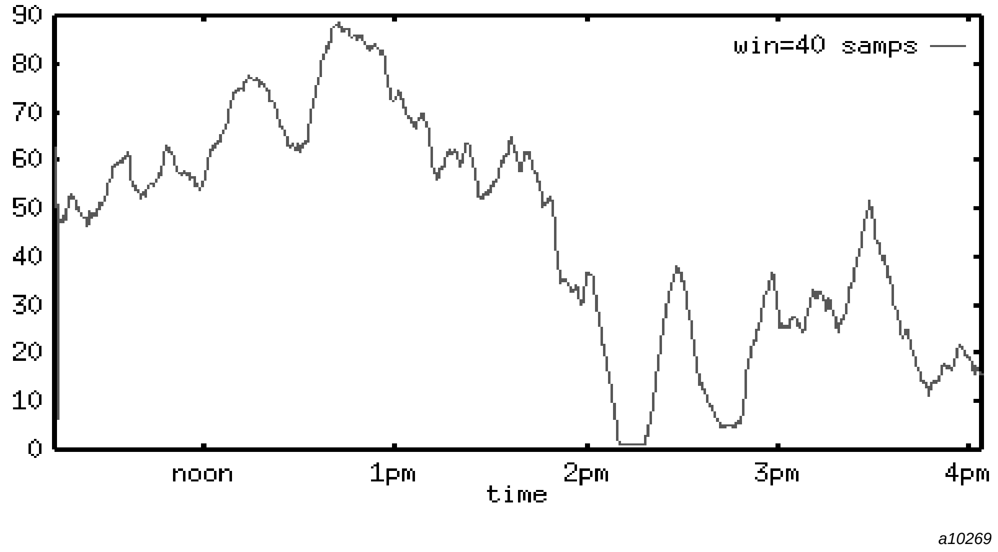


Figure 16. Average Machine Load Data

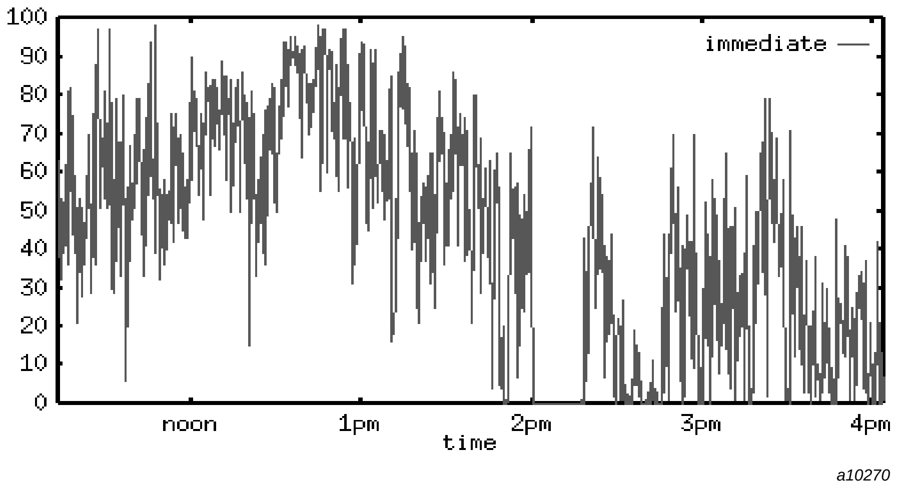


Figure 17. Instantaneous Machine Load Data

The averaged data (stored in attributes beginning with `NLB_A_`) are calculated by adding the instantaneous measures taken over the last *n* seconds and dividing by the number of samples taken (the `ccollect -r` option controls *n).*

You can tune the averaging of data in the policy to reflect the type of work being run on a machine. For example, you can user longer sampling periods on machines at which long requests are run. However, the longer the period you use, the longer it takes the measures to catch up with reality; for example, even after a large requests is finished, it takes time for the load it produced to work its way out of the data.

8.3.6 Other Factors: Weighting Machines

A network usually will contain a mix of machines: different types of machines and perhaps machines of the same type but differing capacity (such as CRAY T90 and CRAY J90 systems).

The measures sent by the collectors reflect what is actually happening on a particular machine, but they do not indicate how that activity compares to the maximum capacity of that machine, or how that machine compares to others in the network. To measure these factors, you must introduce weighting or normalization factors into policies.

For example, if you are concerned with the amount of idle CPU time available on a machine (which is reported as a percentage), you could write a policy that sorts on this factor, as follows:

```
policy: idlecpu
sort: NLB_IDLECPU
```

In this case, a small workstation may be selected as the best target for work in preference to a large Cray Research system, even though 10% idle time available on the Cray Research system could well represent much more computing power than 50% idle time on the workstation.

By adding weighting factors to machines, you can balance out this difference. If you define a new attribute called `NLB_SITE_CPUNORM` for each host (as in the example in Section 8.3.9, page 201), and assign values representing the relative CPU powers of the different machines (machine `A` is `100` while machine `B` is `10`), you can factor this into the policy, as follows:

```
policy: idlecpu
sort: NLB_IDLECPU * NLB_SITE_CPUNORM
```

Of course, the relative powers of different machines are related to the nature of the request to be executed. A highly vectorized application probably requires a larger difference in weightings between a workstation and a Cray Research system than would a scalar application.

8.3.7 Other Factors: Alternate Policies for Different Time Periods

You may want to apply different policies at different times of the day or the week. For instance, you may want to allow requests to run on file servers at night when there is little NFS activity. Although there is no direct support for this in the NLB, you could use a `cron` job to load a new set of policies into the server at defined times. This job would replace the `policies` file and cause the server to reread it, as in the following example:

```
cp policies policies.day
cp policies.night policies
nlbconfig -pol
```

The following example would switch the policies files back again:

```
cp policies policies.night
cp policies.day policies
nlbconfig -pol
```

For this example to work, `root` must be in the master access control list (ACL) for the server defined in its configuration file, because this ACL controls who can update the `policies` file.

8.3.8 Policy Examples

The following example selects all hosts that have the attribute `NLB_QUEUE_NAME` and for which the data has been updated within the last minute (`AGE < 60`). The hosts are then sorted by weighted idle CPU time (as described in Section 8.3.6, page 199).

```
policy:       AGE
constraint:   exists(NLB_QUEUE_NAME) && AGE < 60
sort:         NLB_A_IDLECPU * NLB_SITE_CPUNORM
```

The following example selects all machines with an average system load of 10%/CPU (or less); `NLB_A_SYSCPU` is the average system CPU (percentage) and `NLB_NCPUS` is the number of CPUs:

```
policy:        default
constraint: [NLB_HARDWARE == "CRAY"] =>
               (NLB_A_SYSCPU / NLB_NCPUS) < 10.0
constraint: [NLB_HARDWARE == "sun"] => NLB_PHYSMEM >= 32768
sort:          NLB_NCPUS
```

8.3.9 Example of Writing a Load-balancing Policy

The following example defines a new attribute used to normalize CPU loads. This is useful for a site with machines that have widely different CPU power. First a new object attribute is created, then values are assigned to the attribute, and finally the policy is added to the server NLB database so that data associated with it can be stored. All this can occur without any part of the system being restarted.

1. Define an attribute.

   For a new attribute to be usable, it must be placed in the server's `name_map` file.

To extract the name map from the server, use the following command:

```
nlbconfig -mdump -f name_map
```

This command produces a file containing all of the name maps known to the server. To define an attribute, you must edit this file to contain a name, description, and type for the new attribute, as in the following example:

```
map NLB=1024
{
NLB_SITE_CPUNORM "CPU Normalization Factor"     integer(1000);
existing attributes
.
.
}
```

The new name (`NLB_SITE_CPUNORM`) and the new ID (`1000`) must not match any existing ones. The attribute must appear in the list of attributes for the NLB object (the top line of the example), and all existing attributes must be included in the file. If you do not include all of the existing attributes, they will be deleted when the new `name_map` file is read by the server.

2. Add the attribute to the name map by using the following command:

   ```
   nlbconfig -mput -f name_map
   ```

   You have to do this only once; the server stores this data in its NLB database until you explicitly overwrite it again.

3. Add a value for the new attribute to the host data by creating an object definition that sets the value of the attribute for each NLB object, as follows:

   ```
   object NLB (bighost) {
            NLB_SITE_CPUNORM = 1000;
   }
   object NLB (smallhost) {
            NLB_SITE_CPUNORM = 1;
   }
   ```

   This file is then loaded into the NLB database by using the following command:

   ```
   nlbconfig -oupdat -f obj_defs
   ```

   The new attribute is now available for use in policies (in step 4).

4. Add a policy using the newly-defined attributes.

   After values for an attribute have been placed in the NLB database (step 3), you can write policies based on these values. The following policy could be placed in the `policies` file to select hosts based on CPU normalization:

   ```
   policy: idlecpu
   sort:   NLB_IDLECPU * NLB_SITE_CPUNORM
   ```

5. Issue the following command to cause the server to reread its policy file:

   ```
   nlbconfig -pol
   ```

   This policy is now available for use by NQS.

   Note: Ensure an NQS load-balancing queue is defined to use the new policy.

8.4 Testing Policies

The load-balancing policies that work best for your site depend on the workload at your site. To help you develop policies, the `nlbpolicy` program is provided as a direct interface to the NLB. You can, for example, experiment with a policy without submitting requests to NQS. The syntax of the `nlbpolicy` command is as follows:

`nlbpolicy [-a` *attribute = value*`] [-c` *count*`] [-d] [-h` *host*`] [-p` *policy*`]`
`[-s` *server*`]`

The `nlbpolicy` command sends the server a policy query. The `-a` option specifies a list of attribute names and an optional value to find a host with the specified attribute set, or the specified attribute set to the specified value. You may specify more than one attribute and corresponding value. The *value* must be a decimal integer or float, in normal or scientific notation. You must expand (to a value in bytes) unit symbols to be acceptable to NQS commands, such as 10kw (for example, 10kw would be 10 x 8192 = 81,920).

The `-c` option controls the maximum number of hosts returned. The `-p` option selects a policy by name. The `-h` option (which can be repeated) specifies an acceptable target host to return.

Without the `-d` option, the program prints out the returned hosts ordered by the policy sort equation and the result of the equation for that host. With the `-d` option, all the information about each host is printed as an object data file.

The following policy, called `AGE`, returns a list of hosts ordered by the percentage of idle CPU time multiplied by a weighting factor:

```
policy:AGE
sort: NLB_A_IDLECPU * NLB_SITE_CPUNORM
```

To test the policy, use the following command:

```
nlbpolicy -p AGE
```

The following output is returned:

```
cool                 5.000000
cloudy               0.701754
gust                 0.239234
hot                  0.028249
```

The following command uses the same policy, but the `-d` option dumps all of the objects for the host, and the `-c` option limits the number of hosts to 2:

```
nlbpolicy -p AGE -d -c2
```

This command prints out all of the attributes of the selected hosts (objects of object type `NLB`).

8.5 File Formats

The following two sections describe the NLB name map and the object data file formats.

8.5.1 The Name Map File

The NLB server supports mappings between the internal values used to represent objects and attributes, and the human-readable names and descriptions. The mapping is defined by configuration data held within the server. This data can be downloaded from the server and converted to a text form by `nlbconfig`. The `nlbconfig` program also can take the text form of the name map and load it back into the server, allowing you to add new definitions to the map. This text form of the name map is known as a `name_map` file.

The `name_map` file contains a sequence of definitions for different object types and their associated attributes. Each object has a name and an ID. Each attribute of an object has a name, a text description, and an ID formed from a type and a number. The attribute ID and name must be unique from all other attributes of that object.

A name map for an object of object type NLB (information about an individual host) would be as follows; column one is the attribute name, column two is the attribute description, and column three is the attribute type ID:

```
map NLB = 1024
{
         NLB_OSNAME             "OS name"                      string(1);
         NLB_RELEASE            "Release"                      string(2);
         NLB_VERSION            "Version"                      string(3);
         NLB_HARDWARE           "Hardware"                     string(4);
         NLB_QUEUE_NAME         "Queue name"                   string(5);
         NLB_TMPNAME            "Temp file path"               string(6);
         NLB_BSUSPEND           "Auto suspend state"           string(7);
         NLB_IDLECPU            "Idle cpu"                     float(1);
         NLB_RUNQLEN            "Run queue"                    float(2);
         NLB_SWAPPING           "Swap activity"                float(3);
         NLB_POLICY             "Policy Weighting"             float(4);
         NLB_SYSCPU             "System cpu"                   float(5);
         NLB_RUNQOCC            "Run queue occupancy"          float(6);
         NLB_A_IDLECPU          "Ave Idle cpu"                 float(11);
         NLB_A_RUNQLEN          "Ave Run queue"                float(12);
         NLB_A_SWAPPING         "Ave Swap activity"            float(13);
         NLB_A_SYSCPU           "Ave System cpu"               float(14);
         NLB_PHYSMEM            "Memory"                       integer(1);
         NLB_FREEMEM            "Free memory"                  integer(2);
         NLB_CLOCK              "Clock speed"                  integer(3);
         NLB_NCPUS              "Num CPUs"                     integer(4);
         NLB_PSWCH              "Context switches"             integer(5);
         NLB_NUMPROC            "Process count"                integer(6);
         NLB_FREETMP            "Free temp space"              integer(7);
         NLB_KEYIDLE            "Keyboard idle time"           integer(8);
         NLB_SWAPSIZE           "Swap Device size"             integer(9);
         NLB_SWAPFREE           "Free Swap space"              integer(10);
         NLB_A_FREEMEM          "Ave Free memory"              integer(11);
         NLB_A_PSWCH            "Ave Context switches"         integer(12);
         NLB_A_NUMPROC          "Ave Process count"            integer(13);
         NLB_A_FREETMP          "Ave Free temp space"          integer(14);
         NLB_A_SWAPFREE         "Ave Free Swap space"          integer(15);
         NLB_IOTOTAL            "Total user I/O rate"          integer(16);
         NLB_A_IOTOTAL          "Ave Total user I/O Rate"      integer(17);
         NLB_IOREQS             "I/O Requests per sec"         integer(18);
         NLB_A_IOREQS           "Ave I/O Requests per sec"     integer(19);
         NLB_SYSCALLS           "System calls per second"      integer(20);
         NLB_A_SYSCALLS         "Ave system calls per second"  integer(21);
         NLB_TPDAEMONUP         "Tape Daemon running"          integer(22);
```

```
        NLB_MPP_UP              "Cray T3D available"            integer(23);
        NLB_NQS_UP              "NQS available flag"            integer(24);
        NLB_NQS_VALTYPE         "NQS validation type"           integer(25);
        NLB_TIMESTAMP           "Update time"                   time(4096);
        NLB_HOST                "Originating host"              string(4098);
}
```

You can change the attribute names (such as `NLB_OSNAME`) and the attribute descriptions. However, you cannot change the attribute type ID. The type ID numbers are built into `ccollect`(8); because of this, specific attributes always represent the same piece of information.

> **Note:** It is recommended that you do not change attribute names because they are used in X resource definitions for the NQE GUI `Load` window displays and in policy definitions.

Table 6 lists the allowed attribute types:

Table 6. Attribute Types

Type	Contents
`float`	Floating-point value
`integer`	Integer value
`string`	Null-terminated ASCII string
`time`	Time stamp (the number of seconds since 1 January 1970)

You can add new attributes and store values to the NLB database at any time by using the `nlbconfig` command. However, the data cannot be used anywhere until the attribute has been defined in the `name_map` file, even if objects in the NLB database possess the attribute. These site-defined values can then be used in policies and the X resource definitions for the machine load displays, which are displayed by using the NQE GUI `Load` window. For an example of writing a load-balancing policy, see Section 8.3.9, page 201.

A number of object and attribute names are reserved. They are hardcoded into the NLB collectors. A default name mapping file is supplied for these objects and attributes. Attribute numbers below `8192` are reserved for Cray Research. Table 7 lists the reserved host object attributes.

There are averaged versions of all integer and floating-point attributes except for `NLB_POLICY`, `NLB_KEYIDLE`, `NLB_PHYSMEM`, `NLB_CLOCK`, `NLB_NCPUS`, and `NLB_SWAPSIZE`. These attributes use the naming convention _A inserted in the middle of the name (for example, `NLB_A_IDLECPU` is the average version of `NLB_IDLECPU`).

Table 7. Reserved Host Object Attributes

Attribute	Description	Type
`NLB_BSUSPEND`	Status of interactive activity on a server running `csuspend`(8).	String
`NLB_CLOCK`	Clock speed.	Integer (picoseconds)
`NLB_FREEMEM`	Amount of memory currently available.	Integer (Kbytes)
`NLB_FREETMP`	Amount of free temporary file space.	Integer (Mbytes)
`NLB_HARDWARE`	Hardware type as shown by the `uname`(1) command.	String
`NLB_IDLECPU`	Percentage of idle CPU time.	Float (0 to 100)
`NLB_IOTOTAL`	Amount of I/O being performed by user processes (read and write system calls).	Integer (Kbyte/s)
`NLB_IOREQS`	Number of read and write system calls per second.	Integer
`NLB_KEYIDLE`	Amount of time since console key pressed or mouse moved. Only valid on workstations.	Integer (minutes)
`NLB_MPP_UP`	CRAY T3D MPP system available.	True/false
`NLB_NCPUS`	Number of configured CPUs.	Integer
`NLB_NUMPROC`	Number of processes in system at the last sample point.	Integer
`NLB_OSNAME`	Operating system name as shown by the `uname`(1) command.	String
`NLB_PHYSMEM`	Total amount of physical memory available to user processes.	Integer (Kbytes)
`NLB_POLICY`	Result of last policy sort equation applied to host.	Float

Attribute	Description	Type
`NLB_PSWCH`	Context switches per second.	Integer
`NLB_QUEUE_NAME`	Queue name for load-balanced NQS requests, defined by the `ccollect` option `-Q`.	String
`NLB_RELEASE`	Operating system release as shown by the `uname`(1) command.	String
`NLB_RUNQOCC`	Percentage of the time that the run queue was occupied (that is, processes were waiting for the CPU). A low number indicates the system is under used.	Float (0 to 100)
`NLB_RUNQLEN`	Number of runnable processes.	Float
`NLB_SWAPFREE`	Amount of unused swap space.	Integer (Mbytes)
`NLB_SWAPPING`	Rate of data movement to and from swap devices.	Float (Kbyte/s)
`NLB_SWAPSIZE`	Amount of swap space configured.	Integer (Mbytes)
`NLB_SYSCALLS`	Number of system calls executed per second.	Integer
`NLB_SYSCPU`	Percentage system CPU time.	Float (0 to 100)
`NLB_TMPNAME`	Path name of file system used for the `NLB_FREETMP` attribute, defined by the `ccollect` option `-d`.	String
`NLB_TPDAEMONUP`	UNICOS tape daemon is present.	True/false
`NLB_VERSION`	Operating system version as shown by the `uname`(1) command.	String

Table 8 lists the availability of the attribute data across platforms. The following conventions are used in the table:

Y	Indicates data is available
-	Indicates no kernel data is available

The `NLB_TPDAEMONUP` and `NLB_MPP_UP` attributes are collected only from UNICOS and UNICOS/mk systems.

Table 8. Attribute Implementation Across Platforms

Attribute	AIX	Digital UNIX	HP-UX	IRIX	Solaris	UNICOS	UNICOS/mk
NLB_OSNAME	Y	Y	Y	Y	Y	Y	Y
NLB_RELEASE	Y	Y	Y	Y	Y	Y	Y
NLB_VERSION	Y	Y	Y	Y	Y	Y	Y
NLB_HARDWARE	Y	Y	Y	Y	Y	Y	Y
NLB_QUEUENAME	Y	Y	Y	Y	Y	Y	Y
NLB_TMPNAME	Y	Y	Y	Y	Y	Y	Y
NLB_IDLECPU	Y	Y	Y	Y	Y	Y	Y
NLB_RUNQLEN	Y	Y	Y	Y	Y	Y	Y
NLB_SWAPPING	Y	Y	Y	Y	Y	Y	-
NLB_SYSCPU	Y	Y	Y	Y	Y	Y	Y
NLB_RUNQOCC	Y	-	-	Y	Y	Y	Y
NLB_PHYSMEM	Y	Y	Y	Y	Y	Y	Y
NLB_FREEMEM	Y	Y	Y	Y	Y	Y	-
NLB_CLOCK	-	-	-	Y	Y	Y	Y
NLB_NCPUS	-	Y	Y	Y	Y	Y	Y
NLB_PSWCH	Y	Y	Y	Y	Y	Y	Y
NLB_NUMPROC	Y	Y	Y	Y	Y	Y	Y
NLB_FREETMP	Y	Y	Y	Y	Y	Y	Y
NLB_KEYIDLE	-	Y	-	Y	Y	Y	Y
NLB_SWAPSIZE	Y	Y	Y	Y	Y	Y	-
NLB_SWAPFREE	Y	Y	Y	Y	Y	Y	-
NLB_IOTOTAL	Y	-	-	Y	Y	Y	Y
NLB_IOREQS	Y	-	Y	Y	Y	Y	Y
NLB_SYSCALLS	Y	Y	Y	Y	Y	Y	Y

Attribute	AIX	Digital UNIX	HP-UX	IRIX	Solaris	UNICOS	UNICOS/mk
`NLB_TPDAEMONUP`	Y	Y	Y	Y	Y	Y	Y
`NLB_MPP_UP`	Y	Y	Y	Y	Y	Y	Y

The NLB also stores attributes for running requests. There is an attribute for each field in the `cqstatl` or `qstat -f` display (over 200 attributes are included). To obtain a list of these attributes, use the following command:

```
nlbconfig -mdump -t NJS | more
```

The output from this command lists attribute names, descriptions, and types. An abbreviated example follows:

```
# nlbconfig -mdump -t NJS |more
###############################################
# creation date: Wed Mar 22 15:08:13 1995
# created by: ljgm
# object timestamp: Sat Mar 18 11:56:52 1995
########
map NJS = 500
{
NJS_NQSID                  "NQS identifier"                string(1);
NJS_JOBSTAT                "Job status"                    string(2);
NJS_NUMPROC                "Processes in job"              integer(3);
NJS_RUNUSER                "Running user"                  string(4);
NJS_QUENAM                 "Queue name"                    string(5);
NJS_CPUREQLIM              "Per-request CPU limit"         integer(6);
NJS_CPUREQUSE              "Per-request CPU use"           integer(7);
NJS_MEMREQLIM              "Per-request memory limit"      float(8);
NJS_MEMREQUSE              "Per-request memory use"        float(9);
NJS_PFSREQLIM              "Per-request PFS limit"         float(10);
NJS_PFSREQUSE              "Per-request PFS use"           float(11);
NJS_ORGHOST                "Originating host"              string(12);
NJS_ORIGUSER               "Originating user"              string(13);
NJS_QUEENT                 "Position in queue"             integer(14);...
```

8.5.2 The Object Data File

An object data file may be used to assign values to attributes for each object. Objects can be represented in a text format. Functions and commands are provided so administrators can create their own objects and insert them into the NLB database.

An object data file consists of a sequence of objects. Each object has a type, a name, and a sequence of assignment statements that define attribute values. The type names for objects and names for attributes are taken from the name map in the NLB database. The following example defines two attributes on three different hosts:

```
object NLB (cloudy) {
    NLB_SITE_CPUNORM = 10;
    NLB_APPLICATIONS  = "UniChem dbase";
}

object NLB (gust) {
    NLB_SITE_CPUNORM = 12;
    NLB_APPLICATIONS = "UniChem TurboKiva";
}
object NLB (hot) {
    NLB_SITE_CPUNORM = 24;
    NLB_APPLICATIONS = "";
}
```

The legal values for an attribute depend on its type as defined in the name map, as follows:

Type	Contents
`float`	Floating-point value. Only fixed-point representation is supported (such as `35000.0` and **not** `3.5e4`).
`integer`	Integer value. This can be a signed integer. A leading 0 implies octal.
`string`	A string surrounded by quotation marks (") or a single word containing only alphanumeric characters.
`time`	Cannot be specified as input.

First you would update the `name_map` file with the new attributes `NLB_APPLICATIONS` and `NLB_SITE_CPUNORM` and load it into the `nlbserver`. Note that because of how the server code works any attributes that are to be used in policies must be in either the NLB object or the `NLB_JOB` object. If any attribute is used in the policy file that is not in either of these two objects the policies file will not be read successfully by the `nlbserver`.

`name_map` file:

```
map NLB = 1024
{
NLB_APPLICATIONS
"Resident Applications"
string(999);
}
```

Second, the appropriate objects must have attributes set properly. For this example, these are set with the following object data file that is downloaded to the `nlbserver` with the following command:

```
nlbconfig -t nlb -oupdat filename
```

```
object NLB (gust) {
NLB_APPLICATIONS = "Unichem TurboKiva Nastran";
}
object NLB (wind) {
NLB_APPLICATIONS = "Nastran";
}
object NLB (rain) {
NLB_APPLICATIONS = "Unichem";
}
object NLB (frost) {
NLB_APPLICATIONS = "TurboKiva";
}
```

Third, write some policies that use the attribute in question.

Policy file:

```
policy:
first
constraint:
[NLB_APPLICATIONS == "Nastran"]
sort: (NLB_A_IDLECPU +5)/10000
policy:
second
constraint:
[NLB_APPLICATIONS == "Unichem"] ||\n  [NLB_APPLICATIONS == "TurboKiva"]
sort: (NLB_A_IDLECPU +5)/10000
```

Finally see the results of the policies when they are invoked.

Output from `nlbpolicy`:

```
nlbpolicy -p first
gust
0.0092525
wind
0.003213

nlbpolicy -p second
gust
0.009225
rain
0.004950
frost
0.000500
```

8.6 Implementing NQS Destination-selection Policies

It is relatively simple to set a policy that selects target NQS systems with the lightest CPU load.

To use this destination selection mechanism, batch requests must be submitted to a pipe queue that is configured to perform destination selection. By defining both standard NQS pipe queues and destination-selection pipe queues, and by setting an appropriate default batch queue, sites can define several workable environments that allow integration of destination selection into their environment.

The examples in Section 8.6.5, page 219, and Section 8.6.6, page 219, show how sites might configure NQS destination-selection queues to determine how batch requests are routed.

8.6.1 Configuring NQS Destination Selection

To configure an NQS destination selection environment, define a pipe queue as a destination selection pipe queue on each system that will route requests using NLB (as described in Section 8.6.5, page 219). No destinations should be defined for this pipe queue. The queue must be marked as a destination selection pipe queue (`CRI_DS`). (For additional information, see Section 5.6.4, page 65.)

The collectors on your NQE nodes have been reporting their availability. When you submit a request to the new queue, the NLB selects one of these servers as the destination queue for your job.

8.6.2 NQS User Interface to NLB

If the `qmgr` default batch queue is set to a destination-selection pipe queue, all requests that do not designate a specific queue are routed using NLB. If the default queue is not a destination-selection queue, you can use the `General Options Queue Name` selection of the `Configure` menu on the NQE GUI `Submit` window, use the `-q` *queues* option, or designate the queue in the script file to select a destination-selection queue.

If you are using file validation, each user must create a `.rhosts` or `.nqshosts` file on each system to which a request might be routed by the NLB. If you are using password validation, the user's password must be the same on all systems to which the NLB could route the request.

8.6.3 Policy Definition

The following example could be used as a policy definition in the NLB `policies` file:

```
# Policy for load-balancing between Cray systems
# Note that NLB_HARDWARE can be "CRAY J90", "CRAY Y-MP",
# "CRAY T3E", etc. and still match [ NLB_HARDWARE == "cray" ]
# ( AGE < 300 ) for systems that have sent data in the
#      last 5 minutes
# (NLB_PHYSMEM - NLB_A_FREEMEM....) for a ratio of memory used
#     (real and swap) to real memory of less than 4
# (NLB_NCPUS <= 2.... for less than 20% system CPU per CPU for
#      hosts with 1 or 2 CPUs
# (NLB_NCPUS > 2.... for less than 3.5% system CPU per CPU for
#      hosts with more than 2 CPUs
#
#Sorted by ratio of number of CPUs to length of the run queue
#
policy: RUNQUEUE
constraint:[NLB_HARDWARE == "cray"] && age < 300
constraint:(NLB_PHYSMEM - NLB_A_FREEMEM +
               (NLB_SWAPSIZE - NLB_SWAPFREE) * 1024)
                  /NLB_PHYSMEM < 4
constraint: NLB_NCPUS <= 2 => NLB_A_SYSCPU / NLB_NCPUS <20
constraint:(NLB_NCPUS > 2) => (NLB_A_SYSCPU /  NLB_NCPUS <3.5)
sort:        NLB_NCPUS / NLB_A_RUNQLEN
```

Destination selection can be used to limit requests when host resources reach a threshold. For example, the following policy omits any host with system CPU usage greater than 10%. This can keep a busy system from receiving more requests.

```
policy:      AGE
constraint: AGE < 300
constraint: NLB_A_SYSCPU < 10
sort:          NLB_A_IDLECPU * NLB_NCPUS
```

NQS destination-selection queues used for load balancing specify the policy for that queue. For this reason, you may want to have multiple load-balancing queues for requests with disparate resource requirements. For example, CPU-bound requests could be routed to a machine with idle CPU but little free memory. The following policies, `MEMORY` and `CPU`, serve this purpose. The first limits hosts to those with more than 32 Mbytes of memory and a low memory demand, calculated by the memory oversubscription. Hosts are then sorted so that larger memory machines are chosen first.

The CPU policy uses additional NLB attributes that would be added by the site. NLB_SYSCPU_LIMIT would be defined as an integer value that could be different for each host. NLB_RUNQLEN_LIMIT also could be different for each host. The CPU policy limits the system CPU time and the run queue length. Hosts are sorted so that those with idle CPU time are first, and those with more CPUs are preferred. If there is no idle CPU time, the hosts are sorted by the number of CPUs.

```
policy:      MEMORY
constraint: NLB_PHYSMEM > 32768
constraint: (NLB_PHYSMEM - NLB_A_FREEMEM +
               (NLB_SWAPSIZE - NLB_SWAPFREE) * 1024)
                 / NLB_PHYSMEM < 3
sort:          NLB_PHYSMEM

policy:       CPU
constraint: NLB_A_SYSCPU / NLB_NCPUS < NLB_SYSCPU_LIMIT
constraint: NLB_A_RUNQLEN / NLB_NCPUS < NLB_RUNQLEN_LIMIT
sort:         NLB_A_IDLECPU * NLB_NCPUS + NLB_NCPUS
```

8.6.4 NQS Processing

This section discusses NQS behavior when processing a batch request submitted to a destination selection queue.

When a batch request is submitted to a destination selection queue, it obtains a list of destinations from the NLB server. The list of destinations is tried one at a time until the request is accepted by one of the destinations or until the list is exhausted. If no destination accepts the request, NQS waits and tries again, based on the interval set by the `qmgr` command `set default destination_retry time` (the default is 5 minutes). NQS continues to try to deliver the request to a destination until the request is accepted or the request has waited the maximum allowable time, which is defined by the `qmgr` command `set default destination_retry wait` (the default is 72 hours). If no destination accepts the request during the maximum time, the request is deleted and the submitting user is notified by mail.

One of the following situations also may occur:

- If a destination selection pipe queue is defined incorrectly, the destination selection function is not invoked and the request is rejected. The user receives mail indicating that the request was submitted to a queue with no destinations.

- If an NLB server is not available at any of the hosts defined by `$NLB_SERVER`, a message is written to the NQS log file. No other attempt is made to route the request to a destination host. NQS continues to attempt to deliver the request until it is accepted or until NQS has waited the maximum allowable time.
- If the list of destinations is empty, a message is generated in the NQS log file. NQS continues to attempt to deliver the request until it is accepted or until the request has waited the maximum allowable time indicated by the `qmgr` command `set default destination_retry wait`.
- If no host in the list of destinations will accept the request, a message is generated in the NQS log file. NQS continues to attempt to deliver the request until it is accepted or until the request has waited the maximum allowable time indicated by the `qmgr` command `set default destination_retry wait`.
- If you are using file validation, each user submitting requests to a load-balanced pipe queue must have a `.rhosts` or `.nqshosts` file in their home directory for each batch system that could run work. These files are required to route requests to execution hosts and return results to the originating host. If you are using password validation, the user's password must be the same on all NQE servers; for example, if you are using NIS.
- Requests that are routed using destination selection are not guaranteed to run, because the local NQS systems may apply local limits to its queues. For information about using batch request limits in policies, see Section 8.7, page 221.

8.6.5 Example of Load Balancing Using the Default Policy

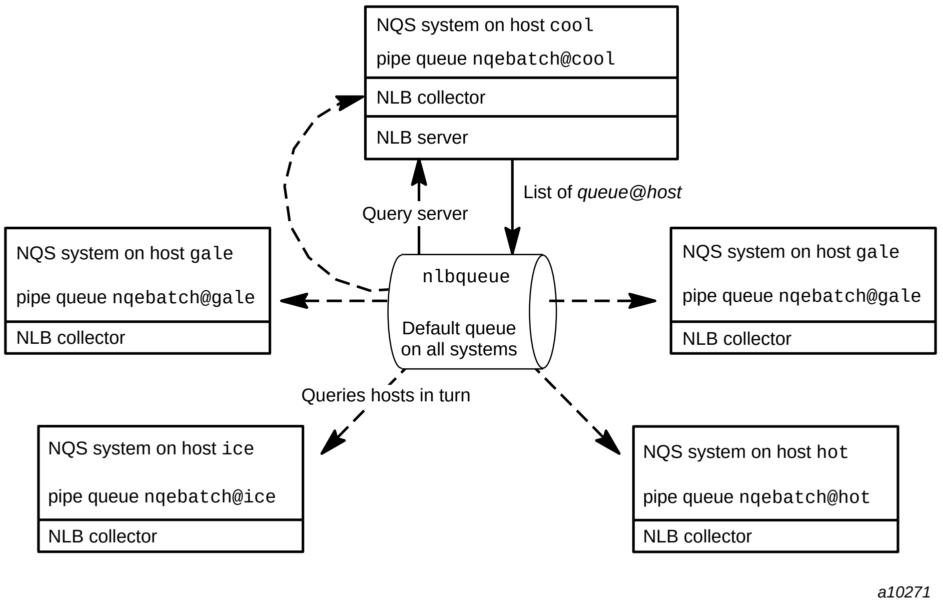


Figure 18. NQS Configuration with Load Balancing by Default

The following `qmgr` commands create the pipe queue `nlbqueue` and define it as the default queue for batch job requests:

```
% qmgr
Qmgr: create pipe_queue nlbqueue server=(pipeclient CRI_DS) priority=33
Qmgr: set default batch_request queue nlbqueue
Qmgr: quit
```

All requests that use the default queue are routed to a destination host by the destination selection algorithm. The NLB policy (for load balancing) defaults to `nqs`. The NLB server host defaults to `NLB_SERVER`.

8.6.6 Example of Load Balancing by Selection

In this scenario, a site has multiple NQS systems. The systems are similar, but there is production work that must run on specific systems. There is also work that could run on any available system. The default is that requests are submitted to a standard pipe queue. Users are encouraged to choose a destination selection queue when they submit work that can run anywhere in the complex.

This situation can be accommodated by defining two pipe queues, as shown in Figure 19. One is a standard pipe queue, called `standard`. The second, called `nlbqueue`, is configured to perform destination selection.

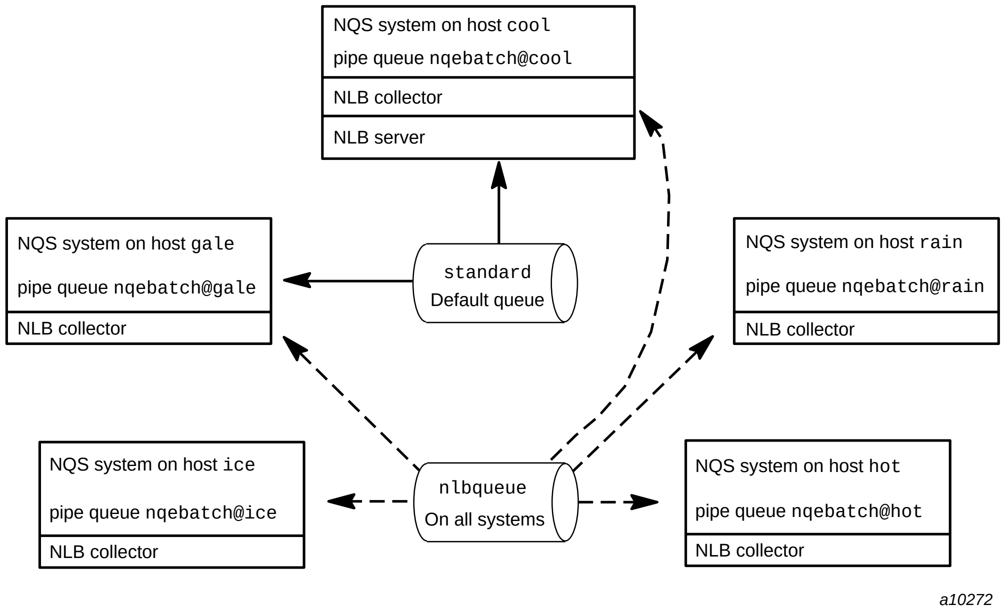


Figure 19. NQS Configuration with Load Balancing by Selection

The following `qmgr` commands create these pipe queues and set the default queue to `standard` so that requests are routed by standard NQS routing. The pipe queue `standard` explicitly defines a policy name, NLB host, and NLB port.

```
Qmgr: create pipe_queue standard server=(pipeclient) \
destination=(nqebatch@cool,nqebatch@gale) priority=30
Qmgr: create pipe_queue nlbqueue \
server=(pipeclient CRI_DS CPU cool:nlb) priority=30
Qmgr: set default batch_request queue standard
```

Requests that are submitted to the queue `nlbqueue` are routed to the most lightly loaded system.

8.7 Using Batch Request Limits in Policies

Users can set various per-process and per-request limits on batch requests when they submit the request. You can use these limits to define policies that route a request based on the size of the limits. NQS lets you specify attributes on a request by using the `General Options Attributes` selection of the `Configure` menu on the NQE GUI `Submit` window or by using the `cqsub -la` option. These attributes can be used within NLB policies by defining the same attributes in the NLB name map.

For example, using the `cqsub -la` option, the user specifies an NQS attribute as follows:

```
cqsub -la nastran
```

When it receives the request, NQS adds the characters `job_` to the beginning of the attribute name and checks to see if the attribute has been defined in the NLB name map. `JOB_` attributes are attributes of type `NLB_JOB`. In this example, the `JOB_NASTRAN` attribute is defined in the name map, and it will have the value 1 when applying the policy.

For example, you could define a policy that restricts large memory requests to large memory machines, or requests with small time limits could use less strict constraints when selecting a destination.

Using `nlbconfig`, you can define absolute request size limits for hosts and have policies enforce these limits. For example, a policy could constrain requests that require more than a specified amount of memory, disk space, or CPU time from being sent to specific machines.

The data sent by the destination selection module of the NQS `pipeclient` includes the attributes shown in Table 9. The attributes are present only when the associated command-line option was used in the request. All attributes, with the exception of tape drives, are floating-point numbers, and all values are normalized so that measurements are in bytes (or seconds for CPU limits). Tape drive attributes are integers.

The string attribute `JOB_REQNAME` that contains the request name is always present. You can use this attribute to make specific decisions for special request names.

Note: The following limits can also be set by selecting the `General Options Job Limits` submenus of the `Configure` menu on the NQE GUI `Submit` window.

Table 9. Request Limit Attributes

Option	Description	Attribute name
`-lc`	Specifies the per-process core file size limit.	`JOB_PPCORESIZE`
`-ld`	Specifies the per-process data segment size limit.	`JOB_PPDATASIZE`
`-lf`	Specifies the per-process permanent file space limits.	`JOB_PPPFILESIZE`
`-lF`	Specifies the per-request permanent file space limits.	`JOB_PRPFILESPACE`
`-lm`	Specifies the per-process memory size limits.	`JOB_PPMEMSIZE`
`-lM`	Specifies the per-request memory size limits.	`JOB_PRMEMSIZE`
`-lQ`	Specifies the per-request SDS limits.	`JOB_PRQFILESPACE`
`-ls`	Specifies the per-process stack size limit.	`JOB_PPSTACKSIZE`
`-lt`	Specifies the per-process CPU time limits.	`JOB_PPCPUTIME`
`-lT`	Specifies the per-request CPU time limits.	`JOB_PRCPUTIME`
`-lw`	Specifies the per-process working set size limit.	`JOB_PPWORKSET`
`-lU` *type*	Specifies the per-request tape drive limits for *type*, which can be `a`, `b`, `c`, `d`, `e`, `f`, `g`, or `h`.	`JOB_TAPES_A` to `JOB_TAPES_H`

8.7.1 Defining Request-attribute Policies

If you have three different application packages, NASTRAN, Oracle, and UniChem, which are used by requests but not available on all machines, you would set up a policy as described in this section. To let users specify on the command line which application packages are required, you can perform the following steps:

1. Define attribute names for hosts and requests to represent these packages. Download a copy of the existing name map by using the following command:

   ```
   nlbconfig -mdump -f name_map
   ```

   Edit the file `name_map` as shown in the following example. The attributes for the request must be defined for object type `NLB_JOB` with ID `1025` and must start with `job_`. When you edit the file, do not delete the existing names from the name map. If you do not include all of the existing names, they will be deleted when the server reads the new `name_map` file. The host attributes enable you to specify whether or not each host has that application available. Notice in step 2 that two of the three machines have UniChem, two have Oracle, and only one has NASTRAN. If a job requires an application, you want to submit it to a machine that has that application. This is how you define whether or not a machine has that application.

```
map NLB_JOB = 1025
{

(All names from the existing name map must be included)

     job_nastran     "Job requires nastran"    integer(8192);
     job_oracle      "Job requires oracle"     integer(8193);
     job_unichem     "Job requires unichem"    integer(8194);
}

map NLB = 1024
{

(All existing names from the name map must be included)

     host_nastran    "Host supports nastran"   integer(8192);
     host_oracle     "Host supports oracle"    integer(8193);
     host_unichem    "Host requires unichem"   integer(8194);
}
```

To add these names to the name map, use the following command:

```
nlbconfig -mput -f name_map
```

2. Define the `host_` attributes by creating an object data file and downloading this into the server.

```
object NLB (cool) {
    host_nastran = 1;
    host_unichem = 1;
}

object NLB (gust) {
    host_oracle = 1;
    host_unichem = 1;
}

object NLB (hot) {
    host_oracle = 1;
}
```

To load this file into the server, use the following command:

```
nlbconfig -oupdat -f obj_defs
```

3. Add constraints to the policy.

   You must define constraints for the policy that selects hosts based on the `job_` attributes. These constraints have no effect unless the user specifies an attribute using the `-la` option of the `cqsub` command or if using the `General Options Job Limits` submenus of the `Configure` menu on the NQE GUI `Submit` window.

   Use the implies operator (=>), as follows:

   ```
   constraint:    exists(job_nastran) => exists(host_nastran)
   constraint:    exists(job_oracle)  => exists(host_oracle)
   constraint:    exists(job_unichem) => exists(host_unichem)
   ```

   Using the implies operator (=>) in these constraints means that if a `job_` attribute is specified by the user, the equivalent `host` attribute also must exist. If the `job_` attribute is not present, the equivalent `host` attribute is not required either.

Such `job_` attributes can be used for any purpose. They are not restricted to the presence or absence of applications on hosts.

8.7.2 Testing the Policy

You can test a policy by using the `nlbpolicy -p` option. Its syntax is as follows:

```
nlbpolicy -p policyname
```

Optionally, you can use the `-a` option to specify any attribute described in Table 9, page 223, or you can create your own attributes and add them to the name map. Valid attributes are defined in the NLB for object type `NLB_JOB`.

They must begin with the characters `JOB_` and can be either a decimal `integer` or a `float`. The *value* can be either a decimal or a float, in normal or scientific notation. You cannot specify negative, hexadecimal, or octal numbers.

8.8 Storing Arbitrary Information in the NLB Database (Extensible Collector)

The extensible collector is a mechanism built into NQE that allows you to store arbitrary information in the NLB database. This information is periodically sent to the NLB along with the data that is normally stored and updated by NQE. Once the customized data is in the NLB database, it can be used in policies or displays just as any other data in the NLB database. NQE collects and stores the customized data, but you define, generate, and update it.

The customized data that is stored in the NLB database is added as new attributes to either the `NLB` object or the `NLB_JOB` object. Only attributes that are in these two objects can be used in policies and displays. Attributes that are in any other object that is used in a policy will result in the policy file not being read, and no policies will be available.

As a background for the rest of this description, you should be familiar with the following terms:

- Name map or name map file
- Object or NLB object
- Attribute
- Object data file
- `nlbconfig` command
- `nqeinfo` file; which is the site NQE configuration file, and it is generated at installation time (see Chapter 3, page 35)

The following steps are necessary to get customized objects into NQE:

1. Define the new attributes and put them in either the `NLB` object or the `NLB_JOB` object in the name map file so that they are available for use in a load-balancing policy.

   To add attributes to an object, the current `name_map` file should be retrieved from the NLB server by using the `nlbconfig` command. Then add the new attribute definitions to the current `name_map` file. (The current `name_map` file retrieved from the NLB server is an ASCII file, so any additions are done using an editor.) Finally, using the `nlbconfig` command, download the updated `name_map` file to the NLB server.

2. Generate an ASCII file of the following form:

   ```
   object NLB ("host name") {
                 NEW_ATTRIBUTE_1 = 12345;
                 NEW_ATTRIBUTE_2 = "This is a text string";
            }
   ```

   The attribute values may be added to an object without specifying all attributes of the object. This ASCII file is referred to as a *custom object file*.

3. Write a program or a shell script to populate a file with instances of the newly defined attributes. The format of this file is that of an object data file. This program or script will periodically update this object data file with new values for the attributes that have been defined.

4. Instruct the NQE collector program to read from the new object data file. This is done by using either the `-C` option of the `ccollect` program or by using the variable `NQE_CUSTOM_COLLECTOR_LIST` in either the `nqeinfo` file or as an environment variable. To specify multiple input files, use the `-C` option for each file.

   Note: If the environment variable method is used, it must be defined before the collector is started.

   The custom object files specified to the collector will be read once each interval and forwarded to the NLB server. After the program that generates the custom objects exists, the collector provides an additional option that starts the user program each time the collector is started. The collector automatically checks the `NQE_CUSTOM_COLLECTOR_LIST` variable for a list of programs to start. In addition to starting a program, the collector detects when the program terminates and restarts it.

The following example shows the syntax required to specify a list of custom collectors that the NLB collector will start. The NLB collector looks for this list in the `NQE_CUSTOM_COLLECTOR_LIST` environment variable or looks in the `nqeinfo` file for a variable of the same name. This example assumes the `nqeinfo` file is being used:

`NQE_CUSTOM_COLLECTOR_LIST="`*prog1*`,` *arg1*`,` *arg2*`:` *prog2*`:` *prog3*`,` *arg1*`"`

The program names are separated by a colon, and the arguments are separated by a comma. The program names are either full path names, or they are interpreted as being relative to the directory from which the NQE collector was started, which is `NQE_BIN` if the `nqeinit` script is used.

An additional feature, called *customized collector startup*, can help automate the generation of the extended object data files. This feature allows the administrator to specify one or more programs that will be automatically started by the NLB collector. The program can be either an executable file or a script file. Each program can have multiple command-line arguments. The NLB collector detects when any of the customized collectors has terminated and will restart it during its next cycle.

8.9 NLB Differences on UNICOS/mk

The NLB database information is supplied by the `ccollect` program. `ccollect` relies on the `sar` command for retrieving system performance data. The `sar` command on UNICOS/mk systems currently does not provide system performance data for memory or swapping usage statistics. As a result, the NQE GUI `Load` window for memory demand displays a fixed value of 96% when used for UNICOS/mk systems.

In addition, the following NLB attributes are not meaningful for UNICOS/mk systems: `NLB_A_SWAPPING`, `NLB_SWAPPING`, `NLB_SWAPSIZE`, `NLB_SWAPFREE`, `NLB_FREEMEM`, and `NLB_A_FREEMEM`.

NQE Database [9]

This chapter provides basic information about the NQE database and its components. It includes the following information:

- NQE database model
- NLB scheduling versus NQEDB scheduling
- An overview of the NQE database
- NQE database components
- NQE database concepts and terms
- Starting the NQE database
- Stopping the NQE database
- Obtaining status
- Security
- Configuring the NQE database components
- Compacting the NQE database

Chapter 10, page 261, describes the NQE scheduler in detail and includes a tutorial for writing a simple NQE scheduler.

9.1 NQE Database Model

While the NQE database model works well for specific environments, it has some limitations. The database used has a limit of 40 simultaneous connections. A connection is used for each execution server and each active client command. The performance of the database makes it well suited for environments where there are a small number of long running job scripts. It does not perform as well when there are large numbers of job scripts or very large single scripts.

Also, using the NQE database model requires that an installation write their own scheduler. The scheduler must be written using the TCL script language.

9.2 NLB Scheduling Versus NQEDB Scheduling

This section provides a brief overview of the advantages and disadvantages of both NLB scheduling and NQE database scheduling.

9.2.1 Failure Recovery

Failure recovery for the NLB is possible by specifying a list of nodes for the `NLB_SERVER` variable in the `nqeinfo`(5) file. The collectors will update the NLB database on each specified NLB server so that all nodes have up-to-date information. The NLB clients query the NLB servers one by one in the configured order until they find one that is running or until they exhaust the list. The one potential problem with NLB failure recovery involves using the `cevent`(1) command for Job Dependency. For more information, see Section 12.2, page 314.

There is no failure recovery for the NQEDB node. If the NQEDB node goes down or loses network access, the cluster becomes a set of individual NQS servers. If jobs finish while the NQEDB node is inaccessible, the NQE database will not be updated and the jobs will still be marked as running in the NQE database.

9.2.2 Number of Connections

The NQE database is limited to 36 connections. Four permanent connections are used by the NQEDB sever and one permanent connection is used by each lightweight server. Client connections are all transient and therefore limited only by performance considerations. However, if all 36 connections are used by servers, there is no connection left for clients connections. Therefore, the maximum number of nodes that may be in an NQE cluster is somewhere between 30 and 35, depending on the amount of expected client traffic.

9.2.3 Cluster Rerun

Cluster rerun of jobs is not available with NLB scheduling.

Cluster rerun of jobs is available with the NQE database. However, if there are network problems between the NQEDB sever and an NQS server, the NQS server will appear to be down. In this situation, the NQEDB sever will rerun the job on another server; this results in the job running twice.

9.2.4 Performance

Performance is better with the NLB. The NQE database and scheduler are written in TCL, an interpretative language, and is slower. Also, you can run the NLB without the NQEDB sever, but you can not run the NQEDB sever without the NLB, if you are interested in total cycles.

9.2.5 Relative Priority of Jobs

NQS queues give users a rough idea of the relative position of their jobs on that node.

In the NQE database implementation, there is no way for users to understand their job request priority relative to other job requests in queued state.

9.2.6 Site-defined Scheduling

The NLB allows sites to define policies for destination selection (routing) but not for priority scheduling of job requests. Priority scheduling is strictly first in, first out (FIFO).

The NQE Database as shipped uses FIFO scheduling but also allows sites to define their own schedulers using TCL scripts. This flexibility adds a level of complexity, as well as the obvious level of effort. However, the following types of scheduling would be possible with a customized NQE Scheduler:

- Destination selection where the job may not be sent immediately. An example of this would be sending a job to a server only if that server has an average idle CPU of greater than 50%, with the job remaining queued if no server is available.
- Network-wide limits. An example of this would be maintaining a global or network-wide limit on the number of jobs that a given user may run simultaneously.
- Normalizing resources across machines. For example, CPU time-limit requirements in a heterogeneous network are generally found empirically. It would be useful for the submitting user to supply the CPU requirements in an arbitrary unit that is then converted to seconds for the particular server on which the job is run.
- Network-wide rerun of failed jobs. For example, if a server on the network is detected as no longer running, certain jobs that were running on that server could be rerouted and run on another server.

- Scheduling on arbitrary data. Certain site-specific information may be important in scheduling. For example, a site may possess only a limited number of network-wide or server licenses for a particular application and the scheduler should not attempt to start up too many jobs using this application.

9.3 Overview of the NQE Database

The NQE database provides the following:

- A mechanism for the client user to submit requests to a central storage database.
- A scheduler that analyzes the requests in the NQE database and chooses when and where a request will run.
- The ability for a request to be submitted to the selected NQS server once the NQE scheduler has selected it.

Figure 20 shows an example of the layout of the NQE database. Jobs are submitted using client commands from client machines. The jobs are placed in the NQE database on one machine. A scheduler program determines which machine should run the job. In the example shown in Figure 20, the job is sent to Node D, where it is submitted and run in NQS.

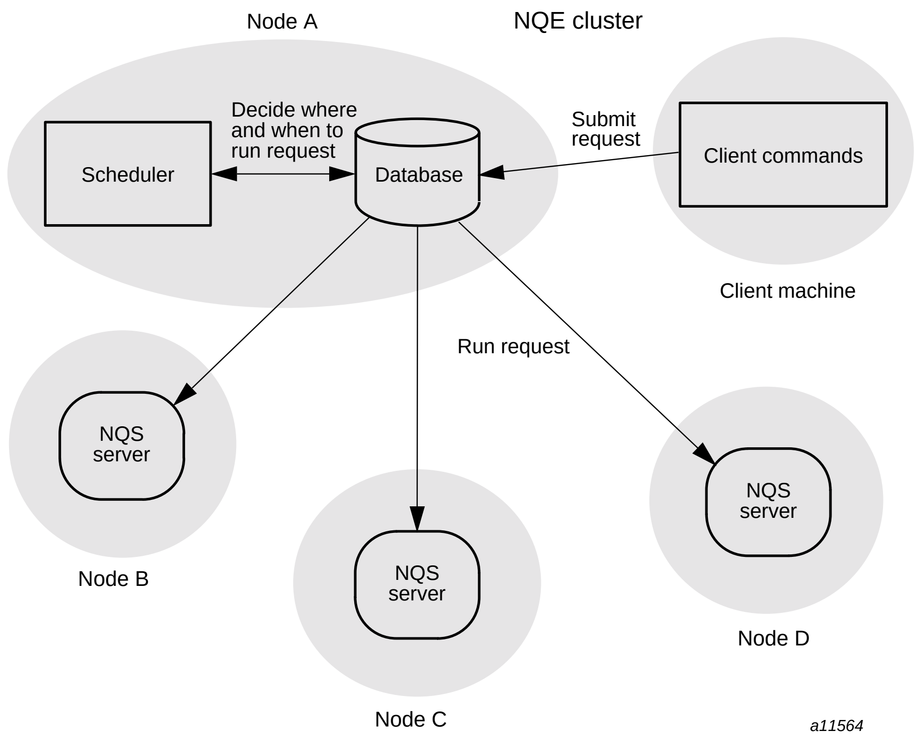


Figure 20. NQE Database Layout

9.4 NQE Database Components

Figure 21 shows in greater detail the processes and components of the NQE database. The components are described in the following sections.

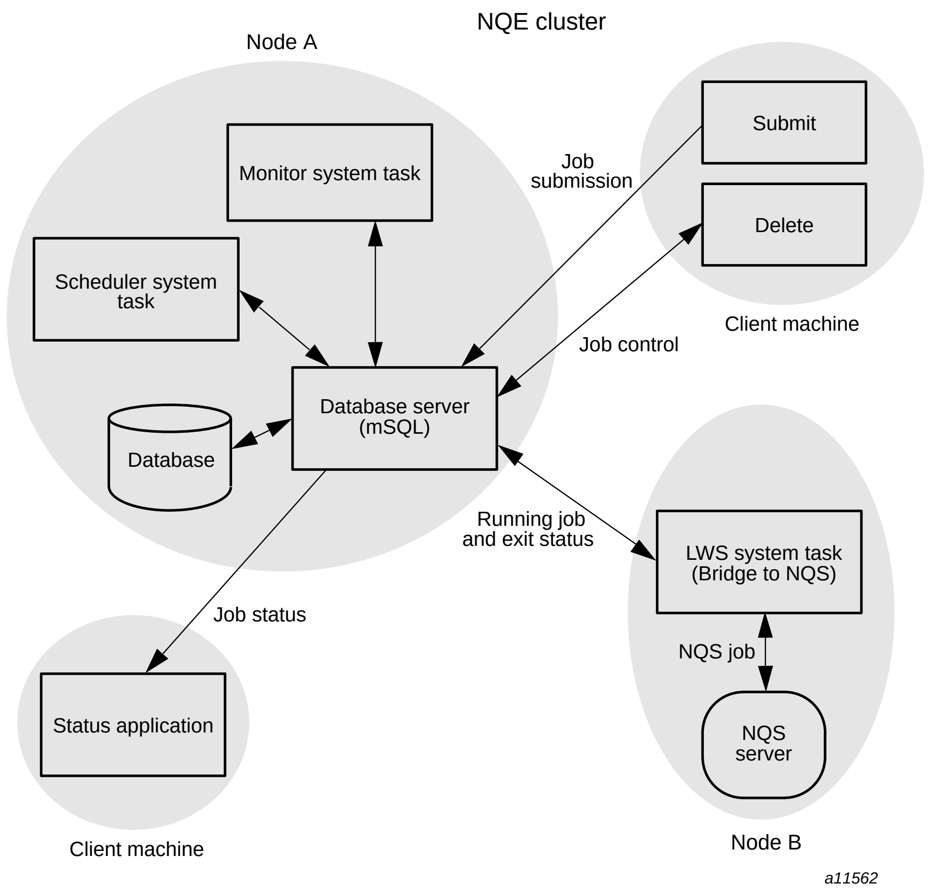


Figure 21. NQE Database Components

9.4.1 Database Server (MSQL)

The NQE database server (mSQL) serves connections from clients in the cluster or locally to access data in the database. The database used is mSQL, and it has the following features:

- Simple, single-threaded network database server
- Relational database structure with queries issued in a simple form of SQL

9.4.2 Monitor System Task

The monitor system task is responsible for monitoring the state of the database and which NQE database components are connected. Specifically, it determines which system tasks are connected and alive and, optionally, it purges old requests (called *tasks* when they are within the database) from the database.

A *system task* is an application that remains constantly connected to the database and performs a special task.

9.4.3 Scheduler System Task

The scheduler system task analyzes data in the database, making scheduling decisions. Specifically, it does the following:

- Keeps track of lightweight servers (LWSs) currently available for use in destination selection
- Receives new requests, verifies them for validity, and accepts or rejects them
- Executes a regular scheduling pass to find NQS servers to run pending requests

Administrator-defined functions may be written in Tcl to perform site-specific scheduling (see Chapter 10, page 261).

9.4.4 LWS System Task

The lightweight server (LWS) system task is a bridge to NQS, which performs the following tasks:

- Obtains request information from the database for requests assigned to it by the scheduler
- Checks authorization files locally
- Changes context and submits requests to NQS
- Obtains the exit status of completed requests from NQS and updates the NQE database with this information

9.4.5 Client Commands

The following client commands give access to the NQE database; for additional information, see the man page for the specific command or see the *NQE User's Guide*, publication SG–2148:

- `cqsub -d nqedb ...`

  This command submits the task to the NQE database rather than to NQS directly. The NQE database, scheduler, and LWS then submit an NQS request.

  > **Note:** If you use the `nqe` command to invoke the NQE GUI, the NQE GUI `Submit` window has a `Submit to NQE` selection on the `General Options` menu that provides the same function as this command.

- `cqstatl -d nqedb ...`

  This command gets the status of tasks in the NQE database, which allows you to see the status of all tasks in the NQE database, not just the local NQS system.

- `cqdel -d nqedb ...`

  This command sends a signal event to the task in the NQE database. This event is translated to signal or delete the corresponding NQS request; this action does not delete the task from the NQE database.

  > **Note:** If you use the `nqe` command to invoke the NQE GUI, the NQE GUI `Status` window has a `Signal Job` selection and a `Delete` selection on the `Actions` menu that provide the same functions as this command.

- If you use the `nqe` command to invoke the NQE GUI, the NQE GUI `Status` window provides status information for all requests (from the NQE database as well as for requests submitted directly to NQS).

- `nqedbmgr`

  This is the NQE database administration command. The command allows administrators to connect to the NQE database, and to examine and modify data in the database. In addition, it provides commands to start and stop aspects of the NQE database. These functions are described in the following sections.

- `compactdb`

This is an NQE database administration command. The command allows an administrator to compact the NQE database to improve its performance. For more information, see Section 9.10, page 253.

9.5 NQE Database Concepts and Terms

This section describes the following NQE database terms and concepts:

- Objects
- Task owners
- Task states
- Events

9.5.1 Objects

The NQE database is comprised of *objects*. Each object is, in turn, made up of *attributes*, which are simply name and value pairs. An instance of an object in the NQE database can contain any number of attributes.

Each instance of an object always contains at least the following attributes:

Attribute name	Description
`type`	One-character attribute used to differentiate between object types. Object types are described below.
`id`	A unique identifier for the object.
`timein`	The UNIX time (seconds since 00:00:00 GMT, January 1, 1970) at which the object was first inserted into the NQE database.
`timeup`	The UNIX time at which the object was last updated in the NQE database.
`sec.ins.dbuser`	The NQE database user name of the user who inserted the object.

The following object types are defined:

Object type (and abbreviation)	Description
Task or User task (`t`)	The object associated with an NQS request. Whenever a request is inserted into the NQE database, a new object of type `t` is created. Examples of attributes in a task object are: `script`, `exit.status`, and `env.HOME`. For a list of user task object attributes, see Table 21, page 292.
Event (`e`)	The object describing an event. Event objects are inserted into the NQE database in order to cause an action to occur. For a list of standard events, see Table 10, page 244. Posting an event to the NQE database simply means inserting an event object into the NQE database.
System task (`s`)	The object describing a system task. One object of this type is created for each system task connected to the NQE database. Examples of information stored include `s.host` (host in which the system task is running and `s.pid` (process ID) of the system task).
Global object (`g`)	The object containing global configuration data. There is only one instance of this object type; it contains any configuration information required by any client connecting to the NQE database. Configuration information stored here includes the rate at which system tasks query the NQE database for events, tasks, and so on. For a list of configuration attributes, see Table 14, page 255.

9.5.2 Task Owners

The *task owner* is the system task that is currently responsible for a given user task. Only the task owner may modify the attributes of a task. The task owner is defined in the attributes `t.sysclass` and `t.sysname`.

The name of a system task is expressed in the following form:

class.name

For example, the default scheduler's name is `scheduler.main`.

- *class* groups together system tasks that perform similar functions. There are three classes, which are defined as follows:
  - The monitor system task has a class name of `monitor`.
  - The scheduler system task has a class name of `scheduler`.
  - All LWS system tasks have a class name of `lws`.
- *name* describes the system task. The name used is dependent upon the system task class. A default name of `main` is given to both monitor and scheduler system tasks. The default name of an LWS is the name of the host on which the LWS runs.

  As administrator, you may set the name of the scheduler and LWS system tasks. This feature can be used to create, for example, a test scheduler.

Each system task regularly queries the NQE database to find any new user tasks it owns.

The scheduler chooses a host on which the user task will run and then changes ownership of the user task to the LWS system task for that host. The LWS system task then submits the task to the LWS daemon on that host. The LWS daemon issues the job request to the local NQS system.

9.5.3 Task States

User tasks have a state attribute (`t.state`). Different states signify different stages in the progress of the task through the system. There is no limit to the number of states a user task may pass through, and the name of a state is arbitrary. However, there are some well-defined states.

A task's state may be changed without changing the owner. Also, the owner of a task may be changed without changing its state.

Figure 22 shows some ownership changes and state changes possible for a user task.

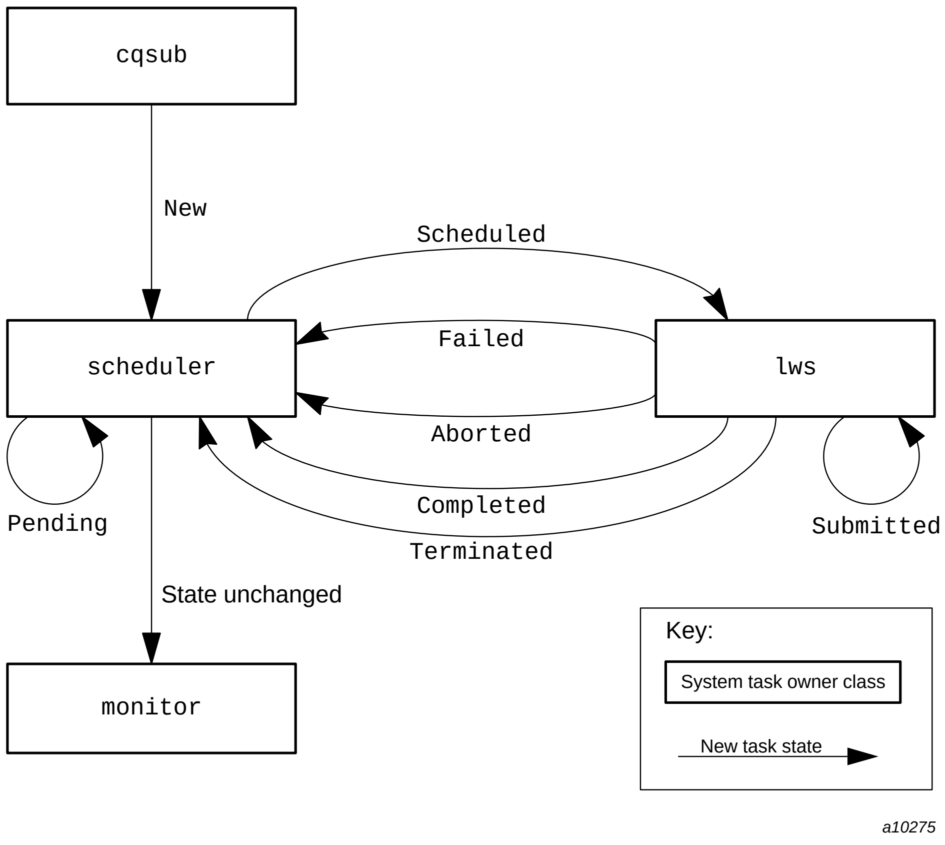


Figure 22. User Task Owners and States

A typical path for a user task is as follows:

1. The user uses the `cqsub` command to insert the task into the NQE database.

2. `cqsub` assigns ownership to the default scheduler system task (`scheduler.main`) with a state, `New`.

3. `scheduler.main` checks the task's validity and places it in state `Pending`. The `Pending` state is an example of an internal state; the scheduler may define many such states. (How long the task is in the `Pending` state depends upon the scheduler's algorithm.)

4. When the task is scheduled to run, `scheduler.main` assigns it to one of the `lws` system tasks with state `Scheduled`. There is, in general, one `lws`

system task for each node in the cluster. The LWS system task sends the request to the LWS daemon on its node.

5. The `lws` attempts to submit the task as an NQS request to the NQS running locally. If this action succeeds, the state is changed internally to `Submitted`. Until NQS completes the job, the user task remains in the `Submitted` state.

6. Upon job completion, the `lws` assigns task ownership back to `scheduler.main`. The state will be one of the following:

   - `Completed`, meaning that the request completed normally.
   - `Failed`, meaning that the request was not successfully submitted to NQS.
   - `Aborted`, meaning that something went wrong with the request while it was assigned to NQS.
   - `Terminated`, meaning that the request was deleted because a signal or delete command was sent to the request.

7. After job completion, `scheduler.main` examines the task's state and either places the task back in `Pending` state (signifying that the task is to be rerun) or passes the task to the monitor system task (`monitor.main`) without changing its state.

8. `monitor.main` receives all user tasks that will never run again. It is responsible for removing the task from the NQE database.

9.5.4 Events

Events form the mechanism for communication among users, administrators, user tasks, and system tasks.

Any client program connected to the NQE database may insert and post an event by inserting an object of type `e`. System tasks regularly check for any new user tasks they own. Once an event has been acted upon, it is marked as acknowledged and may be removed from the NQE database. (The monitor system task can be configured to automatically purge old events from the NQE database.)

Users may only post the signal event (by using either NQE GUI menu selection or the `cqdel` command) when it is targeted at user tasks they submitted. System tasks and administrators may send events to any object (including system tasks).

To send an event, use the `nqedbmgr event post` or `ask` command. For example, to post the event `xyz` with value `123` to object `s23`, use the following command; case is ignored for `xyz` in the example:

```
event post s23 xyz 123
```

Table 10 lists the standard events:

Table 10. Standard Events

Event name	Usual target	Description
`DUMP_VARS`	System task	Request that a system task dump to its log file all global Tcl variable names and values matching the event value. The value may be a glob pattern to match the desired variable(s), such as `*config*`, or values.
`SHUTDOWN`	System task	Request that the targeted system task shut down.
`SIGDEL`	User task	Send a signal to the NQS request represented by the user task. Delete the NQS request if the job request is not running. Assign the user task to the monitor system task. Note that this does not delete the user task from the NQE database. The value is optional and may be a signal number or name.
`TINIT`	System task	Reinitialize without shutting down. This event is typically posted to the scheduler to force it to restart.
`TPASS`	Scheduler	Request that the scheduler perform a scheduling pass.
`TRACE_LEVEL`	System task	Change the tracing level of the system task.

9.6 Starting the NQE Database

During installation, one NQE node may have been defined to run the mSQL database server by specifying `NQEDB`, `SCHEDULER`, and `MONITOR` in the NQE componets to be started or stopped by default (set by the `NQE_DEFAULT_COMPLIST` variable of the `nqeinfo`(5) file). For more information on the `NQE_DEFAULT_COMPLIST` variable and starting or stopping NQE components, see Chapter 4, page 45. If so, ensure that the `MSQL_SERVER` `nqeinfo` file variable is set to the name of the local machine. `MSQL_SERVER` should have the same value across all nodes, and clients in the cluster.

The `LWS` component should be specified in the `NQE_DEFAULT_COMPLIST` variable of the `nqeinfo` file on each NQE node where an NQS server is configured.

The `nqeinit` script is used to start all NQE components.

The `nqedbmgr` utility allows finer control over process startup. Table 11 describes commands available for this control; these commands are also described on the `nqedbmgr`(8) man page:

Table 11. `nqedbmgr` Process Startup Control Commands

Command	Description	Example
`db start`	Starts the database server (`msqld`). This command will attempt to start the mSQL daemon. If the daemon is already running, an appropriate message is displayed. You must be `root` and be on the NQE database server node to perform this command.	`db start`
`db create database`	Creates the NQE database. This command creates all the table definitions required for NQE's database. If the tables are already created, an appropriate message is displayed. You must be `root` and be on the NQE database server node to perform this command.	`db create database`
`create global` *name* `[`*tclarrayname*`]`	Creates the NQE global object. This command inserts an object into the database containing global configuration data. This object is read by all processes connected to the database. No action is performed if the object already exists. *tclarrayname* is an optional name of a Tcl array containing any extra configuration attributes for the object.	`create global config`

Command	Description	Example

`create systask` *name* [*tclarrayname*]	Creates a system task object.
	This command inserts an object into the database to be used by system tasks.
	One `systask` object is required for each system process that is to connect to the database. That is, one is needed for the `monitor` process, one is needed for the `scheduler` process, and one is needed for each `lws` process.
		`create systask monitor`
		`create systask lws`

`start` *systask*	Starts up the system task.
	This command starts the named system task process. On the `MSQL_SERVER`, `nqeinit` will start a `monitor`, a `scheduler`, and an `lws` process. On other NQE servers, `nqeinit` will create an `lws` system task object and start an `lws` process on the NQE server.
		`start lws`
		`start monitor`
		`start scheduler`
		`start lws.$host`

The `nqeinit` startup script uses `nqedbmgr` Tcl input scripts to perform all the startup operations. See those input scripts for examples of how the individual components are started.

Note: If NQS is not running on a system, the corresponding LWS system task will be marked `Exited`, and no tasks will be scheduled for that node.

9.7 Stopping the NQE Database

The `nqestop` script stops all system tasks and shuts down all NQE components.

The `nqedbmgr` utility allows finer control over process shutdown. Table 12 describes commands available for this control:

Table 12. `nqedbmgr` Process Shutdown Control Commands

Command	Description	Example
`shutdown all`	Shuts down all system tasks. This command sends events to all system tasks, requesting that they shut down.	`shutdown all`
`db shutdown`	Shuts down the database server. This command stops the mSQL database daemon. Any clients connected to the database are immediately disconnected. Any system tasks running when the mSQL daemon is shut down continue to run but enter a `Connection retry` loop.	`db shutdown`

The `nqestop` shutdown script uses `nqedbmgr` Tcl input scripts to perform all the shutdown operations. See those iput scripts for examples of how the individual components are stopped.

9.8 Obtaining Status

The `nqedbmgr` utility provides commands for determining the status of the NQE database and examining data in the database. Table 13 describes commands available to obtain status; these commands are also described on the `nqedbmgr`(8) man page:

Table 13. `nqedbmgr` Status Commands

Command	Description	Example
`status [all|db]`	Provides a status of the components of the NQE database. `status` by itself displays which system tasks are running, their PIDs, and the host on which they are running. `status db` obtains version information from the mSQL server. `status all` displays everything. **Note:** If you are running NQE on more than one host in your cluster and the system times for each of these hosts are not synchronized, the `Last heartbeat:` value for one or more of the LWS objects shown by the `status` command may show a negative number of seconds. This has no effect on the operation of the NQE cluster but can be resolved by synchronizing the system times on each host running NQE in your cluster.	`status`
`select` *options*	Reads data from the database and displays it. This command reads (or *selects*) data from the database and displays it. See the `nqedbmgr`(8) man page for details.	`select -f t`

9.9 Security

Authorization is performed at the following points:

- Connection: Upon connecting to the NQE database (all client commands and system tasks must connect to the database).
- Target: On the LWS machine just before a request is submitted on the target host selected by the NQE scheduler.
- Events: When an event is inserted into the NQE database (a delete or signal request sent by using the `cqdel` command or the NQE GUI inserts an event in order to delete a job).
- Status: When reading (*selecting*) information from the database.

Configuration requirements for each of these authorization points are described in the following sections.

9.9.1 Connection

Every client user (including a system task) wishing to connect to the mSQL daemon requires a valid database user entry in a file called `nqedbusers`; the `nqedbusers` file resides in the NQE spool directory defined by `NQE_SPOOL` in the `nqeinfo` file. When connected, any object (that is, request) inserted into the database is owned by the `dbuser`. The `dbuser` name may be a string of any alphanumeric characters and does not necessarily have to be a UNIX user name (`dbuser`s do not need `/etc/passwd` entries on the machine running the mSQL daemon).

The mSQL daemon reads and uses the contents of the `nqedbusers` file to determine whether a given client may connect to the database. If, as a result of this check, the client user's connection request is refused, a `Connection disallowed` message is passed back to the client and the connection is broken.

The `nqedbusers` file is made up of an optional `%host` segment followed by multiple `%dbuser` segments. Blank lines and lines beginning with # are ignored.

The `host` segment specifies an optional list of client host names explicitly allowed or disallowed access to the database. If the segment is not supplied, or the list of host names is empty, any client host may connect to the database, and the checks move on to the `dbuser` segments. The syntax in the `nqedbusers` file is as follows:

```
%host
      [clienthostspec
      clienthostspec
      ...]
```

A *clienthostspec* is a host name optionally prefixed by + (to allow access explicitly) or — (to disallow access).

The database user segment defines allowed or disallowed access for a specified database user. The syntax in the `nqedbusers` file is as follows:

```
%dbuser dbusernamespec privtype
        [clienthostspec clientuserspec
        clienthostspec clientuserspec
        ...]
```

The *dbusernamespec* is the `dbuser` name and may be optionally prefixed by — to explicitly disallow the dbuser. The following formats allow any user to connect to the NQE database:

```
%dbuser
%dbuser +
%dbuser + user
```

If the *dbusernamespec* is not supplied, or the list of `dbuser` names is empty, any user may connect to the database. A *dbusernamespec* of + alone has a special meaning and corresponds to a `dbuser` equal to the client user name. This entry allows access for many users using their existing UNIX user names without the necessity of explicitly specifying each name in the file. The `%dbuser + user` format allows any user to connect to the NQE database; `user` is the *privtype*.

System tasks (monitor, scheduler and LWS) connect using a `dbuser` of `root`.

privtype specifies the type of connection privilege. Two types are defined, as follows:

- `user`, which allows a request to be submitted and deleted and allows a user to obtain the status of the request by using the `cqsub`, `cqstatl`, `cqdel`, and NQE GUI interfaces.
- `system`, which allows full access to the NQE database.

clientuserspec is a UNIX user name optionally prefixed by + (to allow access explicitly) or - (to disallow access). An entry of simply + allows any client to connect as the given *dbusername*.

An example of an `nqedbusers` file follows:

```
# nqedbusers file
#
# Allow root access from any host
%host
       # Disallow hosts
       -badhost
       # Allow other hosts
       +
%dbuser root system
       localhost root
       server1 root
       server2 root
%dbuser fred user
       client1 shirley
```

```
#
# Allow any user access from any host.
# ( db username = client username )
%dbuser + user
       -badclient bob
       - amy
       + +
```

In this file, access is completely denied for host `badhost`, but all other hosts are allowed access. A `dbuser` of `root` has been defined with full access privileges (`system`) for hosts `localhost`, `server1`, and `server2`, which allows the LWS to run from those three hosts. A `dbuser` of `fred` has been defined and may be used by `shirley` from client host `client1`. All other users are allowed to use their UNIX names to connect as the `dbuser` name, with the exceptions of `bob` from `badclient` (`bob` can connect from anywhere except from `badclient`) and `amy`, who cannot connect from any host.

9.9.2 Target

The administrator may define how the LWS validates jobs from the NQE database before submitting them to NQS locally. Valid values are as follows:

- `FILE`. The LWS checks for a file called ~*user*/`.nqshosts` or ~*user*/`.rhosts`. Within the file is a list identifying which client users from which client hosts are allowed or disallowed access. The task attributes `sec.ins.clienthost` and `sec.ins.clientuser` are used to determine which client hosts allow access by which users. These client host and client user names must be included in the `.rhosts` file or `.nqshosts` file. For information about user task attributes, see Section 10.3.3, page 292.
- `PASSWD`. A valid UNIX password is required.

The initial value when NQE is first started is `FILE`. After that, you must update the global configuration object in the NQE database and post a `TINIT` event to the `lws` to tell it to re-read the configuration. For example:

```
% select s
Connected to coffee as DB user root (conn=3)

     ID      CLASS NAME              STATE
     s2    monitor main            Running
     s3  scheduler main            Running
     s4        lws coffee          Running
     s5        lws latte           Running
     s6        lws cappuchino      Running

% select -f s4 {run_mechanism}
Object s4:
         run_mechanism = File

% update s4 {run_mechanism Passwd}
Object s4 updated

% select -f s4 {run_mechanism}
Object s4:
         run_mechanism = Passwd

% event post s4 TINIT
Posted event e16516
```

9.9.3 Events

Event objects are used to communicate between tasks, both system tasks and user tasks. For example, if you change the global configuration, you need to send a `TINIT` event to the appropriate system tasks in order to tell these tasks to re-read the configuration and get the new values.

Table 20, page 292, lists the predefined events you may want to send.

A `dbuser` connected with the privilege type of `user` may only send events to job tasks created by the same `dbuser`. Those connected with the privilege type of `system` may send events to any task, including system tasks.

9.9.4 Status

A `dbuser` connected with `system` privilege may read any object in the database by using the `nqedbmgr` interface. That is, any task may be examined.

A `dbuser` connected with `user` privilege is restricted to using the `cqsub`, `cqstatl`, `cqdel`, and NQE GUI interfaces.

The administrator may allow the user to see summary information on all tasks in the NQE database, not just the user's requests and tasks. This is achieved with the global configuration attribute `status.access`.

Note: You do not need to post an event when you issue the following `nqedbmgr` commands.

The following `nqedbmgr` command allows users to see summary information on all requests and tasks (the default is `all`):

```
update config {status.access all}
```

The following `nqedbmgr` command allows a user to see only the requests and tasks the user created:

```
update config {status.access own}
```

9.10 Compacting the NQE Database

The NQE database is an mSQL database that grows, but never shrinks. This can degrade the performance of the database. You can use the `compactdb`(8) script to compact the NQE database (`nqedb`) to improve its performance.

Warning: You must shut down the NQE database server before you invoke the `compactdb`(8) script to avoid corruption of the database or lost data. You can use the `nqestop`(8) script to stop all system tasks and shut down the NQE database.

9.11 Configuring the Database Components

The `MAX_SCRIPT_SIZE nqeinfo` file variable lets the administrator limit the size of scripts that are submitted through `nqedb`. If a script contains more lines than the number of lines defined by the `MAX_SCRIPT_SIZE` variable, it is not allowed. If `MAX_SCRIPT_SIZE` is 0 or is not set, scripts of unlimited size are allowed.

You can set and/or update various global configuration attributes for the NQE database by using the following `nqedbmgr` command syntax:

```
update config_obj {attribname attribvalue...}
```

For example:

```
update g1 {status.access all}
```

A system task must be re-initialized or restarted in order to read any updated global configuration attributes.

> **Note:** If you are not certain which system tasks are affected by your change, it is recommended that you re-initialize all system tasks to ensure all tasks that read a specific variable are notified of the variable's updated attributes.

To re-initialize a system task (for example, the `scheduler`), enter the `nqedbmgr` command that inserts a `TINIT event` object into the database; this object is read by the scheduler and acknowledged:

```
ask scheduler tinit
```

The following sections describe the predefined global configuration attributes for the NQE database, NQE scheduler, and LWS attributes that you can use.

9.11.1 Global Configuration Attributes for the NQE Database

Table 14 lists the predefined global configuration attributes for the NQE database (this table is also provided on the `nqedbmgr`(8) man page):

Table 14. `nqedbmgr` Predefined Global NQE Database Configuration Attributes

Attribute name	Description	Default if not specified
`hb.systask_timeout_period`	Time before monitor system task marks a given system task as down. The value may be a mathematical expression. Units are seconds.	`90`
`hb.update_period`	Time between each update of the heartbeat. Each system task will update the database within this period. It is used by the monitor to determine whether a system task is still alive. The value may be a mathematical expression. Units are seconds.	`30`
`purge.systask_age`	Age (elapsed time since last update) of acknowledged events concerning system tasks that the monitor is to delete. The monitor will delete all tasks that are the specified age or older. The value may be a mathematical expression. Units are seconds.	`24*60*60`
`purge.task_age`	Age (elapsed time since last update) of completed user tasks and user events that the monitor is to delete. The value may be a mathematical expression. Units are seconds.	`24*60*60`
`status.access`	Type of access allowed by users when attempting to obtain summary information of jobs in the database. Value can be: • `all`, which allows user to select summary information from all tasks • `own`, which allows user to select only tasks submitted by the user A user may only see full information for tasks submitted by that user.	`all`

Attribute name	Description	Default if not specified
`sys.ev_rate`	Rate at which system tasks check for new events targeted at them or at tasks they own. This attribute may be a mathematical expression. Units are seconds.	`10`
`sys.purge_rate`	Rate at which monitor checks the age of user tasks, user events, and system events. This attribute may be a mathematical expression. Units are seconds.	`5*60`
`sys.st_rate`	Rate at which system tasks check for new tasks, changes in task state, and so on. This attribute may be a mathematical expression. Units are seconds.	`10`
`sys.upd_rate`	Rate at which an LWS system task checks for messages from NQS. This attribute may be a mathematical expression. Units are seconds.	`10`

9.11.2 Scheduler Attributes

The default scheduler file is situated in the /*nqebase*/`bin` directory and is called `local_sched.tcl`. The default scheduler has the following properties:

- The number of tasks that may run on an LWS can be limited.
- The number of tasks per user that may run on an LWS can be limited.
- LWS selection can be performed by a simple round-robin or by using an NLB policy (passing memory and CPU requirements).
- Tasks are ordered by time in.
- The user may narrow the list of target NQS servers by passing a colon-separated list of hosts as a named request attribute, `lws`. For example, `cqsub -la lws=hosta:hostb` narrows the target server list to `hosta` and `hostb`.
- A simple cluster rerun facility is implemented.

The default scheduler can interpret the following predefined attributes, which are overridden if they are set in the system scheduler object. The following example sets the scheduler `passrate` attribute, which overrides the default value of 60:

```
nqedbmgr update scheduler {passrate 70}
```

Table 15 lists these attributes (this table is also provided on the `nqedbmgr`(8) man page).

Table 15. nqedbmgr Predefined Scheduler Attributes

Attribute name	Description	Default if not specified
max_fail_count	Cluster rerun attribute. Maximum job failures allowed (i.e., number of times LWS returns a job to the scheduler without completing it) before the job is no longer requeued.	5
nlbpolicy	NLB policy to use to determine a destination-selection list for each task. none means use round-robin.	none
passrate	Rate, in seconds, at which scheduling passes are made.	60
s.clrerun	Allow cluster rerun.	No
s.schedfile	Name of scheduler file. The file name may be the absolute path name or relative to the NQE bin directory as defined by NQE_BIN in the nqeinfo file. Note: This attribute is always recognized; it defines which scheduler is running.	local_sched.tcl
total_lws_max	Limit on number of tasks that may be running on any one LWS.	5
total_lws_user_max	Limit on number of tasks that any one user may have running on any one LWS.	2

9.11.3 LWS Attributes

Table 16 lists the predefined LWS attributes (this table is also provided on the nqedbmgr(8) man page):

Table 16. `nqedbmgr` Predefined LWS Attributes

Attribute Name	Description	Default if not specified
`default_queue`	Name of the default NQS queue to which jobs are submitted.	`nqebatch`
`qmgrnow`	If set and true, requests that LWS follow any submission to NQS by the `qmgr> scheduler request` *xyz* `now`, where *xyz* is the NQS request ID. This will force NQS to run a job, even if local NQS scheduling parameters deny it.	`Yes`
`run_mechanism`	Defines the validation mechanism used to check the target user against the client user for local submission to NQS. The mechanisms available are: • `File`, which indicates that NQS file validation is to be used • `Passwd`, which indicates that NQS password validation is to be used.	`File`

Writing an NQE Scheduler [10]

The scheduling algorithm used by the NQE scheduler is designed to be modified. Also, it is possible to completely rewrite the scheduler. This chapter describes the NQE database and scheduler in greater detail than Chapter 9, page 231, and it guides you in writing a site-specific scheduler.

You should note the following points:

- All data concerning requests submitted to the NQE database is contained within that database. Therefore, any data that the NQE scheduler wishes to remain in existence after it shuts down must be stored in the NQE database. Examples of this data include request information, request script files, job log/history information, and scheduling configuration and request exit statuses.
- If the scheduler is killed and then restarted, or the connection to the NQE database is lost and then regained, the scheduler should be capable of recovering its state based entirely on the contents of the NQE database.

This chapter contains information about the following:

- Scheduler system task
- Writing a simple scheduler
- Attribute names and `nqeinfo` file variable names
- Tcl built-in functions

10.1 The Scheduler System Task

All system tasks, including the scheduler, consist of Tcl scripts that do the following:

- Run with `root` privilege.
- Connect to the NQE database and remain continuously connected.
- Regularly make queries of the NQE database, analyze data, and modify or act on that data.

Figure 23 shows the structure of the scheduler system task (referred to as the *NQE scheduler*). As shown by the diagram, it consists of three main parts:

- `nqe_tclsh`. This is the NQE version of the Tcl shell. It contains built-in Tcl functions to perform, among other things, the task of connecting to and accessing the NQE database.
- `nqedb_scheduler.tcl`. This is the name of the Tcl script that implements the mandatory parts of the NQE scheduler. It may be invoked as the first argument to `nqe_tclsh` or directly as an executable script. The `nqedbmgr` command `start scheduler`, simply executes this Tcl script as a background UNIX process.
- `local_sched.tcl`. This is the name of the local scheduler (that is, the Tcl script containing the local or site-specific scheduling algorithm). It is sourced by `nqedb_scheduler` and contains callback functions to implement the algorithm.

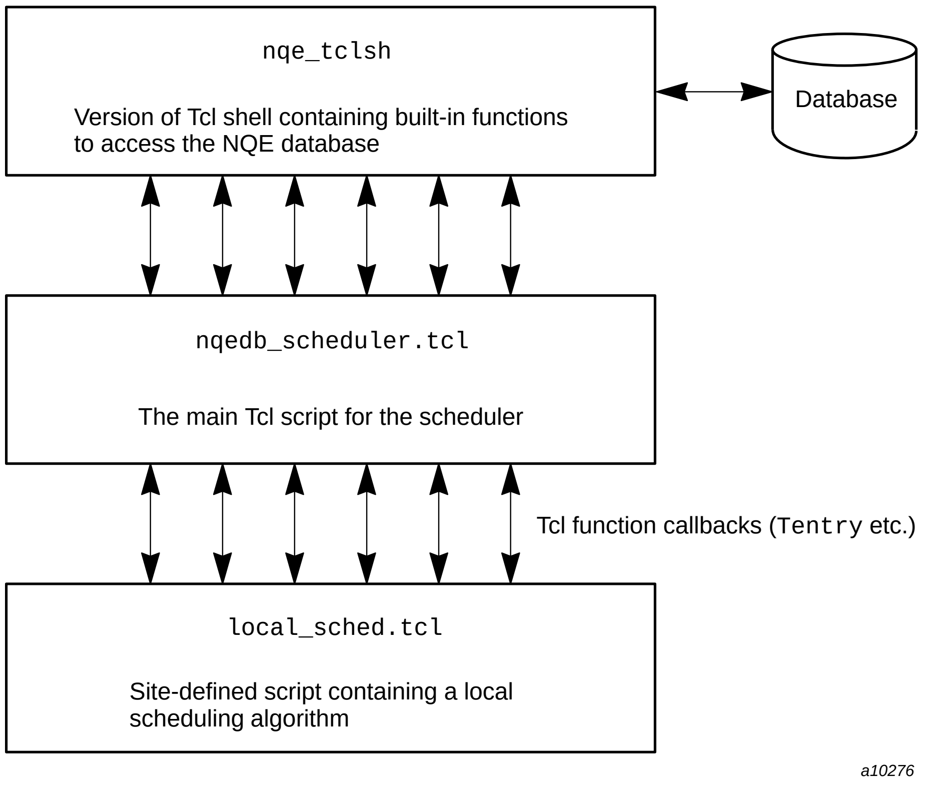

nqe_tclsh
Version of Tcl shell containing built-in functions to access the NQE database
Database
nqedb_scheduler.tcl
The main Tcl script for the scheduler
Tcl function callbacks (Tentry etc.)
local_sched.tcl
Site-defined script containing a local scheduling algorithm
a10276

Figure 23. Scheduler Program Structure

By default, the local scheduler script resides in the location defined by `NQE_BIN` in the `nqeinfo` file. Its name and location may be changed by either of the following:

- Defining another `nqeinfo` file variable, `NQEDB_SCHEDFILE`, set to the name of the script file (this can be an absolute path name or relative to `NQE_BIN`).
- Setting a new attribute, `s.schedfile`, for a particular scheduler system task object. See Section 10.2 for an example of how to do this.

10.2 Writing a Simple Scheduler

The scheduler is written in the Tcl language. The most effective way of describing the operation of the NQE scheduler and the way in which it can be modified is by performing the steps necessary to create a new, site-specific NQE scheduler.

This section provides a tutorial in writing an NQE scheduler.

10.2.1 Requirements

The following is required before starting the tutorial:

- You must have knowledge of the Tcl language in order to modify the NQE scheduler. Tcl is a very simple but powerful interpreted language. Many Tcl language books include excellent tutorials on the language.
- You must have `root` access on the NQE database server node running the mSQL daemon, the NQE scheduler system task, and the NQE monitor system task. Only one NQE node is configured to run the mSQL daemon. This is typically determined during installation of NQE. To ensure that a given machine is configured to run the NQE database server, check that the `nqeinfo` file variable `NQE_DEFAULT_COMPLIST` setting contains the `NQEDB`, `SCHEDULER`, `MONITOR`, and `LWS` components, and that the `MSQL_SERVER` variable is set to the name of the local machine.
- Your `PATH` must include the directory defined by the `nqeinfo` file variable, `NQE_BIN`.
- The NQE database , NQE scheduler, and NQE monitor must be running.

The `nqeinit` process should start these system tasks by default; the following message is displayed:

```
NQE database startup complete
```

To check that system tasks are running, execute the `nqedbmgr` command `status`. The output should show that mSQL is running and that the `monitor`, `scheduler`, and at least the local `lws` system tasks are running.

Following is an example of the `status` output for a two node cluster with the NQE database server running on `latte` and a LWS running on `pendulum`:

```
nqedbmgr status
----------------------------------------------------------------
Connection:
-----------
    Connecting... Successful
    Client user name: root
    DB user name:     root
    DB server name:   latte
    DB server port:   603
----------------------------------------------------------------
Global config object:
---------------------
    Heartbeat timeout:       90
    Heartbeat rate:          30
    System event purge rate: 24*60*60
    User task purge rate:    24*60*60
    Summary status:          all
----------------------------------------------------------------
System tasks:
-------------
monitor.main (id=s2):
    Status:         Running
    Pid and host:   2669 on latte
    Last heartbeat: 26 seconds ago (Update rate is every 30 secs)

scheduler.main (id=s3):
    Status:         Running
    Pid and host:   2678 on latte
    Last heartbeat: 22 seconds ago (Update rate is every 30 secs)

lws.latte (id=s4):
    Status:         Running
    Pid and host:   2686 on latte
    Last heartbeat: 20 seconds ago (Update rate is every 30 secs)
    Default queue: nqebatch

lws.pendulum (id=s5):
    Status:         Running
    Pid and host:   24149 on pendulum
    Last heartbeat: 22 seconds ago (Update rate is every 30 secs)
    Default queue: nqebatch
```

10.2.2 Goal

The goals of the tutorial are as follows:

- Create a new NQE scheduler, called `simple`, that will work alongside the default NQE scheduler (by default called `main`).
- Write a simple scheduler algorithm that schedules tasks to lightweight servers (LWSs) in a round-robin manner. The simple scheduler should also limit the total number of jobs a given LWS can run at any time; in other words, you will implement a global LWS run limit.
- Test the scheduler.
- Add the enhancement of allowing the run limit to be configurable.

The steps in achieving this are listed below and are described in the following sections:

1. Create a new scheduler system task object for `simple`
2. Create a `simple` local scheduler file
3. Write a `Tinit` callback function to initialize variables
4. Write a `Tentry` callback function to handle user tasks newly assigned or newly reassigned to your scheduler
5. Write a `Tpass` callback function to schedule user tasks
6. Write a `Tnewowner` callback function to update internal counts and lists
7. Write a `Tentry` callback function for user tasks returned from an LWS
8. Run the new scheduler and submit jobs that it will schedule
9. Add configuration enhancements to the `Tinit` callback function

10.2.3 Step 1: Create a New Scheduler System Task Object

It is desirable to develop and test a scheduler without disturbing any production NQE scheduler already running on the system. This can be achieved by creating a new scheduler system task object with the following command:

```
nqedbmgr create systask scheduler.simple
```

This command inserts a new system task object into the NQE database.

Note: This action does not start the scheduler; it only creates the system task object.

The output should look similar to the following:

```
Connected to latte as DB user root (conn=4)
Object scheduler.simple inserted (s6)
```

Here the local host name is `latte`. A new system task object has been created with the ID `s6`. This ID varies depending upon how many system task objects already exist in the NQE database. The system task class is `scheduler`. Its name is `simple` (the default scheduler name is `main`). When test jobs are submitted to the NQE database, you will be able to specify which scheduler should receive, own, and schedule the job.

The system task object can be deleted by using the following commands:

```
nqedbmgr shutdown scheduler.simple
nqedbmgr delete scheduler.simple
```

Caution: The `delete` command should never be issued while the `simple` scheduler is still running.

For additional information about the `nqedbmgr shutdown` command, see Section 9.7, page 246.

10.2.4 Step 2: Create the Simple Scheduler File

The following local scheduler files are provided with NQE:

File	Description
`template_sched.tcl`	(Located in the /*nqebase*/`bin` directory). This is a template file for local schedulers. If run without modifications, this scheduler will accept all tasks but never schedule them.
`local_sched.tcl`	(Located in the /*nqebase*/`bin` directory). This is the default scheduler supplied. (For more information about the default scheduler, see Section 9.11.2, page 256.)
`simple_sched.tcl`	(Located in the /*nqebase*/`examples` directory). This is a final version of the simple scheduler

described in the remainder of this tutorial and may be used for comparison.

Copy the template file to a file named `simple_sched.tcl` in the `$NQE_BIN` directory. Ensure that the file has write permission because it will be modified in later steps.

The newly created scheduler system task object (step 1) must be configured as follows to read the Tcl file you just created:

```
nqedbmgr
% update scheduler.simple {s.schedfile simple_sched.tcl}
% exit
```

This command updates the `scheduler.simple` system task object, setting the attribute `s.schedfile` to `simple_sched.tcl`.

You can list the attributes in `scheduler.simple` by using the following command:

```
nqedbmgr select -f scheduler.simple
```

The scheduler will read this object and source the file specified by this attribute.

It is the responsibility of the local scheduler file, `simple_sched.tcl`, to do the following:

- Define at least four special Tcl functions to perform the scheduling. These special Tcl functions are callbacks used by the scheduler program.
- Invoke the Tcl function `sched_set_callback` for each of the four callbacks to register the functions' names. The syntax is as follows:

```
sched_set_callback callback functionname
```

callback may be one of the following:

Function	Description
`Tinit`	Initialize scheduler
`Tentry`	Handle new tasks
`Tpass`	Make a scheduling pass

`Tnewowner`	Assign a new owner

An example of its use follows:

```
# Register function fnc_Tinit for callback Tinit
sched_set_callback Tinit fnc_Tinit
```

The remaining steps describe how to create the functions listed above and explain their purpose.

10.2.5 Step 3: Write a `Tinit` Callback Function

Write a `Tinit` callback function.

The `Tinit` callback is called when the following occurs:

- The scheduler starts up.
- The scheduler receives the `TINIT` event. You can send an event to the simple scheduler by using the following `nqedbmgr` command:

  ```
  nqedbmgr ask scheduler.simple tinit
  ```

- The scheduler reconnects to the NQE database after a previous, unexpected disconnection.

The `Tinit` callback should be used to perform any local initialization required by the scheduler. It returns with a value of 1 if initialization was successful; otherwise, it returns with a value of 0, and the scheduler shuts down.

The following two global Tcl variables are defined before the `Tinit` callback is invoked:

- `Obj_config`. This is a global Tcl array containing the global configuration object. The elements of the array correspond to the attributes of the NQE database object of type `g` and name `config.main`.

  The `config.main` object is read from the NQE database just before the `Tinit` callback function is called. The object can contain any configuration information required by user commands and system tasks.

  For a list of predefined attributes, see Section 10.3.5, page 299. You may add others.

- `Obj_me`. This global Tcl array contains the attributes corresponding to the scheduler system task object itself.

This array is the main mechanism for passing configuration data to a particular system task.

For a list of predefined attributes, see Section 10.3.4, page 297.

The first version of the simple scheduler will not make use of the two global Tcl arrays. It will, however, use its own global Tcl array (called `Sched`) to store all running information concerning the tasks. This array should be unset and initialized at this point. The code for the `Tinit` callback follows. The variables initialized will be described later. The following code should be added to `simple_sched.tcl`:

```
#
# Register the function
#
sched_set_callback Tinit fnc_Tinit

#
# Define function fnc_tinit
#      Arguments: None
#
proc fnc_Tinit {} {
      global Obj_config Obj_me

      global Sched

      sched_log "TINIT: Entered"

      if { [info exists Sched] } {
             #
             # Clear internal information array if it exists
             #
             unset Sched
      }

      # Global run limit per LWS
      set Sched(total_lws_max) 5
      sched_trace "TPASS: Global per-LWS run limit:\
            $Sched(total_lws_max) "

      # Scheduler pass rate (when idle)
      set Sched(passrate) 60

      # The round-robin index
      set Sched(rrindex) 0

      # Initialize lists
```

```
        set Sched(pending_list) ""
        set Sched(schedule_list) ""

     return 1
}
```

10.2.6 Step 4: Write a `Tentry` Callback Function for New Tasks

Write a `Tentry` callback function to handle new tasks.

The `Tentry` callback is called when the following occurs:

- A newly submitted user task is detected. New tasks always enter the system with state `New`.
- Upon startup or `Tinit`, `Tentry` is called once for each user task already owned by the scheduler.
- A user task is completed or fails on an LWS.

The `Tentry` callback should be used to update local lists and counts, depending upon the state of the user task. The `Tentry` callback may also change the state and/or owner of the user task.

A local scheduler may define as many states as it wants for the user tasks it owns. Table 17 lists the states that must be recognized by the `Tentry` callback:

Table 17. Mandatory Task States Recognized by the `Tentry` Callback

Task state	Description
`New`	Indicates the initial state immediately after submission by user (by using NQE GUI or the `cqsub` command).
`Scheduled`	Indicates the scheduler has assigned a user task to a specific LWS. The LWS system task interprets this state as meaning "This task is to be submitted to the local NQS." After successful submission, the LWS places the task in `Submitted` state. However, this latter state is never communicated to the scheduler and does not need to be trapped.
`Completed`	Indicates that the job request has completed successfully.
`Failed`	Indicates that the job request cannot be submitted to NQS.

Task state	Description
`Aborted`	Indicates that NQS cannot run the job request associated with this user task. This may occur, for example, if the job request is successfully submitted to an NQS pipe queue but then cannot be placed in a batch queue due to queue limits.
`Terminated`	Indicates that a user task has been terminated by execution of a delete request (either through the NQE GUI or the `cqdel` command).

In this step, you will write code to deal with the `New` state and to define a new local state, `Pending`, which indicates that the user task has been accepted by this scheduler and is queued.

Replace the template version of `fnc_Tentry` with the following `fnc_Tentry` function:

```
#
# Register the function
#
sched_set_callback Tentry fnc_Tentry

#
# Define function fnc_Tentry
#      Argument: objid. Object ID of user task
#
proc fnc_Tentry { objid } {
       global Obj_config Obj_me Us Lwsid
       global Sched

       # Associate global variable Obj_$objid
       # with variable obj in this fnc.
       upvar #0 Obj_$objid obj

       #
       # Set some local variables for ease of use
       #
       set state $obj(t.state)
       set substate $obj(t.substate)

       # Log a message to scheduler log file
        sched_log "TENTRY: objid = $objid, state =
 $state/$substate"

       # Look at the state of this task
        switch -regexp $state {

        New {
             # New: Task has just been submitted to NQE by cqsub
             set upd(t.state) Pending
             sched_update $objid upd
        }

        Pending {
             # Pending: Scheduler-specific state
             # Add task to pending_list
             nqe_listrep Sched(pending_list) $objid
```

```
                # Acknowledge task
                sched_update $objid
                # Ask for a scheduling pass as soon as possible
                sched_post_tpass
        }

        }

        return
}
```

The following two additional global Tcl variables are defined before the `Tentry` callback is invoked:

- `Us`. This global Tcl array contains various scheduling values. It is described in more detail in the next step.
- `Obj_$objid`. Each attribute of each user task object owned by the scheduler is represented in Tcl by an array with a name of the form `Obj_$objid`; `$objid` is the object ID of the task (such as, `t6`).

  Unfortunately, Tcl arrays with variable names are difficult to access directly in Tcl scripts. Therefore, it is usually more convenient to associate the name of the array with a local name for the duration of the function. This is achieved by using either the Tcl function `upvar #0` or the NQE Tcl function `globvar`. See standard Tcl documentation for a full description of these functions.

The following important functions have been introduced in this step:

Tcl function	Description
`sched_log` *message* [*severity*]	

Log a message to the log file.

message is the message to log.

severity describes the severity level of the message. It may be one of the following:

`i`	Informational
`w`	Warning
`e`	Error
`f`	Fatal

If a severity level is provided, the message is prefixed in the log file by a string showing that severity. If no severity level is provided, the message is logged without a prefix.

The following example

```
sched_log "My message" w
```

logs a message similar to the following:

```
11/08/95 16:36:51 WARN: My message
```

The function does not return any value.

`sched_post_tpass`

Request a scheduling pass.

This function causes the scheduling algorithm to be called as soon as possible. In other words, as soon as the `Tentry` callback completes, the `Tpass` callback is invoked.

The function does not return any value.

`sched_update` *objid updatearr*
or
`sched_update` *objid updatelist*

This is the main NQE database update function. It must be used to update any attributes in a user task object.

objid is the object ID of the object to be updated.

updatearr is the name of a Tcl array that contains all the attributes to update.

The following example updates the object with ID `t6`, setting attributes `t.state` and `otherattrib`:

```
set upd(t.state Pending)
set upd(otherattrib 23)
sched_update t6 upd
```

The equivalent syntax is as follows:

```
sched_update t6 [list t.state Pending
otherattrib 23]
```

The function does not return any value. If an error occurs, Tcl traps it and deals with it elsewhere.

10.2.7 Step 5: Write a `Tpass` Callback Function

Write the `Tpass` scheduling algorithm.

The `Tpass` callback is called as follows:

- Once upon receipt of `Tinit` or upon startup
- Regularly (as defined by the `Tpass` callback's return value)
- Soon after any call to `sched_post_tpass`

> **Note:** The `Tpass` callback is not called when the state of any LWS changes. This means that the scheduler may not react to the startup of a new LWS until the next regular scheduling pass.

The `Tpass` callback should attempt to find a user task that can be scheduled on an LWS. In other words, the main scheduling algorithm should be coded here.

Only one task should be scheduled in any one pass, and no internal counts or lists should be updated by this function. These jobs should be left to the `Tentry` and `Tnewowner` callbacks. This ensures that lists and counts are based solely on the contents of the NQE database.

The simple scheduler algorithm is as follows:

```
#
# Register the function
#
sched_set_callback Tpass fnc_Tpass

#
# Define function fnc_Tpass
#      Argument: None
#
proc fnc_Tpass {} {
       global Obj_config Obj_me Us Lwsid
       global Sched

        sched_trace "TPASS..."

       # Skip if there are no lws running
       if { $Us(lwsruncount) <= 0 } {
          sched_trace "TPASS: No running LWS"
          return $Sched(passrate)
       }

       # Loop through tasks
       foreach taskid $Sched(pending_list) {

              # Find an ordered list of target LWS
              set lwstouselist [getrrlist]

              foreach lws $lwstouselist {

                    # Has the lws any room?
                    if { [info exists Sched(total $lws)] && \
                     $Sched(total_$lws) >= $Sched(total_lws_max)} {
                           #This LWS is already at its limit
                           continue
                    }

                    # Schedule the task to this LWS
```

```
                        sched_log "TPASS: Scheduling $taskid\
                             to $lws ($Lwsid($lws))"
                        set upd(t.state) Scheduled
                        set upd(t.sysclass) $Us(lwsclass)
                        set upd(t.sysname) $lws
                        sched_update $taskid upd

                        # Update round-robin index
                        incr Sched(rrindex)

                        # Ask for another pass as soon as possible
                        return 0

                  }
        }
        return $Sched(passrate)
}
```

The `Us` global Tcl array contains useful scheduling information. The following array items are defined:

- `lwsclass`, which is the name of the LWS class. This is needed to assign ownership of a user task to an LWS. It is always set to `lws`.
- `lwscount`, which is set to the number of LWS system tasks configured. Not all LWSs configured are necessarily running.
- `lwslist`, which is set to the names of the LWS system task objects configured. The number of names in this list equals `lwscount`.
- `lwsruncount`, which is set to the number of LWS system tasks currently running (that is, in `Running` state).
- `lwsrunlist`, which is set to the names of the LWS system tasks currently running.
- `monitorclass`, which is the name of the monitor class. This is needed to assign ownership of a user task to the monitor when the task is completed. It is always set to `monitor`.
- `monitorname`, which is the name of the monitor. This is needed to assign ownership of a user task to the monitor. It is always set to `main`.

- `schedulerclass`, which is the name of the scheduler class. This is needed to assign ownership of a user task to another scheduler. It is usually set to `scheduler`.
- `schedulername`, which is the name of this scheduler. For the tutorial example scheduler, the name will be `simple`.

Note: `sched_update` may change any attribute. The following attributes are often modified:

Name of attribute	Description
`t.state`	New state of the task. This serves both as information and as a verification for the system task that owns this task. When changing ownership to a new system task, the state must be set to a value known to that system task.
`t.sysclass`	Name of the class of the system task object that is to own this task. The classes may be obtained from the global Tcl array `Us`.
`t.sysname`	Name of the system task object that is to own this user task. The name is, by default, `main` for the scheduler and monitor, and is usually the name of the host for the `lws`.

The function to obtain an ordered list of LWSs by the round-robin method (`getrrlist`) is as follows; this function is in the scheduler released with NQE, in the file `local_sched.tcl`:

```
proc getrrlist {} {
      global Us Sched
      # Round-robin tasks to the current active LWS list
      #
      if { $Us(lwsruncount) <= $Sched(rrindex) } {
             #
             # Round-robin index is greater than the current number
             # of active LWSs, reset to 0
             #
             set Sched(rrindex) 0
      }

      # Create the LWS to use list in round-robin order
      #
      return [concat \
            [lrange $Us(lwsrunlist) $Sched(rrindex) end] \
            [lrange $Us(lwsrunlist) 0 [expr $Sched(rrindex) - 1]] \
      ]
}
```

10.2.8 Step 6: Write a `Tnewowner` Callback Function

Write the `Tnewowner` callback function.

The `Tnewowner` callback is called as follows:

- Once for every user task owned by any LWS, immediately after `Tinit` or upon startup.
- Whenever the system task ownership of a task changes away from the scheduler. In other words, the `Tnewowner` callback is invoked after `sched_update` has been used to assign a user task to an LWS to run or to a monitor to be disposed.

The `Tnewowner` callback should update any lists based on the new owner of the task. When the function returns, the scheduler will unset the in-memory Tcl global array of the user task because the task will no longer be owned by the scheduler. The following code writes the `Tnewowner` callback function:

```
#
# Register the function
#
sched_set_callback Tnewowner fnc_Tnewowner

#
# Define function fnc_Tnewowner
#      Arguments:   objid - object ID of task
#                   oclass - new owner class
#                   oname - new owner name
#
proc fnc_Tnewowner { objid oclass oname } {
      global Obj_config Obj_me Us Lwsid
      global Sched

      # Associate global variable Obj_$objid with variable obj
      upvar #0 Obj_$objid obj
      lg_log "TNEWOWNER: objid = $objid, owner now $oclass/$oname"
      # See which type of system task now owns the task
      switch $oclass {
      lws {
             #
             # A LWS now owns task. This occurs immediately after
             # changing the owner of the task to a lws (i.e. after
             # scheduling a task to run in Tpass).
             #
             # Remove it from the pending list
             nqe_listrem Sched(pending_list) $objid
             # Add task ID to list of scheduled tasks
             nqe_listrep Sched(schedule_list) $objid
             #Update counts
             if { ! [info exists Sched(total $oname)] }{
                    set Sched(total_$oname) 0
             }
             incr Sched(total $oname)
             sched_trace "LWS $oname run count increased to\
                   $Sched(total_$oname)"
      }
      monitor {
             # Monitor now owns task. This occurs when a task
             # has completed and is never to run again.
             # Remove it from the lists.
             nqe_listrem Sched(pending_list) $objid
             nqe_listrem Sched(schedule_list) $objid
      }

      }

      return
}
```

The following additional functions were introduced in this section (step 6):

Tcl function	Description
`nqe_listrem`	*listname element* Remove an element from a list.

This function searches for and deletes an element of a list. No action is performed if the list does not exist or if the element is not found.

`nqe_listrep` *listname element*

Adds an element in a list.

If the list does not yet exist, this function creates the list and adds the element to the list. If the list exists, this function adds the element to the list. Otherwise, it performs no action.

10.2.9 Step 7: Write a `Tentry` Callback Function for User Tasks Returned from an LWS

Add to the `Tentry` callback.

When a user task that is owned by an LWS completes or fails in some way, it is reassigned to the scheduler. This action allows the scheduler to update local counts and possibly requeue the task to run again.

Tasks to be rerun enter the scheduler in the same way as new tasks.

In this simple scheduler, the scheduler will assign any task in `Completed`, `Failed`, `Aborted`, or `Terminated` state to the monitor unmodified. A more sophisticated scheduler could analyze the reasons for completion and perform some other action.

Merge the following code into the `Tentry` function that you have already defined:

```
proc fnc_Tentry { objid } {

...
       # Note use of -regexp to allow "|" in cases
       switch -regexp $state {

       New {
              ...
       }

       Pending {
              ...
       }

       Completed|Failed|Aborted|Terminated {
```

```
                # Task finished
                # Send to the monitor.
                sched_log "$objid: Task has completed ($state)" i
                # Update the LWS lists/counts
                # Note that task attribute t.lastlws
                # contains the name of the lws which owned the task
                if { [lsesarch -exact $Sched(schedule_list) $objid] \
                   >=0 } {
                      incr Sched(total_$obj(t.lastlws)) -1
                      sched_trace "LWS $obj(t.lastlws) run count\
                          reduced to $Sched(total_$obj(t.lastlws))"
                }
                set upd(t.sysclass) $Us(monitorclass)
                set upd(t.sysname) $Us(monitorname)
                sched_update $objid upd
                sched_post_tpass
        }
        default {
                # Catch-all for unknown states.
                sched_log "$objid: Unknown state,\
                       $state. Task disposed"
                sched_update $objid [list
                       t.state $state \
                       t.sysclass $Us(monitorclass) \
                       t.sysname $Us(monitorname) \
                ]
        }

        }

        return
}
```

The `t.lastlws` attribute is defined automatically and contains the name of the LWS that last ran this task.

10.2.10 Step 8: Run the New Scheduler

The following two mechanisms exist for running the scheduler:

1. Invoke the `nqedb_scheduler.tcl` script interactively as a command in a terminal by using the following command:

   ```
   nqedb_scheduler.tcl name
   ```

   This command is useful while debugging. Logging and tracing output is sent to the screen.

2. Use the `nqedbmgr start scheduler.simple` command. This command sets the process in the background and creates a log file in the log directory under NQE's spool directory (see the `nqeinfo` file variable `NQE_SPOOL`). This command is useful when the scheduler is in production.

Before invoking the scheduler, set a trace level so that scheduling trace messages are reported; this will aid debugging. Use the `nqedbmgr ask` or `nqedbmgr post event` command to set the trace level. For information about trace levels and how to set them, see Section 10.2.10.2.

This tutorial uses the first mechanism to invoke the scheduler; to do this, simply type the following command:

```
nqedb_scheduler.tcl simple
```

The first argument is the name of the scheduler. If no name is supplied, `main` is assumed. Output and trace messages will appear on the screen.

To submit a job that will be scheduled by scheduler `simple`, use the `scheduler` attribute of the `cqsub` command, as follows:

```
cqsub -d nqedb -la scheduler=simple
```

10.2.10.1 Tcl Errors

It is very possible that the Tcl interpreter will encounter syntax errors while a local scheduler is being developed.

All Tcl errors are trapped. If the error was due to a failure in the connection to the NQE database server, the scheduler enters a retry state. All other Tcl errors cause the scheduler to abort immediately. A description of the error and a Tcl stack trace are placed in the log file (or printed to the screen if the scheduler is being run interactively).

A significant advantage to Tcl is the ability to quickly correct this type of problem.

10.2.10.2 Trace Levels

It is possible to increase the amount of information logged by the scheduler by setting its `trace_level` attribute. Its value is a mask of the types of tracing required. Values may be separated by commas. The following list provides some useful trace-level names:

Trace-level name	Usage
`sched`	Local scheduler trace messages; this type of trace is reserved for use by local schedulers.
`pr`	Process `fork/exec` tracing.
`gen`	General system task tracing.
`db`	NQE database function tracing.
`0`	Disable tracing; this is the default trace-level setting.

You can set the trace level while the system object is running or when it is not running. To set the trace level while the system object is running, use the `nqedbmgr ask` or the `nqedbmgr post event` command. The `nqedbmgr ask` command is the same as the `nqedbmgr post event` command, except that it waits for a response. The following example sets the trace level to `sched` on the simple scheduler as it runs:

```
nqedbmgr ask scheduler.simple trace_level sched
```

The `nqedbmgr post event -o` command sets the trace level whether or not the system object is in a `Running` state. The following example sets the trace level to `sched` on the simple scheduler whether or not it is running:

```
nqedbmgr post event -o scheduler.simple trace_level sched
```

10.2.10.3 Examining Tcl Variables

The `DUMP_VARS` event causes the contents and names of a running scheduler's global Tcl variables to be logged. The event value is a glob pattern to match the global Tcl variable name or value.

For example, to ask the simple scheduler to dump out the contents of the `Sched` array, use the following command:

```
nqedbmgr ask scheduler.simple dump_vars Sched
```

To dump out all variables, use the following command:

```
nqedbmgr ask scheduler.simple dump_vars "*"
```

The quotation marks are needed if the command is passed on a UNIX command line.

10.2.11 Step 9: Add Configuration Enhancements

The following simple addition to the `Tinit` callback allows modification of configuration:

```
proc fnc_Tinit {} {
   global Obj_config Obj_me

...
   # Global run limit per LWS
   set Sched(total_lws_max) 5
   catch {set Sched(total_lws_max) $Obj_me(total_lws_max)}
   sched_trace "TPASS: Global per-LWS run limit:\n      $Sched(total_lws_max) "

   # Scheduler pass rate (when idle)
   set Sched(passrate) 60
   catch {set Sched(passrate) $Obj_me(passrate)}

...

   return 1
}
```

The `total_lws_max` and `passrate` attributes can be overridden by setting attributes with the same names in the system task's object (`Obj_me`). The simple scheduler loads its own system task object on startup or after receiving the `TINIT` event.

Update the system task's object as follows:

```
nqedbmgr
% update scheduler.simple {total_lws_max 2}
% exit
```

Restart the scheduler or post the `Tinit` event. The new maximum should now be in force.

10.2.12 Tracking down Problems

This section describes some common errors you may experience and provides suggested solutions.

```
FATAL: Failure occurred.  Was not loss of connection
FATAL: Don't know what to do here.  Exiting
```

This shows a general problem encountered by a system task. A Tcl error description and Tcl stack trace follow the messages.

```
FATAL: TINIT reports fatal error
```

The local scheduler `Tinit` function returned 0, signifying that it could not initialize correctly. Check the local scheduler's `Tinit` code.

`FATAL: System object` *abc.xyz* `must be created before starting up this system task`

An attempt was made to start a system task before its system task object was inserted into the database. Insert its object by using the `nqedbmgr create` command before starting the system task.

`FATAL: System task file lock could not be obtained.`
`Currently held by PID` *xxxx*
`INFO: Another system task with ID s`*xx* `may be running`

This signifies that a system task of the same ID is already running on this host. Check for a process with this PID on the system.

```
FATAL: Configured time syntax for "xyz" is bad:  expr
```

A global or system task attribute representing a time value (for example, `purge.task_age`) is not a valid mathematical expression. Check the attribute (using `nqedbmgr select -f` *obj*) and modify it. The system task should be restarted.

```
WARN: Callback Tpass is not defined.  No regular scheduling
passes will occur
```

Your local scheduler has not registered a `Tpass` function. This means that no regular scheduling passes can ever occur. Edit the scheduler, and either restart the scheduler or send it a `TINIT` event.

`WARN: System object xyz host name attribute (`*hostA*`) different from this host (`*hostB*`)`

This warning is issued if a given system task is started on a machine other than the machine used for its previous invocation (this means system file locks cannot be used to ensure that multiple system tasks are not running). It may occur as a result of moving the mSQL database to another machine.

`ERROR: DB_CONNECT: Failed.  Retry in 10 secs...`

The system task could not connect to the NQE database. This may be due to one of the following:

- Incorrectly configured `nqeinfo` file; check that the variables `MSQL_SERVER` and `MSQL_TCP` are set to the host name and port running the mSQL server.
- The mSQL server is not running; check that the `msqld` process is running and that the `nqedbmgr` command `status db` executes successfully.
- Connection was refused by the mSQL server; check the `msqld` log file. It may give a security authorization reason for the connection failure.

`ERROR: Cannot find the user scheduler file:` *xyz*

You have specified a local scheduler file that does not exist. Check the `nqeinfo` file variable `NQEDB_SCHEDFILE` and the scheduler's system task attribute, `s.schedfile` (which takes precedence), for the file name given. If the file name is not an absolute path name, it is assumed to be relative to the NQE `bin` directory.

`INFO: Interrupt occurred`

An interrupt was received by the system task. This generally occurs when the scheduler is being run interactively, and

CONTROL-C is pressed. The scheduler will shut down and must be restarted.

10.3 Attribute Names and nqeinfo File Variable Names

This section lists and describes all predefined attribute names for the nqeinfo file and for NQE database objects.

10.3.1 nqeinfo Variables

Table 18 describes variables that may be defined in the nqeinfo file. The table also indicates its default value if it has one.

Table 18. Variables That May Be Defined in the nqeinfo File

Name of variable	Description	Notes
MSQL_DBDEFS	Location and name of mSQL table definitions for NQE database. The file specified by this variable is used as input to mSQL to create tables for the first time. Once the NQE database is created, the file is no longer required.	Default value is $NQE_BIN/nqedb.struct. Needed by the node configured as the NQE database server only.
MSQL_DBNAME	Name of mSQL database. The mSQL daemon may serve multiple databases. Each database comprises one or more SQL tables. After connecting, NQE database clients such as cqsub and system tasks will select a database based on this name.	Default value is nqedb. Value must not be greater than 8 characters in length. Needed by the node configured as the NQE database server only.
MSQL_HOME	Location of mSQL database files. This directory specifies the location on disk of the data stored by the mSQL daemon.	Default value is $NQE_SPOOL. mSQL automatically uses the msqld directory below this. Needed by the node configured as the NQE database server only.

Name of variable	Description	Notes
`MSQL_SERVER`	Name of mSQL daemon server host. This variable contains the name (plus optional colon followed by TCP port) of the host that is running the mSQL daemon. Only master server machines can run the daemon and only one should be defined for the network.	No default is defined. May be overridden by setting the environment variable `MSQL_SERVER`. Needed by all NQE machines.
`MSQL_TCP_PORT`	Name of mSQL daemon server port. This contains the name or number of the TCP port on which the mSQL daemon listens.	Mandatory if no port specified in `MSQL_SERVER`. Default is 603. May be overridden by setting port number in `MSQL_TCP_PORT` . If `MSQL_SERVER` is set with a port number, the value of `MSQL_TCP_PORT` is ignored. Needed by all NQE machines.
`NQEDB_IDDIR`	Name of directory to contain the LWS ID files. The ID directory contains files used for communication between an NQS and its local LWS system task.	Default is `$NQE_SPOOL/nqedb_iddir`. Needed by nodes configured as the NQE database server, NQS server, and LWS.
`NQEDB_SCHEDFILE`	File specification of local scheduler Tcl script. This variable may be set to give an alternative name (relative to `NQE_BIN`) or the full path name of the Tcl script used to implement local scheduling algorithms.	Default if not specified is `local_sched.tcl`. Overridden by defining the attribute `s.schedfile` in the scheduler's system task object. Used by the node configured as the Scheduler.

10.3.2 Event Objects

The event object type is `e`. Event objects are used to communicate among tasks (both system and user). Attributes are described in Table 19:

Table 19. Event Objects Attributes

Name of attribute	Description	Notes
`type`	Object type (`e`)	
`e.ack`	This field is used by the system to record when a system task has acknowledged the existence of the user task object it now owns. `0` means that this task has changed owner but that the new owner has not acknowledged it. `1` means that the new owner has acknowledged ownership.	
`e.name`	A name for this event.	See Table 20, page 292, for list of predefined event names.
`e.response`	Optional response associated with the event. This is set when an event is acknowledged.	If not explicitly set, is set to `""`.
`e.targid`	Object ID of the target task intended for this event. It may be a user task or system task ID.	
`e.targtype`	Object type of the target of this event.	
`e.value`	Optional value for event.	If not explicitly set, is set to `""`.
`id`	Unique ID for the task. In mSQL, this is in the form `e`*n*, where *n* is an integer.	Allocated when the task is first inserted into the NQE database.
`sec.ins.dbuser`	NQE database user name of originator of this task.	See Section 9.9, page 248.
`timein`	Time object was inserted into NQE database.	
`timeup`	Time object was last updated.	

Table 20 lists the events that are defined by default:

Table 20. Events Defined by Default

Event name	Value	Description
DUMP_VARS	Glob pattern for variables to dump	Dump to the log all variable names and values matching the glob pattern supplied. For debugging.
SIGDEL	Signal name or number	Send the signal to the process if it is running. Otherwise, delete the process. This event is posted by qdel/cqdel.
SHUTDOWN	None	Shut down the system task.
TRACE_LEVEL	Trace level	Set the trace level of the target system object to the value passed.
TINIT	None	Reinitialize the system task. This event causes the system task to leave Running state and enter connected state, where it clears and reloads all information. The principal use of this event is to restart the scheduler.

10.3.3 User Task Attributes

The user task object type is t. This object is used to contain all information concerning a job to run on NQS. Attributes are described in Table 21:

Table 21. User Task Object Attributes

Name of attribute	Description	Notes
att.account	Account name or project name.	cqsub option: -A
att.compartment	Compartment names.	cqsub option: -C
att.seclevel	Security levels.	cqsub option: -L
att.time.sched	Request run time.	cqsub option: -a
file.1.name	Standard output file specification name.	cqsub option: -o
file.2.name	Standard error file name.	cqsub option: -e

Name of attribute	Description	Notes
`file.3.name`	Job log file name.	`cqsub` option: `-j`
`file.eo`	Merge standard error and standard output files.	`cqsub` option: `-eo`
`file.jo`	Job log control (`-J m|n|y`).	`cqsub` option: `-J`
`file.ke`	Keep standard error file on execution machine. The option is set if *value* is 1 and is not set if *value* is 0 or attribute is not set.	`cqsub` option: `-ke`
`file.kj`	Keep job log file on execution machine. The option is set if *value* is 1, and is not set if *value* is 0 or the attribute is not set.	`cqsub` option: `-kj`
`file.ko`	Keep standard output file on execution machine. The option is set if *value* is 1, and is not set if *value* is 0 or attribute is not set.	`cqsub` option: `-ko`
`id`	Unique ID for the task. In mSQL, this will be of the form `t`*n*, where *n* is an integer.	Mandatory attribute. Allocated when the task is first inserted into the NQE database.
`lim.core`	Core file size limit.	`cqsub` option: `-lc`
`lim.data`	Data segment size limit.	`cqsub` option: `-ld`
`lim.mpp_m`	Per-request MPP memory size limit	`cqsub` option: `-l mpp_m`
`lim.mpp_p`	Per-request MPP processor elements.	`cqsub` option: `-l mpp_p`
`lim.p_mpp_m`	Per-process MPP memory size limit	`cqsub` option: `-l p_mpp_m`
`lim.mpp_t`	Per-request MPP time limit.	`cqsub` option: `-l mpp_t`
`lim.p_mpp_t`	Per-process MPP time limit.	`cqsub` option: `-l p_mpp_t`
`lim.ppcpu`	Per-process CPU time limit.	`cqsub` option: `-lt`
`lim.ppfile`	Per-process permanent file size limit.	`cqsub` option: `-lf`
`lim.ppmem`	Per-process memory size limit.	`cqsub` option: `-lm`
`lim.ppnice`	Per-process nice value.	`cqsub` option: `-ln`
`lim.pptemp`	Per-process temporary file size limit.	`cqsub` option: `-lv`
`lim.prcpu`	Per-request CPU time limit.	`cqsub` option: `-lT`

Name of attribute	Description	Notes
`lim.prfile`	Per-request permanent file size limit.	`cqsub` option: `-lF`
`lim.prmem`	Per-request memory size limit.	`cqsub` option: `-lM`
`lim.prq`	Per-request quick file space limit.	`cqsub` option: `-lQ`
`lim.prtemp`	Per-request temporary file size limit.	`cqsub` option: `-lV`
`lim.shm`	Per-request shared memory limit.	`cqsub` option: `-l shm_lim`
`lim.shm_seg`	Per-request shared memory segments.	`cqsub` option: `-l shm_seg`
`lim.stack`	Stack segment size limit.	`cqsub` option: `-ls`
`lim.tape.a`	Per-request drive limit for tape type `a`.	`cqsub` option: `-lUa`
`lim.tape.b`	Per-request drive limit for tape type `b`.	`cqsub` option: `-lUb`
`lim.tape.c`	Per-request drive limit for tape type `c`.	`cqsub` option: `-lUd`
`lim.tape.d`	Per-request drive limit for tape type `d`.	`cqsub` option: `-lUc`
`lim.tape.e`	Per-request drive limit for tape type `e`.	`cqsub` option: `-lUe`
`lim.tape.f`	Per-request drive limit for tape type `f`.	`cqsub` option: `-lUf`
`lim.tape.g`	Per-request drive limit for tape type `g`.	`cqsub` option: `-lUg`
`lim.tape.h`	Per-request drive limit for tape type `h`.	`cqsub` option: `-lUh`
`lim.work`	Working set size limit.	`cqsub` option: `-lw`
`mail.begin`	Send mail upon beginning execution. The option is set if *value* is 1 and is not set if *value* is 0 or the attribute is not set.	`cqsub` option: `-mb`
`mail.end`	Send mail upon ending execution. The option is set if *value* is 1, and is not set if *value* is 0 or the attribute is not set.	`cqsub` option: `-me`
`mail.rerun`	Send mail upon request rerun. The option is set if *value* is 1, and is not set if *value* is 0 or the attribute is not set.	`cqsub` option: `-mr`
`mail.transfer`	Send mail upon request transfer. The option is set if *value* is 1, and is not set if *value* is 0 or the attribute is not set.	`cqsub` option: `-mt`
`mail.user`	User name to which mail is sent; default is the user submitting the request.	`cqsub` option: `-mu`

Name of attribute	Description	Notes
`nqs.export`	Export environment variables. The option is set if *value* is 1, and is not set if *value* is 0 or the attribute is not set.	`cqsub` option: `-x`
`nqs.nochkpnt`	Do not checkpoint; not restartable. The option is set if *value* is 1, and is not set if *value* is 0 or the attribute is not set.	`cqsub` option: `-nc`
`nqs.norerun`	Do not rerun. The option is set if *value* is 1, and is not set if *value* is 0 or the attribute is not set.	`cqsub` option: `-nr`
`nqs.pri`	Intraqueue request priority.	`cqsub` option: `-p`
`nqs.queue`	Queue name for submission. If not set or set to "default", the LWS will use a default queue (defined in its system task attribute `default_queue`).	`cqsub` option: `-q`
`nqs.re`	Writes to the standard error file while the request is executing. The option is set if *value* is 1, and is not set if *value* is 0 or attribute is not set.	`cqsub` option: `-re`
`nqs.ro`	Writes to the standard output file while the request is executing. The option is set if *value* is 1, and not set if *value* is 0 or attribute is not set.	`cqsub` option: `-ro`
`nqs.shell`	Batch script shell name; initial shell when a job is spawned.	`cqsub` option: `-s`
`sec.ins.clienthost`	Client host name.	Host name used for file validation; name must be in the `.rhosts` or `.nqshosts` file.
`sec.ins.clientuser`	Client user name.	User name used for file validation; name must be in the `.rhosts` or `.nqshosts` file.
`sec.ins.dbuser`	NQE database user name of originator of this task.	See Section 9.9, page 248.
`sec.ins.hash`	Hash value for `sec.ins.*` values.	

Name of attribute	Description	Notes
`sec.ins.tuser`	Target user specification.	
`sec.ins.tuser.pwe`	Target user specification (encrypted).	
`sec.target.user`	Target user for a particular LWS.	This attribute is created by the LWS after the task is submitted locally.
`script.0`	Script file contents.	This may be a very long attribute and may not be displayable by `nqedbmgr`.
`t.ack`	This field is used by the system to record when a system task has acknowledged the existence of the user task object it now owns. `0` means that this task has changed owner but that the new owner has not acknowledged ownership. `1` means that the new owner has acknowledged ownership.	
`t.class`	The class name for this task.	Always set to `USER`.
`t.failedlws`	List of LWS names where this job has failed. This may be used to determine whether the task should be rescheduled to a given LWS.	
`t.failure`	Failure message.	
`t.lastlws`	Name of LWS that last submitted this task to NQS. This is updated by the LWS when the task is completed or has failed.	This is set by the scheduler when passing ownership to an `lws`.
`t.lastsched`	System task name of the scheduler that owned and scheduled the task.	This is set by the scheduler when passing ownership to an `lws`.
`t.message`	Messages concerning success or failure of job.	
`t.name`	A name for this task. The name is not used except for information. By default, it is set to the name of the NQS job, as specified by `-r`.	`cqsub` option: `-r`
`t.reqid`	The current NQS request ID for this task. This ID may change if the task is submitted to one LWS's NQS, fails, and is then submitted to another.	Not set until task has been submitted to NQS.

Name of attribute	Description	Notes
`t.state`	State of task. This serves both as information and as a checkpoint for the system task that owns this task. When changing owner from one system task to another, the state must be set to a value known to the new system task.	
`t.substate`	Substate.	May be `""`.
`t.sysclass`	Name of class of system object that owns this task. The class may be `scheduler`, `lws`, or `monitor`.	
`t.sysname`	Name of system object that owns this task. The name is `main` for the scheduler and `monitor` and the name of the host for the `lws`.	
`timein`	Time at which the task was inserted into the NQE database.	
`timeup`	Time at which the task was last updated.	
`type`	Task type (`t`).	Always set to `t` for user tasks.
`user.`*attr_name*	Job attributes.	`cqsub` option: `-la`

10.3.4 System Task Object

The system task object type is `s`. One system task object is created for each system task that is to run in this system. On a typical system, there is one scheduler system task, one monitor system task, and one LWS system task for each NQS server node. Attributes are described in Table 22:

Table 22. System Task Object Attributes

Name of attribute	Description	Notes
`default_queue`	Provides the name of the NQS queue to which jobs are submitted by default.	Used by the LWS only.
`id`	Unique ID for the object. In mSQL, this will be of the form `s`*n*, where *n* is an integer.	Mandatory attribute. Allocated when the task is first inserted into the NQE database.
`qmgrnow`	If set and `true`, requests that the LWS follow any submission to NQS with the command `qmgr>scheduler request` *xyz* `now`, where *xyz* is the NQS request ID. This forces NQS to run a job, even if local NQS scheduling parameters deny it.	LWS specific.
`run_mechanism`	Defines the validation mechanism used to check the target user against the client user for local submission to NQS.	Used only by the LWS. If it is set to `File`, LWS checks for local `~/.nqshosts` or `~/.rhosts` file; if it is set to `Passwd`, LWS checks for target password.
`s.class`	The class name of this system task. Possible values at present are as follows: • `lws` (LWS system task) There may be more than 1 instance of these. • `scheduler` (scheduler system task) • `monitor` (monitor system task)	
`s.host`	Name of host on which this system task is running.	
`s.name`	The system task's name. This is `main` for the scheduler and monitor and is, by default, the host name for the LWS.	
`s.pid`	UNIX PID of the system task.	Set to `-1` when system task exits.

Name of attribute	Description	Notes
`s.schedfile`	Local scheduler (user scheduler) file. The file may include the absolute path name or the path name relative to the NQE `bin` directory. This attribute is always recognized, no matter which local scheduler is running.	Used by the scheduler only. Default is `local_sched.tcl`. See Section 10.2.4, page 267, for usage.
`s.state`	State of system task: • Exited: The system task is no longer running. • Aborted: The system task aborted. Check log file for reason. • Running: The system task is running. • Down: The monitor has marked this system task as down.	
`sec.ins.dbuser`	NQE database user name of the originator of this task. For a system task, this will usually be `root`.	See Section 9.9, page 248.
`trace_level`	Current trace level for the system task.	Default setting is `0`.

10.3.5 Global Configuration Object

The global configuration object type is `g`. This special object type exists for the purposes of containing global configuration data. There is only one instance of this object type in the NQE database. Attributes are described in Table 23:

Table 23. Global Configuration Object Attributes

Name of attribute	Description	Notes
`hb.systask_timeout _period`	Time before monitor system task marks a given system task as `Down`. The value may be a mathematical expression. Units are seconds.	Default value is `90`.
`hb.update_period`	Time between each update of the heartbeat. Each system task will update the NQE database within this period. The heartbeat is used by the monitor to determine whether a system task is still alive. The value may be a mathematical expression. Units are seconds.	Default value is `30`.
`id`	Unique ID for the object. In mSQL, this will be of the form `g`*n*, where *n* is an integer.	Allocated when the object is first inserted into the NQE database.
`purge.systask_age`	The monitor system task will delete (purge from the NQE database) all acknowledged events concerning system tasks that have not been updated within the specified time period. The value may be a mathematical expression. Units are seconds.	Default value is `24*60*60`
`purge.task_age`	The monitor system task will delete (purge from the NQE database) all user tasks that are finished and have not been updated within the specified time period. The value may be a mathematical expression. Units are seconds.	Default value is `24*60*60`
`s.class`	Object class (`config`).	
`s.name`	Object name (`main`).	
`sec.ins.dbuser`	NQE database user name of the originator of this object (usually `root`).	See Section 9.9, page 248.

Name of attribute	Description	Notes
`status.access`	Type of access granted to users when attempting to obtain summary information about jobs in the NQE database. Value can be one of the following: • `all`: Allows user to select summary information for all tasks. • `own`: Allows user to select only information about tasks submitted by the user. A user may only see full information for tasks submitted by that user.	Default value is `all`.
`sys.ev_rate`	Rate at which system tasks check for new events targeted at them or at tasks they own. This attribute may be a mathematical expression. Units are seconds.	Default value is `10`.
`sys.purge_rate`	Rate at which the monitor checks the age of a completed job. This attribute may be a mathematical expression. Units are seconds.	Default value is `5*60`.
`sys.st_rate`	Rate at which system tasks check for new tasks, changes in task state, and so on. This attribute may be a mathematical expression. Units are seconds.	`10`
`sys.upd_rate`	Rate at which LWS system tasks check for messages from NQS. This attribute may be a mathematical expression. Units are seconds.	`10`
`timehb`	Time of last heartbeat.	
`timein`	Time at which object was inserted into the NQE database.	
`timeup`	Time at which object was last updated.	
`type`	Object type (`s`); always set to `s` for system tasks.	

10.4 Tcl Built-in Functions

Tcl was chosen to become part of NQE scheduling for the purpose of providing a simple, standard interpreted language with which administrators can write schedulers. It is very portable and can be interfaced to C very easily.

The manager command, `nqedbmgr`, is simply Tcl invoked interactively. All `nqedbmgr` commands are Tcl functions (defined in `nqedbmgr.tcl`).

In order to use Tcl effectively in system tasks, various built-in functions (that is, functions written in C and linked with Tcl) were implemented. Table 24 provides a brief description of these functions.

Table 24. Tcl Built-in Functions

Tcl built-in function	Description
`globvar`	Because Tcl arrays with variable names are difficult to access directly in Tcl scripts, `globvar` is used to associate the name of the array with a local name for the duration of the function.
`lg_init`, `lg_config`, `lg_log`, `lg_trace`	Logging and tracing functions.
`pr_create`, `pr_set`, `pr_get`, `pr_execute`, `pr_signal`, `pr_wait`, `pr_clear`	Provide a mechanism for `fork`ing and `exec`ing processes, and changing context to another user.
`nqe_after`, `nqe_pause`	Implementation of a simple asynchronous timing loop. These functions are used to give the system tasks event-driven characteristics.
`nqe_curtime`	Returns the elapsed seconds since UNIX Epoch.
`nqe_daemon_init()`	Initializes the caller as a daemon process. It makes a `setid()` call to create a new session with no controlling tty and also ignores `SIGHUP` signals.
`nqe_strtime`	Converts UNIX Epoch to a human-readable string.

Tcl built-in function	Description
`nqe_lock`, `nqe_locked`, `nqe_unlock`	File locking functions.
`nqe_pipe`	UNIX `pipe()` implementation.
`nqe_rmfile`	Unlink or remove a file.
`nqedb_connect`, `nqedb_insert`, `nqedb_select`, `nqedb_getobj`, `nqedb_update`, `nqedb_delete`, `nqedb_disconnect` `nqedb_insert_event`, `nqedb_insert_task`, `nqedb_cancel_event`	Tcl implementations of the NQE database C APIs.
`nqe_getenv`, `nqe_getvar`	Obtain environment variable/ `nqeinfo` file variables.
`nqe_getid`	Returns effective/real UIDs and GIDs.
`nqe_gethostname`	Returns local host name.

Table 25 lists other useful functions implemented in Tcl:

Table 25. Additional Functions Implemented in Tcl

Tcl built-in function	Description
`nqe_listrem` *listname element*	Removes an element from a list. This function searches for and deletes an element of a list. No action is performed if the list does not exist or if the element is not found.
`nqe_listrep` *listname element*	Replaces an element from a list. This function adds an element to a list if the list does not yet exist or if the element is not in the list; otherwise, it performs no action.
`sched_log` *message [severity]*	Logs a message to the log file. message is the message to the log. severity describes the severity level of the message. It may be one of the following: `i` Informational `w` Warning `e` Error `f` Fatal If a severity level is provided, the message is prefixed in the log file with a string showing that severity. If no severity level is provided, the message is logged without a prefix. There is no return.

Tcl built-in function	Description
`sched_post_tpass`	Requests a scheduling pass. This function causes the scheduling algorithm (`Tpass`) to be called as soon as possible. There is no return value.
`sched_set_callbac` *callback functionanme*	Registers the Tcl function, *functionname*, with callback *callback*. *callback* may be one of the following: `Tinit` Initialize `Tentry` Task entry `Tpass` Scheduling pass `Tnewowner` New owner There is no return value.
`sched_update` *objid updatearr* or `sched_update` *objid updatelist*	This is the main NQE database update function. It must be used to update any attributes in a user task object. *objid* is the object ID of the object to be updated. *updatearr* is the name of a Tcl array that contains all the attributes to update. Alternatively, you may pass a list of attribute names and values in *updatelist*. The function does not return any value. If an error occurs, Tcl traps it and deals with it elsewhere.

Using `csuspend` [11]

NQE lets you put to use the unused cycles on servers generally reserved for interactive use. This process is sometimes called *cycle stealing*. This chapter describes how to use the `csuspend`(8) command to enable or disable batch work, depending on interactive use. The following topics are discussed:

- `csuspend` options
- `csuspend` functionality
- `csuspend` examples

11.1 csuspend Options

The syntax of the `csuspend` command is as follows:

```
csuspend [-i value] [-l loopcount] [-o value] [-p period] [-s] [-t
   interval]
```

The options are described as follows:

Option	Description
`-i` *value*	Specifies the input character rate (in bytes) that is compared to the input character rate of `sar -y`. If the current tty input rate on the server equals or exceeds *value*, `csuspend` initiates suspension of batch activity. If the current tty input rate on the server is less than *value*, `csuspend` initiates resumption of batch activity. If you omit *value*, the default is 1.
`-l` *loopcount*	Specifies the number of periods for which `sar` collects interactive activity data to determine if NQE will resume, be suspended, or continue to be suspended. The default number of periods (*loopcount*) is 7. See the `-p` *period* option for information about specifying the length of a period.

`-o` *value*	Specifies the output character rate (in bytes) that is compared to the output character rate of `sar -y`. If the current tty input rate on the server equals or exceeds *value*, `csuspend` initiates suspension of batch activity. If the current tty input rate on the server is less than *value*, `csuspend` initiates resumption of batch activity. If you omit *value*, the default is 100.
`-p`*period*	Specifies the length of time (in seconds) that `sar` collects interactive activity data. The default *period* is 10 seconds. The minimum setting allowed for *period* is 5 seconds; if *period* is set to less than 5, it is automatically set to 5 seconds.
`-s`	Specifies that the NLB object `NLB_BSUSPEND` attribute is set to `YES` when batch activity is suspended and `NO` when batch activity is resumed. NLB policies can use this value to distribute work to the server. The default is not to update the NLB database.
`-t` *interval*	Specifies the interval (in seconds) that `csuspend` waits before it begins to check tty status. The check of tty status takes 60 seconds. The default value to wait between checks is 10 seconds.

11.2 csuspend Functionality

The `csuspend` command uses `sar` to monitor interactive terminal session (tty) activity on a specific server. You must invoke `csuspend` on the server. If tty activity equals or exceeds the input or output thresholds set by using the `-i` or `-o` options, NQE suspends NQS batch activity. All queues are disabled, and no additional batch work is accepted. This process allows interactive users the full use of the server. When interactive activity drops below the thresholds specified, batch jobs are resumed. Using the `-l` and `-p` options gives an administrator greater control over how `sar` is used and the frequency of the check on whether to suspend or start NQE.

To use `csuspend` in Network Load Balancer (NLB) policies, invoke it with the `-s` option. Using this option ensures that the NLB server is aware when NQS is suspended.

The `csuspend` command runs in a scan loop until the command is terminated with a `CONTROL-C` or the `kill-15` signal. It is often run in the background with standard output (`stdout`) and standard error (`stderr`) directed to a file.

The `csuspend` utility has no log file; by default, `stdout` and `stderr` are written to the terminal screen. If you want to log `csuspend` activity, redirect the output to a file.

When you stop `csuspend`, it resets queue states and exits. Even if the NQS server is shut down (by using `nqestop`(8) or `qstop`(8)), `csuspend` continues to run. When NQS is started (by using `nqeinit`(8) or `qstart`(8)), `csuspend` will recognize it.

Note: `nqeinit` and `nqestop` do not start or stop the `csuspend` process.

When `csuspend` detects interactive use, it performs the following functions:

- Reports tty input and output rates.
- Disables and suspends all pipe queues on the server.
- Disables and suspends all batch queues on the server.
- Suspends all running batch requests.
- Logs a suspend message to the request's job log.
- If you used the `-s` option, the NLB object `NLB_BSUSPEND` attribute is set to `YES`.

When `csuspend` detects cessation of interactive use, it performs the following functions:

- Reports tty input and output rates.
- Resumes all running batch requests.
- Resumes and enables all batch queues on the server.
- Resumes and enables all pipe queues on the server.
- Logs a resume message to the request's job log.
- If you used the `-s` option, the NLB object `NLB_BSUSPEND` attribute is set to `NO`.

The `csuspend` command uses the following commands and files; they **must** be present on the server on which you invoke `csuspend`:

- `awk`(1)
- `getopts`(1)
- `ksh`(1)
- `nqeinfo` file
- `qmgr`(8)
- `qstat`(1)
- `sar`(1)
- If you use the `-s` option, the `nlbconfig`(8) command and the NLB must be available.

Because `csuspend` needs access to NQS spool directories and uses `qmgr`(8) commands, you must invoke it from `root`.

Because `csuspend` is a `ksh` script, you can tailor it for individual site and server needs. The default values for its options appear after the initial comment lines at the start of the script file (`$NQE_ETC/csuspend`). To change the defaults or parts of the command logic, edit this file.

11.3 csuspend Examples

The following example invokes `csuspend` and places it in the background.

```
nohup csuspend -i 5 -o 50 -s -t 30 &
```

If the `sar -y` tty input character count equals or exceeds 5 (set by using `-i`), or if the output character count equals or exceeds 50 (set by using `-o`), batch request are suspended. When the rates fall below these values, batch request are resumed. The `NLB_BSUSPEND` attribute is updated (set by using `-s`). `csuspend` examines interactive tty activity every 30 seconds.

The following example invokes `csuspend` with all default values, which are echoed to `stdout`:

```
# csuspend
  Mar 30 23:50:58 - csuspend initiated for nqe server latte
  Mar 30 23:50:58 - using options: i=1,o=100,s=NO,t=10
  Mar 30 23:50:58 - using NQE in /usr
  Mar 30 23:50:58 - using NLB server latte
  Mar 30 23:52:09 - auto resume processing cycle initiated
  Mar 30 23:52:09 - input character rate=0, output character rate=0
  Mar 30 23:52:10 - found 1 batch queues to process
  Mar 30 23:52:10 - processing batch queue nqebatch
  Mar 30 23:52:11 - found 2 pipe queues to process
  Mar 30 23:52:11 - processing pipe queue nqenlb
  Mar 30 23:52:12 - processing pipe queue gate


(character input occurs)

  Mar 31 08:07:28 - auto suspend processing cycle initiated
  Mar 31 08:07:28 - input character rate=4, output character rate=0
  Mar 31 08:07:29 - found 2 pipe queues to process
  Mar 31 08:07:29 - processing pipe queue nqenlb
  Mar 31 08:07:30 - processing pipe queue gate
  Mar 31 08:07:31 - found 1 batch queues to process
  Mar 31 08:07:31 - processing batch queue nqebatch
```

The following example uses the `-l` and `-p` options to suspend or enable batch processing based upon the interactive use as determined by calls to `sar`.

```
csuspend -l 11 -p 60 &
```

If *period* is set to 60 and *loopcount* is set to 11, `sar` collects data over a 60-second period, and if the interactive activity is low enough during 10 (11 minus 1) periods, then NQE resumes. Otherwise, NQE is suspended or remains suspended.

Job Dependency Administration [12]

This chapter discusses administrative tasks that job dependency requires. The following topics are covered:

- Managing the NLB database for job dependency
- Multiple NLB servers and job dependency

12.1 Managing the NLB Database for Job Dependency

Job dependency event information remains in the NLB database until it is explicitly cleared either by you or the user who issued the `cevent`(1) command.

In order to keep the size of the NLB database manageable, you should clear unwanted events regularly. To list and clear events, use the `cevent` command.

User `root` can list any event from any event group by using the following command:

```
cevent -la
```

The following example shows how the output looks when you use the default delimiter:

```
# cevent -la
Time:                      Group:         Name:          Value:
---------------------------------------------------------------
Mon Dec 11 12:18:15 1995  g1             n1             <NONE>
Mon Dec 11 12:18:18 1995  g2             n1             <NONE>
Mon Dec 11 12:18:21 1995  g4             n1             <NONE>
```

You can parse and analyze the output from the `cevent -la` command from within a script to produce a set of events to delete. To do this, it may be useful to redefine the delimiter character. If you use the `-d` to specify the delimiter `!`, the output would look like the following example:

```
# cevent -la -d '!'
Mon Dec 11 12:18:15 1995!g1!n1!<NONE>
Mon Dec 11 12:18:18 1995!g2!n1!<NONE>
Mon Dec 11 12:18:21 1995!g4!n1!<NONE>
```

To delete all events from all groups, use the following command:

```
cevent -ca
```

To delete all events from a group named `g1`, use the following command:

```
cevent -c -g g1
```

To delete the events `n1` and `n2` from the group `g1`, use the following command:

```
cevent -c -g g1 n1 n2
```

12.2 Multiple NLB Servers and Job Dependency

A potential problem exists when you use job dependency and multiple NLB servers. This section describes the problem and methods you can use to prevent it from occurring.

Multiple or redundant NLB servers work like disk mirroring. The NLB collector processes on each node configured to run as a collector send the same system and request status information to all defined NLB servers. If an NLB server goes down and then comes back up, the collectors will soon resend their data to it and make it current.

If you specify more than one NLB server in your `nqeinfo` file, the `cevent` command tries to send information to the first server listed. If that server is temporarily down (or unavailable due to network problems), the `cevent` command tries the second server, and so on. If the first server subsequently comes up, the `cevent` command will send information to it (rather than the one to which it previously sent information).

For example, assume that your `nqeinfo` file lists two NLB servers, as follows; quotes are required in the syntax:

```
#
# NLB_SERVER - Location of NLB server host
#
NLB_SERVER="host1 host2"
```

If a user posts job dependency information on the server that uses this `nqeinfo` file, the collector on the server tries to send the information to the NLB server database on host `host1`. If `host1` does not respond, the collector tries to send the information to `host2`. If both fail, `cevent` returns an error.

If a user posts an event when host1 is unavailable, the event is posted to host2. If host1 becomes available, any cevent commands will use it because it is first on the list in the nqeinfo file. If cevent tries to read an event that was posted to host2, it will actually look for the event on host1. The event will not be read because it is not posted on host1.

To prevent these problems, users can set their NLB_SERVER environment variable to one machine before they invoke cevent. This will force all subsequent cevent invocations to use the specified server. The cevent -w command will wait a specified number of seconds for a response, so the user can wait for transient network, NLB, or system problems on the server to be corrected.

You can use the following methods to avoid the problem:

- If the NLB server on host host1 or host1 goes down, ensure that the NLB server on host1 stays down until some quiet period. Then bring the NLB server on host2 down. Move the NLB data on host2 to host1 by copying all obj_* files in /usr/spool/nqe/nlbdir on host2 to /usr/spool/nqe/nlbdir on host1. Then bring both NLB servers back up.
- This method requires a manual procedure or a script. If the NLB server on host host1 or host1 goes down, log in to host2 as root during a quiet period and dump (using nlbconfig -odump) all of the cevent objects for all event groups in the NLB database on host2. Log in to host1 as root and update all events with this data (by using nlbconfig -oupdat).
- You can rename the cevent script, create a new script in the same directory, and name the new script cevent. The new script then sets the NLB_SERVER environment variable and calls cevent under the new name. This guarantees that cevent will use the NLB server that you specify in the script.

FTA Administration [13]

The File Transfer Agent (FTA) can be used to transfer files to and from any node in the Network Queuing Environment (NQE) in either immediate (interactive) or queued mode. It also provides network peer-to-peer authorization (NPPA) of users, which is a more secure method than `ftp`(1) provides.

This chapter provides the following information about managing FTA:

- FTA overview
- Configuration
- Administration

13.1 FTA Overview

The FTA facility consists of the three components shown in Table 26.

Table 26. File Transfer Components

Component	Program	Location of description
User interface	`ftua`(1) and `rft`(1)	Section 13.1.1, page 318
Common services	`fta`(8)	Section 13.1.2, page 318
File transfer services	`ftad`(8) and `nqsfts`(8)	Section 13.1.3, page 319 and Section 13.1.3, page 319

The three-tier stack of file transfer components consists of the file transfer user interface, the common services, and the file transfer services (FTS).

The `ftua`(1) and `rft`(1) commands use FTA to transfer files. A *file transfer request* is an object that contains information describing the file transfer operations that FTA will perform on behalf of a user.

The `fta` command performs the actions required for reliable file transfer between hosts. These functions include transfer retry, status, and recovery. File transfer management options are also provided.

Using `fta`, the file transfer user commands are given a common interface to the file transfer services that operate on various networks. A domain name selects the file transfer services for a given host name.

13.1.1 User Interface

The `ftua`(1) and `rft`(1) commands are the user interfaces to FTA services. They transfer files to and from a remote host by using the file transfer services supported by the underlying file transfer agent. FTA supports file transfer services that use TCP/IP protocols. In fact, `ftua` operates like the TCP/IP `ftp`(1B) file transfer program. The `ftua` utility supports autologin through the use of the `.netrc` file to allow access to accounts without specifying a password.

Note: The `ftua` autologin feature is prohibited on UNICOS and UNICOS/mk systems when the multilevel security feature is enabled.

See the *NQE User's Guide*, publication SG–2148, for an overview of the user interface commands. The `ftua`(1) and `rft`(1) man pages contain complete reference information.

13.1.2 Common Services

The `fta`(8) program provides the following FTA *common services*, which are common to any protocol used to transfer files:

- Creation of the file transfer requests
- Management of file transfer requests
- Invocation of the file transfer services
- Display of status information
- Management of security keys for NPPA
- Status about file transfers

File transfer requests are created when user agents invoke `fta` with the `-service` option.

Management functions must be invoked specifically. The primary management function is transfer recovery, which restarts transfers that failed or were interrupted.

To examine the status of requests, use the `-queue` option. You will get more complete output by also including the `-verbose` option.

> **Note:** For UNICOS and UNICOS/mk systems running the multilevel security feature, in order for `fta` to operate correctly, you must install `fta` with a specific security environment. This can be done by using the `spset` command as follows to ensure that the `fqueue` directory is wildcarded (see the `spset`(1) man page for additional information); the `NQE_FTA_QUEUE` variable is set in the `nqeinfo` file:
>
> ```
> spset -l63 NQE_FTA_QUEUE
> ```

The `fta`(8) man page contains complete reference information.

13.1.3 File Transfer Services

The file transfer service establishes the connection to the remote system. If the file transfer service being used is anything other than `ftp`, the invocation of the service results in an additional process. When the file transfer service completes, FTA updates the status of the request file and exits.

The FTS operation proceeds as follows:

1. Establishes a network connection to the remote host
2. Passes user validation information to the remote host
3. Performs file transfer operations
4. Closes the network connection

The file transfer service for the TCP/IP protocol FTP is integrated into the `fta` program. See the `fta`(8) man page for further details.

The `nqsfts`(8) program provides the file transfer service for exclusive use by NQS. This service supports only the `NPK_MVOUTFILE` subset of the NQS protocol. NQS can use this service through `fta`(8) to transfer job output files between NQS systems. This service supports the NQS hosts by using the TCP/IP network. See the `nqsfts`(8) man page for further details.

Figure 24 shows the relationships among the FTA components.

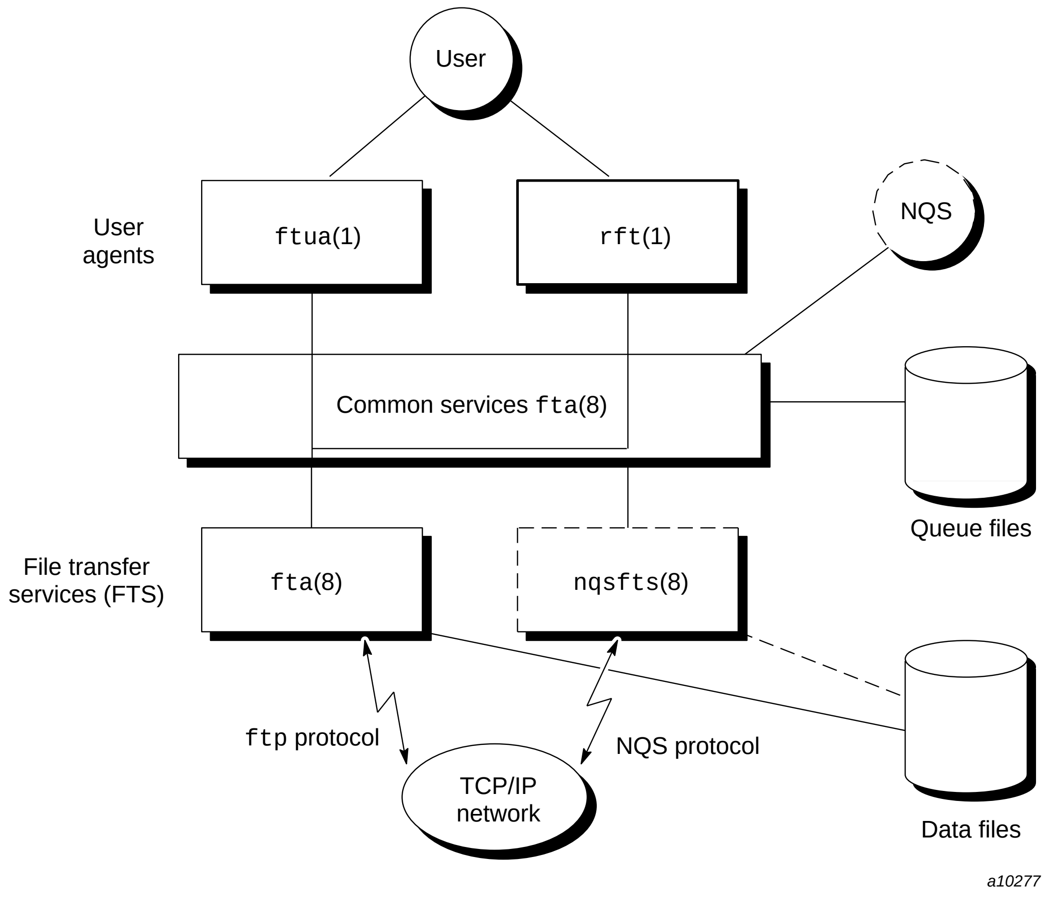


Figure 24. Component Relationships in FTA

13.2 Configuration

FTA configuration information is stored in the Network Load Balancer (NLB) during NQE startup. This FTA configuration applies to all hosts in the NQE except UNICOS systems that run only the NQE subset. For information about configuring FTA for UNICOS systems that run only the NQE subset, see Section 13.2.5, page 327.

The NPPA S-keys must be configured on each separate system. For information about configuring S-keys, see Section 13.2.8.1, page 331.

FTA configuration information is contained in the NLB database. Two FTA object types exist in the NLB database: `FTA_CONFIG` and `FTA_DOMAIN`.

Note: If you change or add a port, you must edit system files on each system, as described in Section 13.2.7, page 329, in order for the FTA daemon (`ftad`(8)) to be invoked when a transfer request is received.

FTA administration must take place on each system, even if the systems are reading the same configuration from the NLB. For more information about FTA administration, see Section 13.3, page 336.

This section explains how to view and change the FTA configuration.

13.2.1 Viewing the FTA Configuration

FTA configuration is contained in the NLB. The FTA configuration is contained in the NLB as objects of the following two attribute types:

Type	Description
`FTA_CONFIG`	Global type
`FTA_DOMAIN`	Describes each domain

Also, the `queue` parameter is defined in the `nqeinfo` file (see Section 13.2.2).

FTA configuration may be viewed by using the `nlbconfig` command.

13.2.2 Global FTA Configuration Parameters

Note: For UNICOS systems that run only the NQE subset (NQS and FTA components), see Section 13.2.5, page 327, for information about configuring FTA.

If you issue the `nlbconfig -odump -t FTA_CONFIG -n DEFAULT` command, the output is as follows:

```
%nlbconfig -odump -t FTA_CONFIG -n DEFAULT
###############################################
# creation date: Mon Nov 13 16:45:09 1995
# created by: luigi
# object timestamp: Tue oct 10 15:16:15 1995
########
object FTA_CONFIG ("DEFAULT") {
    FTA_MODE         = 384;
    FTA_RESTART_LIMIT = 30;
    FTA_ADMIN_USER   = "root";
    FTA_ADMIN_GROUP  = "bin";
```

Note: `FTA_MODE` is represented in the NLB database as a decimal number (`384`).

The `nlbconfig` `FTA_CONFIG` object type, displayed with the command `nlbconfig -odump -t FTA_CONFIG`, is defined as follows:

Parameter	Description
`queue "`*path*`"`	Defines the directory that holds transfer request control files while they are present in the queue. This parameter is always required. You can change this parameter in the `nqeinfo` configuration file. **Note:** On UNICOS and UNICOS/mk systems running the multilevel security feature, the queue directory must be wildcarded to ensure that users can write a queue file to the database directory. Use the `spset`(1) command to do this.
`FTA_RESTART_LIMIT=`*limit*	Defines the maximum number of FTS processes (*limit*) that will be started during transfer recovery. This figure is independent of any domain-specific limit. If *limit* is `none` or unspecified, limits are not applied.
`FTA_MODE=`*file-mode*	Defines the file permission mode of the files in the queue directory. The default permission mode is 0600. Octal values must have a leading 0.

`FTA_ADMIN_USER "`*username*`"|`
`FTA_ADMIN_GROUP "`*groupname*`" [,...]`

Defines the users, groups, or a combination of the two that can perform FTA management functions on behalf of other users. Without this parameter, only the super user has this privilege.

`FTA_STATUS_USER "`*username*`"|`
`FTA_STATUS_GROUP "`*groupname*`" [,...]`

Defines the users, groups, or a combination of the two that can perform FTA status functions on behalf of other users. Without this parameter, only the super user has this privilege.

13.2.3 FTA Domain Configuration

Note: For UNICOS systems that run only the NQE subset (NQS and FTA components), to configure FTA, see Section 13.2.5, page 327.

If you issue the `nlbconfig -odump -t FTA_DOMAIN -n inet` command, the output is as follows:

```
%nlbconfig -odump t FTA_DOMAIN -n inet
###############################################
# creation date: Wed Nov 15 12:45:39 1995
# created by: luigi
# object timestamp: Fri Nov 13 09:34:31 1995
########
object FTA_DOMAIN ("inet") {
    FTA_PORT         = 605;
    FTA_RESTART_LIMIT = 30;
    FTA_FTS          = "ftp";
    FTA_FLAGS        = "nopassword";
```

The `nlbconfig` `FTA_DOMAIN` object type, displayed with the command `nlbconfig -odump -t FTA_DOMAIN` is defined as follows:

Parameter	Description
`FTA_FTS=`*class*	Defines the file transfer service class to be associated with a domain. The valid class identifiers are described below. The

validity of other domain configuration parameters depends on the `FTA_FTS` class used. This parameter is required. See Table 27 for a list of classes that may be used.

`FTA_RESTART_LIMIT=`*limit*

Defines the maximum number of FTS processes (*limit*) that will be started for a domain during transfer recovery.

`FTA_GATEWAY="`*hostname*`"`

Defines the name of a gateway host.

`FTA_PORT=`*number*

Defines the port number to connect to on the remote system. The default is `605` on NQE systems. By default, the `ftad` daemon listens on port `605`; the `ftpd` demon listens on port `21`.

`FTA_PATH="`*fts_path*`"`

Defines the path of the FTS process to be invoked.

`FTA_FLAGS=`*flag* `[,`*flag*`]...`

Defines the flags to be used with the domain. An exclamation point (!) can precede a flag to explicitly declare a flag to be off. See Table 28 for a list of flags that can be used with this parameter.

The *class* associated with the `FTA_FTS` parameter can be any of the following, as shown in Table 27:

Table 27. `FTA_FTS` Class Parameters

Class	Description
`common`	Defines a domain in which the FTS uses a standard FTS interface module. The path of the FTS program is required.
`dap`	Defines a domain for a remote DECnet (DAP) gateway, which is controlled by using the built-in FTP FTS program. The *hostname* associated with the `FTA_GATEWAY` parameter must be running an FTP/DAP gateway. A gateway TCP/IP *hostname* name is required and defined using the `FTA_GATEWAY` parameter.
`ftp`	Defines an Internet FTP domain. The `ftp` FTS program is built into `fta`. An additional path parameter is not required.

The *flags* associated with the `FTA_FLAGS` parameter can be any of the following, as shown in Table 28:

Table 28. `FTA_FLAGS` Parameters

Flag	Description
`cancel`	Specifies that user agent cancellation will be performed using the FTS.
`debug`	Specifies that the FTS be run with debug tracing enabled. The FTS program determines the exact function of the FTS debug tracing flag.
`nopassword`	Specifies that a password need not be provided before a file transfer request is accepted; that is, the password is optional. NPPA requires this flag.
`nopermanent`	Specifies that any permanent error be treated as transient. This will enable the transfer to be recovered.
`nostart`	Specifies that a request should not be started when the user agent completes a session with the `fta` server; instead, a request is processed when transfer recovery is initiated.
`notify`	Specifies that user agent notification will be performed using the FTS.

`privileged`	Specifies that the FTS will be invoked as a privileged process, inheriting all privileges endowed to the `fta` recovery process. This usually results in the FTA process running as `root` (user ID 0).
`trace`	Specifies that the FTS will be run with protocol tracing enabled. The FTS program determines the exact function of the FTS protocol tracing flag.

13.2.4 Changing the FTA Configuration

You can change the default FTA configuration using the following procedure:

1. List the FTA objects in the NLB database with the following commands:

   ```
   # nlbconfig -olist -t FTA_CONFIG
   ------ type: FTA_CONFIG count: 1 ------
   DEFAULT
   # nlbconfig -olist -t FTA_DOMAIN
   ------ type: FTA_DOMAIN count: 4 ------
   inet            ftp           nqe          nqs-rqs
   ```

   The output indicates that there is one FTA configuration object named `DEFAULT`. There are four FTA domain objects named `inet`, `ftp`, `nqe`, and `nqs-rqs`. When the count is more than one, specify a specific name with the `nlbconfig -t` *type* `-n` *name* command.

2. To display all the attributes of the object type `FTA_CONFIG` with the name `DEFAULT`, issue the following command:

   ```
   nlbconfig -odump -t FTA_CONFIG -n DEFAULT
   ```

   You will receive output similar to the following:

```
###############################################
# creation date: Mon Dec 25 16:45:09 1995
# created by: luigi
# object timestamp: Fri Oct 13 15:16:15 1995
########
object FTA_CONFIG ("DEFAULT") {
    FTA_MODE         = 384;
    FTA_RESTART_LIMIT = 30;
    FTA_ADMIN_USER   = "root";
    FTA_ADMIN_GROUP  = "bin";
```

You can either completely replace this information with new information, or update it to change a part of it. Note that the `FTA_MODE` is represented in the NLB database as a decimal number (`384`).

3. To change the information, create a file (called, for example, `odat.1`) that contains the attribute(s) you want to change or add, as in the following example:

```
object FTA_CONFIG ("DEFAULT") {
    FTA_ADMIN_USER    = "root, luigi";
}
```

4. Update the object file by using the `nlbconfig -oupdat -f odat.1` command. All of the existing information is retained, and the `FTA_ADMIN_USER` attribute is modified.

5. To completely replace the information in the object file, dump the information in the object file into working file (called, for example, `odat.2`) by using the following command:

```
nlbconfig -odump -f odat.2
```

Edit this file to contain the information you want and replace the entire object file with the following command:

```
nlbconfig -oput -f odat.2
```

13.2.5 FTA Configuration for UNICOS Systems That Run Only the NQE Subset

For UNICOS systems that run only the NQE subset (NQS and FTA components), FTA configuration is not maintained in the NLB. Instead, UNICOS FTA configuration is maintained in the FTA configuration file /*nqebase*/`etc/fta.conf`. To view `fta` configuration for UNICOS systems, use the `fta -config` command. For additional information, see the `fta.conf`(5) man page.

> **Note:** If you change or add a port, you must edit system files on each system, as described in Section 13.2.7, page 329, in order for the FTA daemon (`ftad`(8)) to be invoked when a transfer request is received.

You must add the entry `NQE_FTA_FTACONF=/`*nqebase*`/etc/fta.conf` to the `nqeinfo` file (either by using the `nqeconfig`(8) utility or by manually adding the entry to the end of the file). To initially generate the `fta.conf` file and FTA queue directory, use the `configure.fta` script in the `/`*nqebase*`/examples` directory. This script will use values that are in the `nqeinfo` file to create the proper FTA configuration.

If you issue the `fta -config` command, the output is as follows. The section of the output that begins with `FTA` corresponds to the `FTA_CONFIG` NLB object. The section of the output that begins with `FTS` corresponds to the `FTA_DOMAIN` NLB object. `NQE_FTA_QUEUE` is set in the `nqeinfo` file.

```
% fta -config
FTA
queue="NQE_FTA_QUEUE"
mode=0600
admin_users="root"
admin_groups="bin"
restartlimit=30
FTS
domain="inet"
fts=ftp
restartlimit=30
port=605
flags="nopassword"
```

Table 29 describes how the `fta -config` output corresponds to the output from the `nlbconfig` commands.

Table 29. `fta -config` Output and `nlbconfig` Output Comparison

`fta -config` output	`nlbconfig` output
`mode`	`FTA_MODE`
`admin_users`	`FTA_ADMIN_USER`
`admin_groups`	`FTA_ADMIN_GROUP`
`restartlimit`	`FTA_RESTART_LIMIT`
`domain`	`FTA_DOMAIN`
`fts`	`FTA_FTS`
`port`	`FTA_PORT`
`flags`	`FTA_FLAGS`

Note: `nlbconfig` represents `FTA_MODE` as a decimal number, rather than the octal number that is displayed when you use the `fta -config` command.

13.2.6 Changing the `ftpd` Port Number

If you are running a TCP/IP transport service from another vendor (that is, one other than the remote system TCP/IP), and the `ftp` daemon is already configured on port `21`, you must add the `ftpd` service to an unused port number that is greater than 256 and less than 1024.

To add the `ftpd` service to an unused port number, you must ensure that this matches the `ftp/fta` port number on the local system.

13.2.7 Editing System Configuration Files

FTA itself is configured during the installation of NQE. However, if you change or add a port, you must edit the following system files on each machine in order for `inetd` to invoke `ftad` when a transfer request is received:

```
/etc/services
/etc/inetd.conf
```

Optionally, you might edit `/etc/syslogd.conf`.

Add the following line to `/etc/services`:

```
fta             unused_port_number/tcp
```

Add the following lines to `/etc/inetd.conf`:

```
#<service_name> <socket_type> <proto> <flags> <user> <server_pathname> <args> <options>

#
# FTAD daemon
#
fta  stream  tcp  nowait root /nqebase/bin/ftad ftad options
```

You can add the `-d` option to the end of the last line. The `-d` option allows the server to log messages to `syslogd`.

If this option is required, ensure that the following line is included in `syslog.conf` (it may already be included if other daemons are already sending messages to `syslogd`):

```
*.debug                                    /var/adm/messages
```

The `inetd` utility rereads its configuration file once when it is started and again whenever it receives the hangup signal (`SIGHUP`). New services can be activated and existing services can be deleted or modified by editing the configuration file, then sending `inetd` a `SIGHUP` signal.

Note: On AIX systems, if you manually edit the `inetd.conf` file, you must synchronize the `inetd.conf` file and the object data manager (ODM) database by using the `inetimp` command.

For additional information about setting port numbers, see the *NQE Installation*, publication SG–5236.

13.2.8 Configuring Network Peer-to-peer Authorization

You can authorize users to transfer files without exposing passwords across the network. This is especially desirable for NQS users who might otherwise have to make their FTA passwords visible in the NQS job file script. Network peer-to-peer authorization (NPPA) is a method of enabling users to open a connection to a remote system without sending a password across the network.

Note: NPPA is enabled by default on all platforms except UNICOS and UNICOS/mk systems. To enable NPPA on UNICOS and UNICOS/mk systems, use the `ftad -N` command in `/etc/inetd.conf`.

NPPA requires the use of `ftua` or `rft` on the local system and `ftad` on the remote system. Both systems must have support for the NPPA process. NPPA only works if you have a flat user name space; that is the user name of an FTA user must be the same on both systems.

The FTA domains `inet` and `nqe` are set up to use NPPA by default. You do not have to make changes to the FTA domain configuration to use NPPA in these domains. If you want to use a different name for NPPA connections, you can add another domain. This procedure is described in Section 13.2.10, page 335.

Each system that is going to use NPPA must be configured with the authentication information that it will send to another system during session setup, and with the authentication information that it will expect to receive from other NPPA systems. This authentication information consists of an S-key (security key) and a password, both of which are alphanumeric strings. The first S-key and password pair defined on each system is designated as the primary S-key and password pair for that system.

When a user starts `ftua` or `rft` and uses a domain configured for NPPA, the system encrypts its primary S-key and password pair and sends it to the system with which it is attempting to establish a connection. The remote system performs a similar encryption for each S-key and password pair on its list of expected authorization information and accepts the connection when it finds a match.

Note: If multiple `ftad`s are configured on different ports, they will all use the NPPA configuration file (`fta.nppa`) pointed to by the `NQE_FTA_NPPAPATH` variable that is set in the `nqeinfo` file.

13.2.8.1 Configuring S-keys

To configure network peer-to-peer authorization, the administrator must establish the primary S-key and password pair for each system. These S-key and password pairs can be the same for all systems, or they may differ.

The first S-key set up on each system becomes the primary S-key, which is sent to remote systems during connection establishment. After defining the primary S-key on a system, the administrator adds all the S-key and password pairs for all the hosts that will be authorized to open NPPA connections to that system.

To configure network peer-to-peer authorization, you must assign S-key and password pairs on all of the systems that will use NPPA.

To create and delete S-keys, you must be `root`.

Note: On UNICOS systems that run only the NQE subset, use the `fta`(8) command to configure S-keys.

To assign S-keys, use the `skey -addkey` *keylabel* command, as follows:

```
# skey -addkey key1
Enter key password
Verification
```

Alternatively, if FTA is running on the system, you can also use the `fta -addkey` command. The effect of the two commands is identical.

To display the keys on the system, use the `-showkey` option, as follows:

```
# skey -showkey

Keys on this system

 1) key1 -  Primary Key
```

To delete S-keys, use the `skey -delkey` *keylabel* command.

> **Note:** For UNICOS systems running NQE 2.0 or earlier in your network, you cannot use `ftua` to connect to a UNICOS platform (whether it is a Cray Research system-to-Cray Research system or a workstation-to-Cray Research system) because UNICOS systems running the earlier versions of NQE do not support `ftad`. NQE version 3.0 running on UNICOS does support `ftad`.

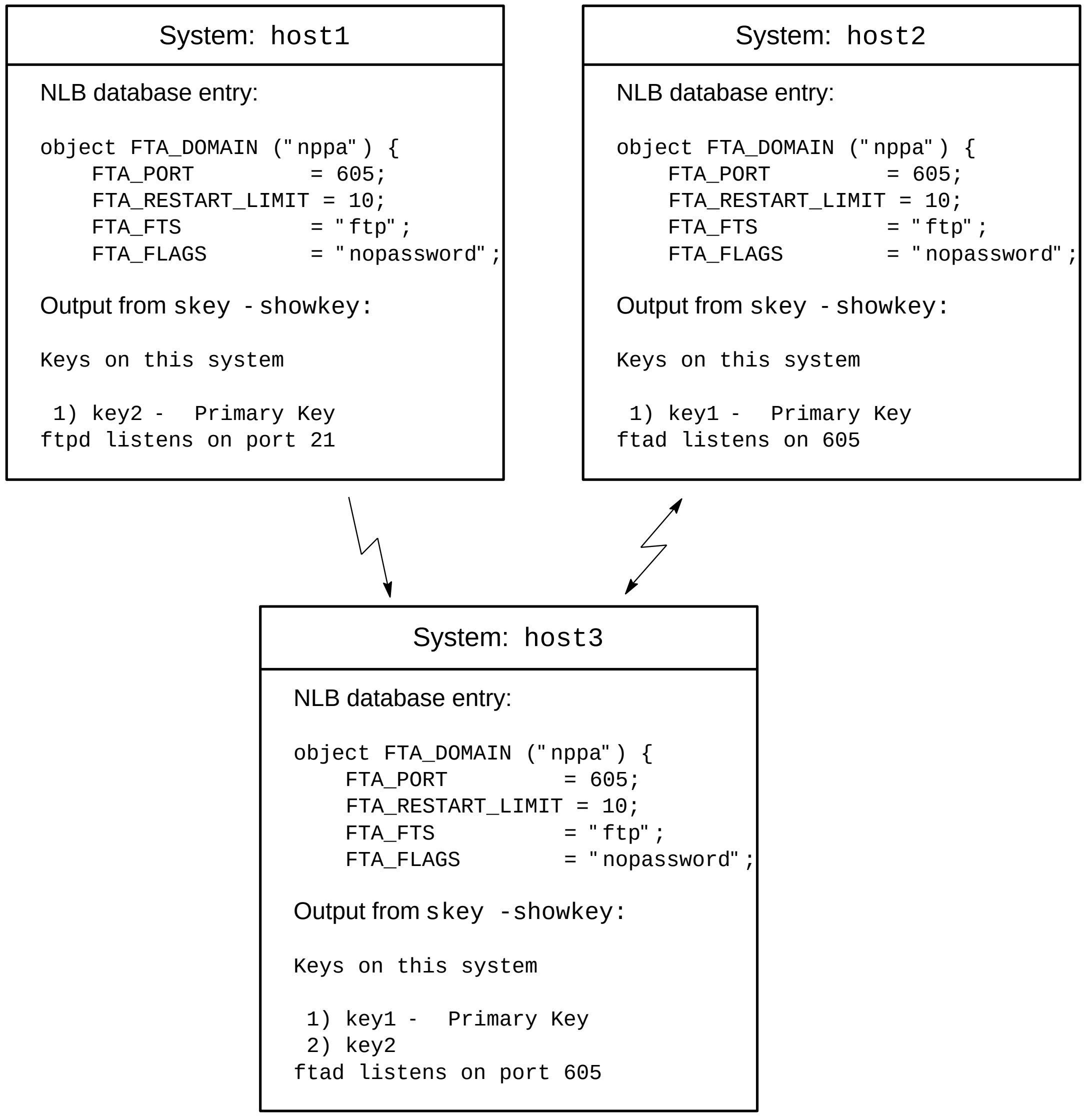
13.2.9 NPPA Example

This example is based on having three systems that you want to communicate by using FTA. The systems are called `host1`, `host2`, and `host3`. They have the following characteristics:

Table 30. Example of NPPA Configuration Characteristics

System	FTS	Port FTS listens on	Port provided for domain `nppa`	S-keys
`host1`	`ftp`	21 (`ftpd`)	605	`key2`
`host2`	`ftp`	605 (`ftad`)	605	`key1`
`host3`	`ftp`	605 (`ftad`)	605	`key1`, `key2`

Figure 25 shows this configuration.



host1:

```
object FTA_DOMAIN ("nppa") {
    FTA_PORT          = 605;
    FTA_RESTART_LIMIT = 10;
    FTA_FTS           = "ftp";
    FTA_FLAGS         = "nopassword";

Keys on this system

 1) key2 -  Primary Key
ftpd listens on port 21
```

host2:

```
object FTA_DOMAIN ("nppa") {
    FTA_PORT          = 605;
    FTA_RESTART_LIMIT = 10;
    FTA_FTS           = "ftp";
    FTA_FLAGS         = "nopassword";

Keys on this system

 1) key1 -  Primary Key
ftad listens on 605
```

host3:

```
object FTA_DOMAIN ("nppa") {
    FTA_PORT          = 605;
    FTA_RESTART_LIMIT = 10;
    FTA_FTS           = "ftp";
    FTA_FLAGS         = "nopassword";

Keys on this system

 1) key1 -  Primary Key
 2) key2
ftad listens on port 605
```

Figure 25. NPPA Configuration Example

In this configuration, you want users to be able to use `cqsub` to submit NQS batch requests from `host3` to `host1` and to have FTA return the output. FTA on `host1` will construct an *authentication packet* that includes the S-key `key2` as one of its components. `host1` transmits its authentication packet to `host3`. On `host3`, two authentication packets are generated, one that uses the S-key `key2` and one that uses the S-key `key1`. Both of the packets generated on `host3` are checked against the packet received from `host1`. The packet that uses the S-key `key2` matches, and the authentication succeeds.

If users initiate an FTA session from `host3` to `host1` that uses the domain `nppa`, the session will fail because no FTA daemon is listening on port `605` on the `host1` system.

If users initiate an FTA session from `host3` to `host1` without specifying a domain, a connection to the `ftp` daemon port (usually `21`) can be made successfully. This session will not use NPPA because the `ftp` daemon does not support NPPA.

Users can use `cqsub` to submit an NQS batch request from `host3` to `host2` and have FTA return the output. FTA on `host2` constructs an authentication packet that includes the S-key `key1` as a component. `host2` transmits this packet to `host3`. On `host3`, two authentication packets are generated, and the one that uses the S-key `key1` matches, and authentication succeeds.

If users initiate an FTA session from `host3` to `host2` that uses the domain `nppa`, the session will succeed because the FTA daemon is listening on the same port number (`605`) on `host2`.

13.2.10 Modifying the NLB Database for NPPA

If you want to create a new FTA domain for users to specify when they use NPPA, you must add that FTA domain to the NLB database. You must have ACL `write` privileges to change the NLB database. The domain must have the following attributes; you can add others if you want:

Attribute	Description
`FTA_FTS = ftp`	File transfer class used by the remote system; `ftp` is an example.
`FTA_PORT =` *nnn*	TCP/IP `ftad` port number on the remote system on which the `fta` daemon is listening; Section 13.2.10.1, page 336 contains an example.

`FTA_FLAGS = nopassword`	Flag (`nopassword`) that informs FTA that NPPA can be used for this domain.

For a complete listing of possible attributes and values, see Section 13.2.1, page 321.

13.2.10.1 Example of Modifying the NLB Database

To add an NPPA domain to the NLB database, create a file (called, for example, `nppa.odat`) such as the following, which adds the domain `nppa` on the local system, with the `ftad` daemon listening on port number `258`:

```
object FTA_DOMAIN ( "nppa" ) {
      FTA_FTS = "ftp";
      FTA_PORT = 258;
      FTA_FLAGS = "nopassword";
}
```

To update NLB object file and add the new domain object to the NLB database, use the following command:

```
nlbconfig -oupdat -f nppa.odat
```

For more details on how to modify the NLB database, see Section 13.2.4, page 326.

13.3 Administration

FTA startup, logging, and recovery functions are not entirely contained within the bounds of the `fta`(8) program. This section explains some of the interactions that occur between the FTA facility and other parts of the operating system. The actions described in this section are done (or a default value is provided) during installation. If you want to make changes, you can use these procedures.

Note: FTA administration must take place on each system, even if the systems are reading the same configuration from the NLB.

13.3.1 FTA System Startup

At system boot time, you can restart any FTA file transfers that stopped when the system shut down by running a recovery process.

Note: Do not start a recovery process before TCP/IP is brought up.

The system startup recovery process is started as follows:

```
fta -recover
```

13.3.2 FTA Recovery

The following sections describe FTA recovery.

13.3.2.1 Byte-level Recovery of Binary Transfers

Binary transfers can be recovered at the byte level. System administrators enable or disable this recovery level by setting the `NQE_FTA_SMART_RESTART` environment variable to be `ON` or `OFF`. The `NQE_FTA_SMART_RESTART` environment variable may be set in the `nqeinfo` file or through the shell. The default value is `ON`.

Note: The configuration variable is ignored if the operation is not taking place in binary mode; restarting will be disabled.

The `FTA_DBY` NLB attribute of the `FTA_ACTION` object appears only after the transfer has begun; it remains even if the transfer is interrupted.

13.3.2.2 Setting the Recovery Interval

FTA does not have a daemon recovery service; that is, the recovery of a queued request is not automatically provided. Therefore, it is recommended that you start a recovery process either at regular intervals or at specific times of the day or week.

If you use the clock daemon, `cron`(8), to start FTA recovery for all users, you must use the super-user `crontab` file. The following example of a `crontab` entry starts an FTA recovery process every 15 minutes, each day of the week. Refer to `crontab`(1) for syntax information.

```
0,15,30,45 * * * *    fta -recover
```

It also is useful to run a recovery process at significant times, such as the following:

- Nightly, to clean up any failures.
- After network failure is resolved. Use `-domain` as a selector to direct the recovery for a specific domain or network.

- After a remote host is rebooted. Use `-host` to recover requests that transfer to or from that specific host.

To recover queued requests for all users of the system, run the `fta -recover` process, using the root user ID or the ID of an FTA administrator.

The following example runs the recovery process on the specific file transfer request entry `aa003yy`; `-debug` is used to monitor the process:

```
% fta -debug -recover -entry aa003yy
fta: debug: getuid=11431, geteuid=11431
fta: debug: Reading config file...
fta: debug: ftsconf: class=1, domain=solaris, gateway=None, path=None, port=510
fta: debug: ftsconf: class=1, domain=osf, gateway=None, path=None, port=535
fta: debug: ftsconf: class=6, domain=nppa_ice, gateway=ice, path=None, port=256
fta: debug: ftsconf: class=1, domain=inet, gateway=None, path=None, port=0
fta: debug: User is an administrator
fta: debug: User can see status
fta: debug: queue: reading dir NQE_FTA_QUEUE
fta: debug: entry aa006bk rejected (name mismatch)
fta: debug: queue: opening file qfaa003yy
fta: debug: entry aa003yy selected
fta: debug: entry aa001bb rejected (name mismatch)
fta: debug: entry aa004WE rejected (name mismatch)
fta: debug: entry aa004YM rejected (name mismatch)
fta: debug: entry aa004dw rejected (name mismatch)
fta: debug: entry aa004fJ rejected (name mismatch)
fta: debug: entry aa0053J rejected (name mismatch)
fta: debug: entry aa0059I rejected (name mismatch)
fta: debug: entry aa005Q0 rejected (name mismatch)
fta: log: aa003yy: starting recovery.
fta: debug: setUserId: uid=11431, euid=11431
fta: debug: ftp_open: connecting to yankee...
fta: debug: ftp_connect: connection up
fta: debug: waiting for initial 220 msg...
fta: debug: got initial 220 msg
fta: debug: doing request
fta: debug: operation Copy (get)
fta: debug: copyio: called
fta: debug: copyio: 2879 bytes transferred
fta: debug: operation completed (status =    20200)
fta: debug: pullstatus: status = 20200
fta: debug: updating/removing request entry
fta: debug: request succeeded (ES_NONE)
fta: log: aa003yy: completed successfully.
fta: debug: rmEntry: aa003yy: request removed
```

13.3.2.3 Types of Failure

The following problems might prevent file transfers from occurring when FTA is used:

- A network route is down.
- A remote system is down.
- User validation information is not valid.
- A file transfer operation is not valid.

These errors are categorized as either transient or permanent.

Type of error	Resulting action
Transient	FTA retains the request file in the request queue directory. Status information showing the reason for the failure is included in the request.
Permanent	FTA notifies the user or user agent (`ftua` or `rft`) with a message containing information that describes the error, and the transfer request is removed from the request queue directory.

When file transfer is successful, FTA removes the request file from the queue directory. You can use the `ftua notify` command to notify the user of successful completion.

A queued request that fails with a transient error can be rerun by invoking a recovery process. Recovery proceeds with the following steps:

1. Scan the queue for entries.
2. Invoke FTA and FTS processes for each request to be recovered.
3. Update the status of the request queue.

If you are `root` or an FTA administrator, you can start a recovery process by executing the following command:

```
fta -recover
```

13.3.2.4 FTA Recovery Limits

If many file transfer requests are queued, recovery can create many processes. This can create a heavy load on the system. To reduce the load, you can limit the number of requests that are recovered at any one time. The `FTA_RESTART_LIMIT` attribute of the `FTA_CONFIG` NLB object controls the number of requests that are started at any one time. The number of requests recovered at any one time, regardless of domain, is controlled by this limit. The following NLB object shows a global configuration restart limit of 30:

```
object FTA_CONFIG ("DEFAULT") {
            FTA_MODE
            FTA_RESTART_LIMIT = 30;
```

To control the load presented by a specific domain, assign a limit on a per-domain basis. This limit is known as a *domain restart limit*. Section 13.2.4, page 326, explains how to change the FTA configuration.

The following example from the NLB object shows that the `inet` domain restart limit as `10`:

```
object FTA_DOMAIN ("inet") {
      FTA_FTS = "common";
      FTA_RESTART_LIMIT = 10;
}
```

Both the `fta` and `domain` restart limits are applied to the queue during recovery. If either limit is reached, the number of requests recovered is constrained. If a domain restart limit is set higher than the `fta` restart limit, the number of requests recovered for that domain is constrained by the `fta` restart limit.

If neither an `fta` nor a `domain` restart limit is defined, no limit is applied. It also is possible to use the keyword `none` in place of a number to explicitly denote that there is no limit.

13.3.2.5 FTA Recovery Strategy

When you start a recovery process, the queue of file transfer requests is sorted before any restart limits are applied. The following criteria determine the order in which requests are recovered. The order of the following list is significant; the first differentiating factor determines the order in which the requests are recovered. Selector options are applied to the queue to filter out the requests chosen by a user.

1. Requests that were never run are favored over those that ran before.
2. Requests that were run least recently are favored over those that were run most recently.
3. Requests that were queued least recently are favored over those that were queued most recently.

The following example shows an application of these criteria. Because requests aa00093 and aa00136 never ran, they are favored over the other requests (rule 1). Because request aa00093 is older, it is placed before aa00136 in an internal recovery list (rule 3). Requests aa00101 and aa00127 have run. Because both have the same run date, they are equally qualified under rule 2; however, aa00101 was queued before aa00127, so aa00101 is favored over aa00127 (rule 3). Recovery would occur in the following order: aa00093, aa00136, aa00101, and aa00127.

QID	Date queued	Date run
aa00093	Thu Nov 16 08:16:35	Never
aa00101	Thu Nov 16 08:18:22	Thu Nov 16 08:28:14 BST 1995
aa00127	Thu Nov 16 08:28:12	Thu Nov 16 08:28:14 BST 1995
aa00136	Thu Nov 16 08:30:43	Never

If the restart limit were set to 3, which limits the number of recovered requests, only the first three requests from this sorted list (that is, aa00093, aa00136, and aa00101) would be recovered.

Suppose that after the recovery the queue state is as follows:

QID	Date queued	Date run
aa00101	Thu Nov 16 08:18:22	Thu Nov 16 09:45:13 BST 1995
aa00127	Thu Nov 16 08:28:12	Thu Nov 16 08:28:14 BST 1995

Requests aa00093 and aa00136 completed, but request aa00101 experienced a transient error and was requeued. If a second recovery starts now, neither request qualifies under rule 1; therefore, queue ordering would be determined by rule 2, the time the requests were last run. In this example, request aa00127 would be placed before aa00101, because aa00127 ran least recently.

13.3.3 Information Log

The NQE_FTA_NLBCOLLECT variable activates logging of transfer information to the NLB. Values may be ON or OFF. The variable may be set in the nqeinfo file or as an environment variable. The default value is ON.

FTA information is logged by the `syslogd`(8) daemon under the facility name `daemon`. Three `syslogd` priorities are used: `notice`, `err`, and `debug`. Normal FTA events are logged with the `notice` priority; FTA errors are logged with the `err` priority, and all events are logged when `debug` is turned on.

Other facilities also use the `syslogd` facility name `daemon`, and a corresponding `syslog.conf` entry might already be in use. Review the `syslog.conf` file for other uses of the `daemon` facility.

The following is a sample entry from a `syslog.conf` file used to direct FTA notice and error logging information to the `/usr/adm/syslog/daylog` system log file:

```
daemon.notice;daemon.err   /usr/adm/syslog/daylog
```

With the addition of the following line, FTA errors also are sent to an operator's terminal:

```
daemon.err  operator
```

A typical entry in a log file that is generated by `fta` using `syslogd` follows:

```
Sep 29 12:27:21 yankee fta[16331]: aa003z7: starting recovery
Sep 29 12:27:22 yankee fta[16331]: User: js Host: yankee Request: aa003z7
  Mode: Immediate
Sep 29 12:27:22 yankee fta[16331]: Remote file: lgn Local
  file: /net/home/palm3/js/temp Size: 879 bytes
Sep 29 12:27:22 yankee fta[16331]: aa003z7: completed successfully
Sep 29 12:28:20 yankee fta[16335]: aa005Q0: cancelled (null operation)
Sep 29 12:28:20 yankee fta[16335]: aa005Q0: failed: Cancelled by user/operator
Sep 29 12:28:26 yankee sendmail[16337]: AA16337:
  message-id=<9309291728.AA16337@yankee.yankeegroup>
Sep 29 12:28:26 yankee sendmail[16337]: AA16337: from=js, size=563, class=0,
  received from local
Sep 29 12:28:27 yankee sendmail[16339]: AA16337: to=js@palm, delay=00:00:06,
  stat=Sent
Sep 29 12:29:18 yankee fta[16342]: aa004YM: cancel: request active
```

Caution: When you invoke FTA recovery, you can use the `-log` option to route the information logged to the standard output of the `fta` recovery process. If this option is used from a tty, you must redirect the standard output. If no redirection or pipe is used, and the output is sent to the controlling tty, some of the log output will be lost. This behavior is a side effect caused by `fta` using the `setjob`(2) system call during the recovery process

13.3.4 FTA Management Functions and Status

FTA management functions include the following:

- Canceling requests
- Deleting requests
- Holding requests
- Releasing requests
- Displaying request status

The super user can perform management functions on any requests that exist in the FTA queue.

If necessary, specific users or groups of users can be authorized to use FTA management functions. In the following NLB object example, user `joe` and the members of the group `operator` are authorized to perform FTA management functions:

```
object FTA_CONFIG ("DEFAULT") {
    FTA_MODE          = 384;
    FTA_RESTART_LIMIT = 30;
    FTA_ADMIN_USER    = "joe";
    FTA_ADMIN_GROUP   = "operator";
```

Specific users or groups of users also can be authorized to display the status of any requests in the FTA queue. In the following NLB object example, user `joe` and the members of the group `staff` can display the status of any request:

```
object FTA_CONFIG ("DEFAULT") {
    FTA_MODE          = 384;
    FTA_RESTART_LIMIT = 30;
    FTA_STATUS_USER    = "joe";
    FTA_STATUS_GROUP   = "staff";
```

Note: Unless explicitly listed, users can perform management functions on, and display the status of, only their own requests.

You can use selector options to filter sets of requests for specific status or management functions (see the `fta` man page). Selectors are provided to filter requests based on the following attributes:

- Time the request was queued
- Domain name
- Host name
- Request owner
- Queue request identifier
- Request status

If you combine different selectors, a request is selected only when it satisfies all attributes specified with the given selectors. If selectors are not specified, all requests are selected. The following example lists all requests that were queued at or before 12 P.M. and that tried to transfer but failed.

```
fta -before 12:00 -failed -queue
```

The following examples use the `-debug` option to show the output from requests to cancel and delete an entry:

```
% fta -cancel -entry aa005Q0 -debug
fta: debug: getuid=11431, geteuid=11431
fta: debug: Reading config file...
fta: debug: ftsconf: class=1, domain=solaris, gateway=None, path=None, port=510
fta: debug: ftsconf: class=1, domain=osf, gateway=None, path=None, port=535
fta: debug: ftsconf: class=6, domain=nppa_ice, gateway=ice, path=None, port=256
fta: debug: ftsconf: class=1, domain=inet, gateway=None, path=None, port=0
fta: debug: User is an administrator
fta: debug: User can see status
fta: debug: queue: reading dir NQE_FTA_QUEUE
fta: debug: entry aa006bk rejected (name mismatch)
fta: debug: entry aa001bb rejected (name mismatch)
fta: debug: entry aa004WE rejected (name mismatch)
fta: debug: entry aa004YM rejected (name mismatch)
fta: debug: entry aa004dw rejected (name mismatch)
fta: debug: entry aa004fJ rejected (name mismatch)
fta: debug: entry aa0053J rejected (name mismatch)
fta: debug: entry aa0059I rejected (name mismatch)
fta: debug: queue: opening file qfaa005Q0
fta: debug: entry aa005Q0 selected
fta: log: aa005Q0: cancelled (null operation).
fta: log: aa005Q0: failed: Cancelled by user/operator.
fta: debug: rmEntry: aa005Q0: request removed
```

```
% fta -delete -entry aa006bk -debug
fta: debug: getuid=11431, geteuid=11431
fta: debug: Reading config file...
fta: debug: ftsconf: class=1, domain=solaris, gateway=None, path=None, port=510
fta: debug: ftsconf: class=1, domain=osf, gateway=None, path=None, port=535
fta: debug: ftsconf: class=6, domain=nppa_ice, gateway=ice, path=None, port=256
fta: debug: ftsconf: class=1, domain=inet, gateway=None, path=None, port=0
fta: debug: User is an administrator
fta: debug: User can see status
fta: debug: queue: reading dir NQE_FTA_QUEUE
fta: debug: queue: opening file qfaa006bk
fta: debug: entry aa006bk selected
fta: debug: entry aa001bb rejected (name mismatch)
fta: debug: entry aa004WE rejected (name mismatch)
fta: debug: entry aa004YM rejected (name mismatch)
fta: debug: entry aa004dw rejected (name mismatch)
fta: debug: entry aa004fJ rejected (name mismatch)
fta: debug: entry aa0053J rejected (name mismatch)
fta: debug: entry aa0059I rejected (name mismatch)
fta: debug: rmEntry: aa006bk: request removed
```

The following examples show the output from requests to display the status of an entry:

```
% fta -queue
--QID-- ---Queued On --- --User-- ---State--- -----Status-----
aa001bb Tue Sep 14 09:19 sparks   Active      UA connected
aa004WE Tue Sep 21 08:54 sparks   Active      Connecting
aa004YM Tue Sep 21 09:00 sparks   Active      Connected
aa004fJ Tue Sep 21 09:36 sparks   Active      Connecting
aa0053J Tue Sep 21 11:52 sparks   Active      Connecting
aa0059I Tue Sep 21 12:09 sparks   Active      Connecting

% fta -queue -verbose
--QID-- ---Queued On --- --User-- --Group--
aa001bb Tue Sep 14 09:19 sparks   rqsgrp
        State: Active
        Status: UA connected
        Priority: 0
        Last Attempt: Tue Sep 14 10:33:22
        Controlling-PID: 27788
        Domain: osf
        Host: alphabet
        Username: sparks
        Copy (get)
                RemoteFile: fta30.1.tar
                LocalFile: /net/home/palm3/sparks/fta.tar

aa004WE Tue Sep 21 08:54 sparks   rqsgrp
        State: Active
        Status: Connecting
        Priority: 0
        Last Attempt: Never
        Domain: inet
        Host: alphabet
```

13.3.5 Transfer Status

FTA provides progress statistics about file transfer status. The total file size, bytes transferred, and estimated time to completion are tracked, and the progress statistics are stored in the NLB. Users and administrators can view this status information by using the `FTA summary` option of the NQE GUI `Status` window. You can change the interval between NLB updates by setting the `NQE_FTA_STAT_INTERVAL` variable, which may be set in the `nqeinfo` file; alternatively, you can override the configured internal value through the `NQE_FTA_STAT_IN` environment variable in the shell. The variable is an integer, which represents the number of seconds between NLB updates. If the `NQE_FTA_STAT_INTERVAL` environment variable is set to 0, progress statistics are disabled. The default value is `2`.

For DFS transfers, the `FTA_DFS` NLB attribute of the `FTA_ACTION` object appears only after the transfer has begun; it remains even if the transfer is interrupted. The `FTA_DFS` NLB attribute is set to `-1` if FTA is unable to determine the file size; this action does not prevent the transfer from continuing.

Configuring DCE/DFS [14]

This chapter describes how to administrate the interoperability of NQE tasks and Distributed Computing Environment (DCE) resources such as the Distributed File System (DFS).

Note: DCE support is optional, and is disabled by default. It is a licensed feature of NQE.

The following topics are discussed in this chapter:

- Enabling DCE support for NQE
- Using a password to submit a task
- Using a forwardable ticket to submit a task
- Checkpoint and restart support
- Support for renewing and refreshing tickets
- User requirements
- Administrating Kerberos and DCE

14.1 Enabling DCE Support for NQE

Before enabling DCE support, system administrators should note the following:

- The NQE Distributed Computing Environment/Distributed File Service (DCE/DFS) feature is restricted to operating within a single DCE cell. Cross realm authentication is not supported.
- Ticket forwarding is dependent upon the use of the NQE database when submitting tasks.
- Support for tasks that use forwarded tickets for DCE authentication is provided on UNICOS, UNICOS/mk, IRIX, and Solaris platforms only. Support for tasks that use a password for DCE authentication is available on all NQE 3.3 (or later) platforms. Tickets may be forwarded from any NQE 3.3 or later client to any NQE 3.3 (or later) server that supports forwarded tickets as a means of DCE authentication. (Ticket forwarding for either clients or servers is not supported for the Digital UNIX platform in the NQE 3.3 release.)

- NQE support for DCE/DFS does not include installations residing in DFS accessible file space. Installed components must reside in UNIX file space.
- If DCE support is enabled, a system cannot be configured to disable ticket forwarding.
- Kerberos is the authentication component of DCE.
- NQE supports Open Software Foundation (OSF) DCE version 1.1 only.

  Note: For UNICOS and IRIX systems, the DCE integrated login feature must be enabled for DCE credentials to be passed through NQE. For more information, see the *Cray DCE Client Services/Cray DCE DFS Server Release Overview*, publication RO–5225.

To enable DCE support on an NQE node, use the `Action` menu of the NQE configuration utility, `nqeconfig`(8)), to add the following variables to the `nqeinfo` file:

- `NQE_AUTHENTICATION`. Set the NQE authentication type. The value of this variable should be `dce` to enable DCE/DFS support.
- `NQE_DCE_REFRESH`. If DCE support is enabled, DCE renew/refresh support is automatically set to a value of 300 minutes. The value assigned to this variable must be an integer representing the number of minutes between each renew/refresh attempt. The value should be approximately half of the minimum ticket lifetime for any user. A value of `0` disables the DCE refresh feature.
- `NQE_DCE_BIN`. If the path to the `kdestroy` and `kinit` commands is not the default `/opt/dcelocal/bin` path, then add the `NQE_DCE_BIN` variable to the `nqeinfo` file, and set it to the correct full path name.

To enable DCE ticket forwarding support on an NQE client machine, only the `NQE_AUTHENTICATION` variable needs to be set in the `nqeinfo` file on the client.

14.2 Using a Password to Submit a Task

In order for a task, which utilizes DCE authentication, such as DFS, to access resources , it must first acquire credentials. This is accomplished in two ways; either by providing a password or by forwarding existing credentials. When a user submits a task to NQE, they may choose to provide a password. The password remains with the task until it is complete, and it is used by NQS to acquire and refresh credentials for the user on the execution host. When submitting a task from the command line, a user may provide a password by appending the `-P` argument to the `cqsub`(1) command. The user will be prompted for their password prior to submission of the task. When submitting a task using the NQE GUI, a user may specify a password by selecting the `Set Password` item of the `Actions` menu in the NQE `Job Submission` window.

Note: This action is separate from the user authentication performed by NQS when queuing a remotely submitted request or validating an alternate user identity. Although a password is required to obtain DCE credentials, the NQS user validation type defined within `qmgr` can remain unchanged.

Caution: If user home directories are located in DFS space, NQS user validation must be set to `password` or `no_validation if ticket forwarding will not be used`.

The integrated login feature is used on UNICOS or IRIX NQE servers for DCE authentication when a password is supplied. The DCE credentials acquired through integrated login are used for both initiating the request and returning request output files. Because integrated login is unavailable on most UNIX NQE servers, NQS makes DCE security login library calls to obtain separate sets of DCE credentials when initiating the request and returning request output files.

Failure to obtain DCE credentials results in a nonfatal error on all NQE platforms. The request will be initiated even if the attempt to obtain DCE credentials for the request owner fails. If DCE credentials are successfully obtained, the `KRB5CCNAME` environment variable is set within the request process that is initiated.

Note: Users may use the `klist`(1) command within a request script to verify that DCE credentials were obtained.

Messages are written to the NQS log file when attempts are made to obtain DCE credentials. A sample message indicating a successful attempt to obtain DCE credentials follows:

```
Request 1927.latte: Acquired DCE credentials for user <jane>,
KRB5CCNAME <FILE:/opt/dcelocal/var/security/creds/dcecred_41fffce1>.
```

A sample message indicating an unsuccessful attempt to obtain DCE credentials follows:

```
Request 1928.latte: Failed to get DCE credentials for user <jane>.
```

14.3 Using a Forwardable Ticket to Submit a Task

Another method of DCE authentication involves forwarding a user's credentials between machines to submit a task. When a user obtains their initial DCE credentials using the `kinit`(1) command, they may explicitly request that they be granted a *forwardable* ticket. This allows the user to establish a new DCE context without having to provide their password again. After the user provides their password to the `kinit` command, their credential cache file is populated with a forwardable Kerberos ticket.

A user may also request that their DCE credentials be *renewable*. Because credentials have lifetimes associated with them, they eventually expire. When a user's credentials expire, they are no longer granted access to resources that utilize DCE as an authentication mechanism. The user's credentials have two lifetimes associated with them; the ticket lifetime and the renewable lifetime. The ticket lifetime is usually less than one day. The renewable lifetime is generally much longer than the ticket lifetime, and may last for weeks or months. Before the ticket lifetime of a user's credentials expires, they may request that it be renewed. If the ticket has not exceeded its maximum renewable lifetime, the lifetime of the ticket is reset. In this way, a user's credentials may be valid throughout the maximum renewable lifetime of the ticket.

For information on DCE policy regarding ticket lifetime, see the *OSF DCE Administration Guide - Core Services*.

In order to use ticket forwarding, DCE authentication must be enabled for NQE (see Section 14.1, page 351). Prior to submitting a task to NQE, a user must acquire a forwardable and renewable Kerberos ticket granting ticket (TGT). This may be accomplished by using the `kinit`(1) command. Here is an example:

```
/opt/dcelocal/bin/kinit -f -r 100h jane
```

This command specifies that user `jane` is requesting that a forwardable ticket be granted with a renewable lifetime of one hundred hours. The user is then prompted for their DCE password. Upon entering the correct password, a credential cache file may be created if one does not already exist. Otherwise, the user's existing credential cache will be updated.

Note: Although this example refers specifically to the `kinit`(1) command supplied with DCE, a properly configured version of `kinit` from the Massachusetts Institute of Technology (MIT) Kerberos Version 5 package may also be used. In this way, it is possible for an NQE client machine without DCE installed to submit tasks that access DCE dependent resources when they are executed.

The user may now use the `cqsub`(1) command to submit a task to the NQE database. It is not necessary to pass any additional arguments to the `cqsub` command to enable ticket forwarding. Forwarding is attempted by default when NQE authentication is set to DCE. If the user's credentials could not be forwarded, an informational message will be printed as shown in the following example:

```
CQSUB: INFO: Unable to forward Kerberos ticket: <Explanation>
```

This is not a fatal error. The task will be scheduled to run, although it will not obtain DCE credentials unless an appropriate password was provided with the task.

14.4 Checkpoint and Restart Support on UNICOS Systems

On UNICOS systems, requests accessing DFS files can be checkpointed and restarted. NQS obtains new DCE credentials for the request owner just before restarting the request. The `qmgr`(8) subcommands which checkpoint and restart requests, handle this automatically.

Caution: A restarted job correctly gets the new credentials obtained from NQS, but the `KRB5CCNAME` environment variable within the restart file is not reset to the new cache file name. After the job is restarted, a `klist` within the job script will incorrectly state that there are no credentials. Consequently, DCE services are affected but not DFS, which continues to work with the new credentials.

14.5 Support for Renewing and Refreshing Tickets

In order for a user's credentials to remain valid throughout the life of a task, it may be necessary to periodically refresh the credentials . The DCE credential renew/refresh feature within NQE provides this functionality. For tasks using DCE ticket forwarding, it allows the user's credentials to remain valid throughout the maximum renewable lifetime of the user's initial ticket granting ticket. For tasks submitted with a password, the ticket may be refreshed as long as the password remains valid for the user. This feature is available on all NQE servers that support DCE.

When a task is submitted to the NQE database, the scheduler may choose not to assign the task to a lightweight server (LWS) immediately. The scheduler provided with the NQE 3.3 release periodically refreshes the credentials associated with the task if DCE ticket forwarding is being used.

> **Note:** It will be necessary to make changes to your default scheduler if you are upgrading to the NQE 3.3 release, and plan to use your old scheduler with this feature. Pay special attention to the addition of the `ticket` commands within the new scheduler.
>
> Once the task has been scheduled, the LWS will submit the job to its NQS server. The NQS request server will acquire credentials for the task and continue to renew/refresh the credentials until the task is complete.

The default renew/refresh interval is 300 minutes. This value should be approximately half of the minimum ticket lifetime for any user. For example, if user `jane` has the smallest default ticket lifetime of any user configured in the DCE cell of two hours, then the refresh interval should be configured to a value of `60`. To change the renew/refresh interval, see Section 14.1, page 351.

14.6 User Requirements

Users who submit tasks that make use of NQE DCE/DFS interoperability should be aware of the following:

- Tasks must be submitted to the NQE database. If this is not configured as the default behavior, the user must include `-d nqedb` as part of the `cqsub`(1) command arguments. This behavior must be configured in the `NQE Job Submission Options` window when using the NQE GUI to submit tasks.
- A password, forwardable credentials, or both must be available at task submission time in order for credentials to be acquired when the task is executed.
- A user's home directory may reside within DFS space.
- The request script file may reside within DFS space.
- The target directory for job output may reside within DFS space. Directing output to DFS space may be accomplished by using the standard `cqsub` arguments, which specify the destination of standard output (`-o`) and standard error (`-e`) files. The format of the DFS path should begin with `/:/` followed by the remainder of the desired location.
- The user supplied password and the DCE registry password for the user must be identical.

⚠ **Caution:** NQS limits a password to eight characters. If a user's DCE password is more than eight characters, DCE authentication will fail and credentials will not be acquired.

- A user may provide a different DCE registry user name when submitting a request by using the `User Name` field of the NQE GUI `Submit` window `General Options` menu or by using the `-u` option of the `cqsub`(1) command.
- After a request completes, NQS on UNICOS systems uses the `kdestroy`(1) command to destroy any credentials obtained by NQS on behalf of the request owner. On non-UNICOS systems, this is accomplished by calling built-in Kerberos library routines.

⚠ **Caution:** Including a `kdestroy` command within a request script file on UNICOS systems will destroy the credentials obtained by NQS and prevent NQS from returning request output files into DFS space.

14.7 Administrating Kerberos and DCE

In order for NQE tasks to utilize DCE/DFS resources, it is necessary to properly configure your DCE/DFS environment. To accomplish this, several issues must be addressed. These issues can be broken into two main groups; Kerberos specific and DCE/DFS specific.

Configuration issues relating specifically to DCE are as follows:

- User accounts (or principals) within DCE should be inspected to determine the range of ticket lifetimes. As discussed earlier, the renew/refresh interval within NQE should be approximately half of the minimum configured value in this range. Performing a ticket refresh is not a particularly expensive operation in most cases, but need not be performed at an unnecessarily high frequency. The administrator may choose to extend the ticket lifetime of all user principals to a period of at least ten hours. This allows the default NQE renew/refresh interval (300 minutes = 5 hours) to fit appropriately with the ticket lifetimes of the user principals.
- User accounts (or principals) within DCE should be inspected to determine the range of renewable lifetimes. The renewable lifetime of a principal should always exceed the maximum projected task lifetime (time from task submission to completion) for a user. If a task lives longer than the renewable lifetime of its associated credentials, the task will lose access to

DCE/DFS resources unless a valid DCE password is associated with the task. This is most important for tasks utilizing DCE ticket forwarding.

- User accounts (or principals) within DCE should be set to allow forwardable, renewable, and proxiable certificates.
- The account and password lifespan policies should be inspected. If an account expires, or if its password becomes invalid, the task may lose access to DCE controlled resources.
- Kerberos performs authentication in part based on an IP address encrypted within the ticket. The KDC (or DCE security service) checks that the ticket granting ticket (TGT) contains a IP address that matches the IP address of the interface that the request was made on. This can create difficulty for machines that have multiple IP addresses (multi-homed). The Domain Name Service (DNS) can store multiple interface IP addresses for a multi-homed host. Therefore , if you run `named`(8) on your Cray Research system you can accommodate multi-homed hosts. Kerberos 5 has the ability to store multiple IP addresses for a host within the TGT. If you do not run named you must ensure that the IP address provided by a `gethostbyname`(3C) call will return the same address on every machine that is to act as a login utility client (that is the host you wish to run `klogin` on for example). This IP address must also be the interface that the destination host (the host running `klogind`) will use to make requests to the KDC (DCE security service).

Configuration issues relating specifically to Kerberos are as follows:

- All systems that have both NQE and DCE or Kerberos installed will need to be configured for Kerberos.
- The file `/etc/krb5.conf` should contain realm specific data. In the following example, the default realm is configured to be the system `latte`, which acts as a DCE key distribution center (KDC):

```
[libdefaults]

                default_realm = latte.cray.com

                default_tkt_enctypes = des-cbc-crc

                default_tgs_enctypes = des-cbc-crc

                kdc_req_checksum_type = 2

                ccache_type = 2
```

```
[realms]

        latte.cray.com = {

               kdc = latte

               default_domain = cray.com

        }

[domain_realm]

        .cray.com = latte.cray.com
```

- The entries in `/etc/services` must be configured so that either the `kerberos` or `kerberos-sec` services operate on port `88/udp`. To preserve an existing Kerberos V4 configuration, use the following entry:

  ```
  kerberos-sec 88/udp kdc
  ```

  For Kerberos V5 only, the entry may be as follows:

  ```
  kerberos 88/udp kdc
  ```

- For more detailed information regarding Kerberos configuration, please refer to the *Kerberos V5 Installation Guide* available from MIT.

Configuring `ilb` [15]

The `ilb` utility executes a command on a machine chosen by the Network Load Balancer (NLB). To use the utility, enter the `ilb`(1) command followed by the command you want to execute. The NLB is queried to determine the machine to which you will be connected. After the login process is complete, the command is executed and I/O is connected to your terminal or pipeline. For information about using the `ilb` utility, see the `ilb`(1) man page.

Administration of `ilb` involves three main components, which affect the behavior of the `ilb` application. The components are as follows and are described in this chapter:

- `ilbrc` file (requires a specific format)
- `$HOME/.ilbrc` file (requires a specific format)
- User's environment variables

15.1 `ilbrc` File

The `ilbrc` file is located in the /*nqebase*/`etc` directory. There are two categories of information contained in the `ilbrc` file: LOGIN blocks and SYSTEM blocks.

15.1.1 LOGIN Blocks

LOGIN blocks are used to define login methods. These blocks start with a line containing LOGIN <*name*> and end with the line `END LOGIN`. There are three directives in the LOGIN block, `USER`, `INVOKE`, and `SYS`, which are described as follows:

Directive	Description
`USER`	Specifies whether a line is to be input from (`USER<`) or output to (`USER>`) the user's terminal. For example, the `USER>` directive indicates that the remainder of the line is printed to the user's terminal. The `USER<` directive must be followed by a temporary variable that indicates where the input will be stored. The `USER_QUIET<` directive is used for non-echoing input so that passwords are not visible when entered.

`INVOKE`	Specifies the application to be started to initiate a connection to a remote system.
`SYS`	Sends data to (`SYS>`) or receives data from (`SYS<`) the remote system in order to complete the connection.

The following three variables, which may be used in LOGIN blocks, are set automatically by the `ilb` utility:

Variable	Description
`$SYS`	Stores the name of the system that was selected by the NLB for the session.
`$USER`	Stores the user name of the account to log into. The `$USER` variable is set with the `-l` option, or by reading the environment variables `ILB_USER` and `LOGNAME` or `USER`.
`$PROMPT`	Specifies a user's prompt; it contains a regular expression and may be changed by setting the environment variable `ILB_PROMPT`.

15.1.2 SYSTEM Blocks

SYSTEM blocks define options specific to machine names. The SYSTEM blocks begin with the line `SYSTEM` *name*; *name* is the name of a system or is `DEFAULT` to define the fallback case. The SYSTEM block ends with `END SYSTEM`.

The `CONNECT` attribute of the SYSTEM block defines which of the LOGIN blocks should be applied to a particular system.

The `AVOID` directive is placed outside of the LOGIN and SYSTEM blocks. It should be followed by a space-separated list of machine names; the `ilb` should not use these names. This is useful when a user or administrator wants to eliminate certain systems for use with the `ilb`.

15.2 `$HOME/.ilbrc` File

Users may have their own `ilb` configuration files residing in their home directory. These configuration files are called `.ilbrc`. If this file is not present, the `ilb` uses the information contained in the system's `ilbrc` file. The format of this file is identical to that of the system's `ilbrc` file, which is defined in Section 15.1, page 361.

15.3 User Environment Variables

The user may also control the behavior of the `ilb` by defining the following specific environment variables.

Note: The user may need to set up these variables if the default values do not work correctly.

Variable	Description
`ILB_USER`	Defines the login name to use on the remote system. This variable also alters the value of `$USER` in the `ilbrc` files. The default value is whatever `$LOGNAME` or `$USER` is set to be in the user's environment.
`ILB_PROMPT`	A regular expression that identifies the user's prompt on a remote machine. The default value is `"^.*\[%$#:\] $"`, which looks for any string ending with `%`, `$`, `#`, or `:`.
`NLB_SERVER`	Defines the machine name and port number of the NLB server.

Problem Solving [16]

This chapter helps you to identify common problems you may experience while administering NQE; possible causes and resolutions are also included.

For information on DCE/DFS issues, see Chapter 14, page 351.

For information on obtaining support from customer service, see Section 16.10, page 382.

The following categories of problems are included in this chapter:

- Verifying that NQE components are running
- Verifying that daemons are running
- NQS troubleshooting
- Client troubleshooting
- NLB troubleshooting
- NQE database troubleshooting
- Network debugging
- Determining Array Services status
- Reinstalling the software
- Calling customer service

16.1 Verifying That NQE Components Are Running

To ensure that NQE components are running, complete the following steps:

1. Ensure that the daemons are running as described in Section 16.2.
2. Determine whether the NLB server daemon is up and running by using the following command:

   ```
   /nqebase/bin/nlbconfig -ping
   ```

3. Determine whether the `nqsdaemon` is up and running by using the following command:

   ```
   /nqebase/bin/qping -v
   ```

4. Determine the systems that are candidates for accepting requests based on current NLB policies by using the following command:

   ```
   /nqebase/bin/nlbpolicy -p nqs
   ```

5. Display load-balancing request information by selecting the `Load` button on the NQE GUI.
6. Display information about queues by using the `cqstatl-f` or `qstat-f` command.
7. Display NQS request information by selecting the `Status` button on the NQE GUI display or by using the `cqstatl-f` or `qstat-f` command.

If all of these steps display nonerror information, your NQE software is ready for use.

16.2 Verifying That Daemons Are Running

To ensure that the daemons are running, complete the following steps:

1. Log in as `root`.
2. Depending on which `ps` command you have in your search path, use either the `ps -e` or the `ps -aux` command to display running processes. Some systems truncate the daemon names in your `ps` display.

If one or more of the daemons shown in Table 31 are not running but are configured to run on this node, NQE is not operational. In that case, you should have received an error from the `nqeinit` script or have an error in the NQS `NQE_SPOOL/log/nqslog` log file or in the `qstart.`*$$*`.out` file (where *$$* is the PID of the process running `qstart`), or in the `nqscon.`*$$*`.out` file, which logs NQS activity until the NQS log daemon starts.

You also can check the license manager log files for any error information.

Table 31 describes the NQE server daemons.

Table 31. NQE Server Daemons

Component	Daemons
Collector	`ccol` *NLBportnumber*
FTA	`ftad` (when FTA NPPA transfers are active)

Component	Daemons
	`fta` (when any FTA transfers are active)
Load balancer	`nlbserver`
NQE database	`msqld`
LWS	`nqedb_lws.tcl`
NQS	`logdaemon`, `netdaemon`, and `nqsdaemon`

If you have checked all of the steps described in Section 16.1, page 365, and the NQE daemons still are not running, try the following steps:

- Ensure that you have added the necessary NQE port numbers to your `/etc/services` file. You may have already had these port numbers assigned to other daemons and not realized it, or you may have made a typing error when you entered the information.
- Ensure that you have configured all desired NQE components in the `nqeinfo`(5) file. Client software contains only commands; it has no daemons.
- Check the /*nqebase*/*nqeversion*/*host*/`nqeinfo` file for typing errors. This file records your choices during installation. The *host* is the TCP/IP host name of the machine on which you installed the software. Ensure that all of the information in this file is correct. If not, you should reinstall the software as described in *NQE Installation*, publication SG–5236.

To ensure that NQE is running properly, try the following steps:

- Ensure that the `nqenlb` and `nqebatch` queues exist, are enabled, and are started on each node NQS is configured for by using the NQE `cqstatl` or `qstat` command.
- Ensure that TCP/IP is running between all NQE clients and servers by using the `telnet` and `ftp` commands between the client and server systems.
- If you are using file validation, ensure that NQS validation is correct by checking that you have a valid account on all NQS servers and that the submitting user name matches the user name at the executing NQS server.

  Check that the submitting account has an entry in either a `.rhosts` or `.nqshosts` file in the home directory of the target user on each NQS server. Entries must have the format *hostname username*.

- Examine the NQS log file for any error messages. The NQS log file contains messages from the various NQS daemon processes, messages indicating successful completion of events, and messages indicating specific errors that may have occurred.

  To determine the NQS log file path name, use the following `qmgr` command:

  ```
  show parameters
  ```

  To increase the amount of information recorded in the log file, use the following `qmgr` command:

  ```
  set message_types on all
  ```

16.3 NQS Troubleshooting

This section describes some common problems users may encounter in running NQS, along with their possible causes and resolutions.

16.3.1 A Request Has Disappeared

If you cannot find a request, try the following steps to locate it.

1. The following may show the location of the request:
   - The NQE GUI `Status` window displays all requests in the complex.
   - The `cqstatl -a` and the `qstat -a` command display all requests on the server defined by the `NQS_SERVER` environment variable. The `cqstatl -a` also displays an NQE database request summary (the NQE database server is defined by the `MSQL_SERVER`, `MSQL_TCP_PORT`, and `NQE_DEST_TYPE` environment variables).
   - The `cqstatl -a -h` *remote-hostname* and `qstat -a -h` *remote-hostname* commands display all requests at a remote host.
2. The request may have been completed. The user should check for the standard output and standard error files produced by the request. Unless the user specified a different location for these files, they should be in the directory from which the request was submitted. If they are not there, check the user's home directory on the remote system.
3. The user should check electronic mail. If NQS encounters problems running a request, it may send a mail message to the issuing user.

16.3.2 A Queued Request Will Not Run

A request may be queued without being run for the following reasons.

16.3.2.1 Limits Reached

A request may be queued without being run when it reaches an NQS batch queue, even though the NQS batch queue has not reached its limits. This could happen because the limits for the queue complex, the user's group, or the NQS server global limits were reached.

To get details of the request, including its requested resource limits, use the `cqstatl` or `qstat` command. Following is an example of the command:

`cqstatl -f -h` *target_host requestids*

Compare these limits with the limits defined for the batch queue by using the same command to receive details about the queue. Following is an example of the command:

`cqstatl -f` *queues*

16.3.2.2 Pipe Queue's Destination List Points to a Disabled Batch Queue

When a pipe queue's destination list points to a disabled batch queue, the default retry time for re-queuing requests has no effect.

A pipe queue's destination list should not contain any elements that are disabled batch queues. In the event that this does occur, any jobs that are submitted to the pipe queue will remain in the pipe queue if they cannot be sent to any of the other destinations in the list. Because the disabled batch queue exists, NQS waits for the queue to become enabled or for it to be deleted before it moves the job from the pipe queue. To ensure that jobs are not sent to a particular destination in the pipe queue's list, remove the destination from the list instead of disabling the batch queue.

16.3.3 Standard Output and Standard Error Files Cannot Be Found

Unless the user specified a different location for these files, they should be in the directory that was current when the request was submitted. If they are not there, check the user's home directory or the home directory at the remote system of the user under which the request executed.

If you cannot find any output files, check for electronic mail messages sent to the user. NQS might have removed the request for a variety of reasons (for example, it requested more of a resource than was allowed for that user or the request violated your system's security). The mail message contains a description of the problem.

The user may get a mail message stating that the standard output and error files could not be written back to the user's system. The message indicates where these files were actually placed (usually in the home directory of the remote user under which the request was run). These files usually cannot be written back because the user's home directory does not contain a suitable validation file entry that authorizes NQS to write the output files. However, it also may be due to a problem with the network connection to the NQS system.

NQS tries the following methods to return output to a user:

1. NQS tries to send the output over an NQS protocol.
2. NQS tries to use FTA.
3. NQS tries to use `rcp`, which works if the user has a `.rhosts` entry for the server on the user's submitting system (NQE client).
4. NQS sends the user mail stating that it could not deliver the output to its first destination. It places the output in the user's home directory (or in the home directory of the target user) on the server that executed the request.
5. If for some reason NQS cannot write to the user's home directory on the server that executed the request (for example, if the permissions on the user's home directory are `r-xr-xr-x`), it writes the output to the `$NQE_NQS_SPOOL/private/root/failed` directory (only `root` can access this directory) .

16.3.4 Syntax Errors

If the standard error file contains an excessive number of syntax errors, the user may be using the wrong shell. Usually, it is the shell flow control commands (such as `if`, `while`, `for`, and `foreach`) that cause errors when the wrong shell is used. Ensure that the user is using the correct shell strategy.

16.3.5 File Not Found Errors in Standard Error File

If the user receives an error message such as `file not found` or `file does not exist` in the standard error file, the batch request may have tried to access a file that could not be found.

NQS assumes that any files the user is trying to access are in the home directory of the user at the execution host. To indicate that a file is in another location, the user should either specify the full path name or use the `cd` command to move to that directory before trying to access the file within the batch request script.

16.3.6 Queues Shut Off by NQS

NQS can shut off a queue or several queues (with no intervention from a manager or operator). The reason for the action might be that the `nqsdaemon` session reached a process limit or that it cannot allocate a resource. To determine the reason, you should examine the NQS log file. After determining the reason and correcting the problem, the queue(s) can be restarted manually.

16.3.7 Unexpected Changes in Request States

Requests may unexpectedly switch states from `queued` to `wait`. A checkpointed UNICOS, UNICOS/mk, or IRIX request may be waiting for a specific process or a job ID to become available. When it becomes available, the request will remain in `queued` state until it executes.

16.3.8 Problems Issuing `qmgr` Commands

Most `qmgr` commands can be executed only by an NQS manager or operator. An operator can issue only a subset of the commands that are available to the manager. If you try to use a command that you are not allowed to use, the following message is displayed:

```
NQS manager[TCML_INSUFFPRV ]: Insufficient privilege at
local host.
```

16.3.9 NQS or `qmgr` Fails with a `no local machine id` Message

If NQS or `qmgr` fails with a `no local machine id` message, and the site is using DNS (Domain Name Service) to resolve host names, follow these procedures:

1. Use the `hostname` command to find the true host name that NQS or `qmgr` would be using. For example:

   ```
   # hostname sn9031
   ```

2. Use the `nslookup` utility to find out what the DNS server thinks your host name really is (that is, the name/alias(s) it goes by); for example:

   ```
   # nslookup
   Default Server: VGER.PRIUS.JNJ.CO
   Address: 122.147.94.7

   sn9031
   Server: VGER.PRIUS.JNJ.COM
   Address: 122.147.94.7

   Name:    Cray1.PRIUS.JNJ.com
   Address: 122.147.92.39
   Aliases: SN9031.PRIUS.JNJ.com

   exit
   ```

3. The DNS on the system above says `sn9031` has the name `Cray1.PRIUS.JNJ.com` and has an alias of `SN9031.PRIUS.JNJ.com`. Use `qmgr` commands to add the machine ID with those names. Assuming that the machine ID is to be `1`, the command would be as follows:

   ```
   # qmgr
   ADd MId 1 sn9031 Cray1.PRIUS.JNJ.com SN9031.PRIUS.JNJ.com
   exit
   ```

   It is important to note that it **does** make a difference in the host names/aliases if they are upper- or lowercase; the names/aliases must be entered exactly as DNS shows them. The result of the preceding example is shown by using the `qmgr sho mi` command as follows:

   ```
   # qmgr sho mi
   MID      PRINCIPAL NAME   AGENTS     ALIASES
   -------- --------------   ------     -------
   1        sn9031           nqs        Cray1.PRIUS.JNJ.com
                                        SN9031.PRIUS.JNJ.com
   ```

qmgr can now find the machine ID for the host name of the local machine.

16.3.10 NQS Seems to Be Running Slowly

If NQS appears to be running slowly, you may need to edit the /etc/hosts file.

Because NQS supports networks of machines, it uses the gethostbyname system call to get the host information for a request. The implementation makes calls to this routine while processing all requests, including requests that are submitted on a local machine for execution on the local machine.

The gethostbyname system call looks up the host name by doing a sequential search of the /etc/hosts.bin binary file (or the text version /etc/hosts if the binary file does not exist).

The bigger the host file, and the further down in the file the local machine name appears, the longer it will take for the system call to complete, and the longer the nqsdaemon will take to process requests.

To solve this problem, edit the /etc/hosts file so that the local machine appears at the top of the file, then use the mkbinhost file to make a new copy of the binary /etc/host.bin file.

16.3.11 Configuring the Scope of User Status Display

NQE allows an administrator to expand the default scope of the information displayed to non-NQS managers using the qstat(1) or cqstatl(1) commands. The default behavior for these commands is to display information only to non-NQS managers regarding jobs that they have submitted themselves. To let users display the status of all jobs residing at that NQS node when they execute the qstat(1) or cqstatl(1) command, use the NQE configuration utility (nqeconfig(8) command) and set the nqeinfo file variable NQE_NQS_QSTAT_SHOWALL to 1, which modifies the information that is displayed to the users of these commands.

16.4 Client Troubleshooting

This section describes several common problems users may encounter in running NQE clients, along with their possible causes and resolutions.

Note: If a user does not specify a user name when submitting a request, the user name must be the same on both the client host and the NQS server or account authorization will fail. If the `NQE_NQS_NQCACCT` environment variable is set to `ORIGIN`, the user must have the same account name on both the client and the server.

16.4.1 Client Users Cannot Access the NQS Server

If the `NQS_SERVER` environment variable is not set, the client tries to connect to the NQS server on the local host. If the user cannot connect to a local NQS server, the following message is displayed:

```
QUESR: ERROR: Failed to connect to NQS_SERVER at host[port]
obtaining UNIX environment failed
NETWORK: ERROR: Connect: Connection refused
```

Ensure that the `NQS_SERVER` environment variable is set. This must be set in each user's environment to contain the host name of the machine running the NQS server.

It also may be true that the NQS server is not running or that a networking problem exists between the workstation and the NQS server (see Section 16.7, page 379).

16.4.2 `Non-secure network port` Messages

NQE clients are installed with set UID (`suid`), are owned by `root`, and use secure ports. This means that clients running at a specific NQE release level are rejected if they try to connect to an NQE server running at a different NQE release level. The following messages are displayed:

```
QUESRV: ERROR: Failed to retrieve status information
QUESRV: ERROR: Non-secure network port verification error at
transaction peer
QUESRV: ERROR: Received response NONSECPORT (01462) for
transaction NPK_QSTAT
```

If you receive these messages, you should upgrade your client software.

16.4.3 A Request Is Accepted by NQS but Not Run

A request that remains in an NQS queue for a long time without executing usually indicates that NQS is not running or that the machine system itself is not running.

However, you should check the status of the queue in which the request is waiting to see whether the queue is currently stopped. Enter the `qmgr` command `show queue` and look under the column `STS`, which displays `on` when the queue is started. A stopped queue cannot process requests that are waiting in it.

You also can check any request attributes specified on the `cqsub` or the `qsub` command line. Ensure that they match the attributes of the batch queue.

16.4.4 `No account authorization at transaction peer` Messages

The default validation type in NQS is file validation. File validation requires a `.rhosts` (or `.nqshosts`) file in the home directory of your account on all of the NQS server systems on which your request may run. This applies even if your target NQS server is your local machine.

If a client command returns the message `No account authorization at transaction peer`, or if a `cqsub` command results in the message being returned in a mail message, one of the following may be true:

- No `.nqshosts` or `.rhosts` file exists for this user. This may be the case if the `$HOME` environment variable points to a location that does not have these files.
- The proper *hostname username* pair is not contained in either the `.nqshosts` or `.rhosts` file. NQS checks the `.rhosts` file only when the `.nqshosts` file does not exist. An example `.rhosts` file entry for the user `jane` to submit requests to host `snow` is as follows:

  ```
  snow jane
  ```
- The file mode of the `.nqshosts` or `.rhosts` file may need to be changed to `644` (`rw-r--r--`). This may not be true for all platforms that NQE supports.

16.4.5 Commands Do Not Execute

Before you can use the NQE commands, you must add the NQE `bin` and `etc` directories to your search path. Before you can use the man pages (which tell you about the NQE commands and command options), you must add the NQE `man` directories to your search path. For a description of how to set these variables, see *Introducing NQE*, publication IN–2153.

16.4.6 `ctoken generation failure` Messages

If you submit requests by using the `cqsub` or `qsub` command and receive `ctoken generation failure` messages in the NQS log, it indicates that NQS tried to use FTA to deliver output files and did not find a user-supplied password, a `.netrc` file, or an NPPA S-key. The log also contains a message indicating that the FTA transfer failed.

When NQS returns output, it tries to use FTA. FTA checks first for a user-supplied password on the `cqsub` or `qsub` command. (To determine the syntax used to specify the FTA password, see the description of the `-e` option on the `cqsub`(1) or `qsub` man page.)

If no password was supplied, FTA tries to send the output to the host and user (with an associated password) by using the user's `.netrc` file on the NQS execution host. If no `.netrc` file exists, FTA tries to send the file by using NPPA. If you have not defined S-keys as described in Section 13.2.8.1, page 331, you will receive `ctoken generation failure` messages.

NQS then tries to return the output by using `rcp`.

You can ensure that you do not receive the messages by using any of the following methods:

- Ensure that users supply a password on the `cqsub` or `qsub` command line.
- Ensure that users have a `.netrc` file on the NQS execution host that contains an entry for the destination host and user name and an associated password.
- Configure NPPA as described in section Chapter 13, page 317.
- Change the configuration of the FTA default `inet` domain so that the `nopassword` flag is not set. Section 13.2.8.1, page 331, describes how to make changes to the FTA configuration.

16.4.7 NQE `Load` Display Fails

If your NQE `Load` display fails, ensure that your `NLB_SERVER` environment variable is set to a host that is running an NLB server. You do not have to specify the host port number unless the NLB server is configured with a NLB port number that is different from the default (604) provided during installation.

16.5 NLB Troubleshooting

This section describes some common problems you may encounter in running NLB, along with their possible causes and resolutions.

16.5.1 NLB Commands Report `cannot determine server location`

If NLB commands report that they cannot determine the server location, no information is available to locate the server. NQE may not have been configured on the host. To connect to the server, define the `NLB_SERVER` environment variable or use the `-s` option on the command line.

16.5.2 Collector Is Running, but No Data Appears in Server

If a collector is running on a host, but no data appears in the NLB server for that host, check the following:

- The collector may be pointing to the wrong location. Check that the machine on which the collector is running is configured to point at the correct host name and TCP/IP port to connect properly to the server.
- An ACL may be preventing the collector from entering data into the server. If you have an ACL for NLB or NQE GUI status data, it must include a record to allow the user ID running the collector to insert and delete data. This can be either an explicit entry for a user ID on the collector host, or an entry for that ID on all hosts using `"*"` for the host name.

16.5.3 `nlbconfig -pol` Reports Errors

If you have modified the `policies` file and the `nlbconfig -pol` command reports an error, the `policies` file that you downloaded to the server may contain a syntax error. If this is true, the `nlbconfig -pol` command fails; however, you do not receive an explanation of the error. To see the error, stop the server (with `nlbconfig -kill`) and start it again (with `nlbserver`). Any errors in parsing the policy file will be reported to `stderr`. The server does not treat an error in the policy file as fatal, but load balancing will not work until you have a correct `policies` file.

16.5.4 Collectors on Solaris Systems Report Incorrect Data

Collectors on Solaris systems may report incorrect data under the following conditions:

- If a collector appears to be working, but reports zero memory size for a machine, you are running the collector from an account that does not have permission to read from `/dev/kmem`, which is used when finding memory size. You must run the collector from an account with permission to read from `/dev/kmem`.
- If a collector appears to be running, but reports random data, check that `sar` is working correctly. The collector tries to execute `/usr/bin/sar -crquw 1` to obtain data. If the command does not work, you must fix the system problem.
- If swap space data seems to be invalid, check that `/usr/sbin/swap -s` works. If the command does not work, you must fix the system problem.

16.5.5 Policies Cause Unexpected Results

If you are using a policy and getting unexpected results, probably the wrong policy is being used. If a policy name is not defined, a default policy is used. The policy name might be defined incorrectly if you misspell the policy name either in the NQS queue definition or in the `nlbpolicy` command.

16.5.6 Modifying ACL for Type `C_OBJ` Reports `no privilege for requested operation`

If you try to modify the ACL for object type `C_OBJ`, you will receive the message `No privilege for requested operation`. You cannot modify the ACL for `C_OBJ` because it is fixed. It consists of the master ACL plus `WORLD read` permission. These permissions are necessary because, if a user cannot read the `C_OBJ` object type, most commands will fail.

16.6 NQE Database Troubleshooting

This section describes some common problems you may encounter in running the NQE database, along with their possible causes and resolutions.

16.6.1 NQE Database Authorization Failures

A client trying to connect to the NQE database without the proper validation will result in an error such as the following:

```
NETWORK: ERROR: NQE Database connection failure:
        Connection disallowed.
latte$
```

If this occurs, verify that the user has an entry in the `nqedbusers` file. For additional information about the `nqedbusers` file, see Chapter 9, page 231.

16.6.2 NQE Database Connection Failures

If a user is trying to submit a request to the NQE database and the NQE database server is not up, the following message will be sent:

```
NETWORK: ERROR: NQE Database connection failure:
        Cannot connect to MSQL server on latte.
latte$
```

For additional information on checking network connectivity, see Section 16.7.

16.7 Network Debugging

The following sections discuss how to diagnose network problems.

16.7.1 Checking Connectivity

The first step in network debugging is to ensure that the machine that is being accessed can be reached by the local host.

To ensure that you can access a host through TCP/IP, use the `ping` command. This command sends a data packet to the destination host and does not rely on the presence of any specific service at the destination. For example, if the node name is `rhubarb`, enter the following command:

```
ping rhubarb
```

Entering the previous command can result in several replies:

- `ping:  unknown host rhubarb`

  If you receive this reply, the host name is unknown on your machine; obtain its Internet address and ensure that this address is in your `/etc/hosts` file or the network information service (NIS).

- `sendto:  Network is unreachable`

  If you receive this reply, your machine does not know how to send a packet to the destination machine. Consult with your network administrator and explain what you are trying to do, because the solution may require action on his or her part. For example, a routing table may need updating, a gateway machine may be down, or the network to which you are trying to connect cannot be reached.

- `no answer from rhubarb`

  If you receive this reply, your machine knows how to get to the destination node, but did not get a reply. This usually means that either the destination machine is shut down, or the destination machine is running but does not know how to send a packet to your machine. In this case, check the routing on the destination machine.

- `rhubarb is alive.`

  As far as `ping` is concerned, everything is working.

16.7.2 Checking Transport Service Connections

If the host-to-host network connection is running, the next step is to examine the individual connections on your machine and the machine with which you are trying to communicate.

The main tool for examining TCP/IP connections is the `netstat` command, which examines kernel data structures and prints the current state of all connections to and from the host on which it is run. For more information on how to use this command, see the operating system manuals for your machine.

When you specify the `netstat`(1) command without any parameters, it reports each open connection on a machine; `netstat -a` also reports the connections offered by servers.

The most important information to check is the protocol type (`Proto`), your local address (`Local Address`), and the status of the connection (`state`).

16.7.3 `Connection failed` Messages

If you receive `connection failed` messages, ensure that your `NQS_SERVER` environment variable is set to a host that is running an NQS server. You do not have to specify the host port number unless the NQS server is configured with an NQS port number that is different from the default (607) provided during installation.

If a transient network error condition exists that allows the connection to be retried, you will receive messages such as the following:

```
Retrying connection to NQS_SERVER on host ice (127.111.21.90) . . .
Retrying connection to NQS_SERVER on host ice (127.111.21.90) . . .
QUESRV: ERROR: Failed to connect to NQS_SERVER at ice [port 607]
NETWORK: ERROR: NQS network daemon not responding
```

These messages indicate that the command received a `connection refused` error from the `connect` () system call or that an address it is trying to use is temporarily in use. This could be true, for example, because the server is just coming up or is not listening, or the network is busy. NQS retries its connection to the server for 30 seconds.

16.8 User's Job Project Displays As `0`

If a user's job project is displayed as `0` instead of the project specified in `/etc/project`, the Array Services daemon may not be running. For information about Array Services administration, see the Silicon Graphics publication *Getting Started with Array Systems*.

16.9 Reinstalling the Software

If you want to reinstall the software, complete the following steps:

1. Stop all NQE daemons by using the `nqestop`(8) command. If this does not stop all of them, or does not work, you can use the `kill -9` command to stop them.
2. Delete the desired version of NQE using the `nqemaint`(8) utility.
3. Reinstall the software as described in *NQE Installation*, publication SG–5236.

16.10 Calling Customer Service

To obtain support for NQE, contact your local call center or local service representative. The `cs_support`(7) man page provides NQE product support information.

You also can send a facsimile (fax) to +1-404-631-2226. To obtain service, you must have your NQE product customer number.

Before you call your local call center or local service representative, have the following information ready to help with your problem:

- The exact error messages and unexpected behavior you are seeing.
- The `ps` displays of all daemons that are running on the systems with which you are having problems.
- A list of NQE binary files, their paths, their protections, and their sizes.
- The `qstat -f` or `cqstatl -f` output from all queues in the batch complex.
- Hardware platforms, model numbers, system software levels, and additional software patches applied to your NQE platforms.
- Accounting information, if any, about the exit status of any daemons or commands aborting (such as copies of the corresponding NQE log file and any error messages sent to the user).

Man Page List [A]

The `man` command provides online help on all NQE commands. To view a man page online, type `man` *commandname.*

You must ensure that your `MANPATH` is set properly; it must include the path name of the NQE man pages. The default name is `/`*nqebase*`/man`.

A.1 User Man Pages

The following user-level online man pages are provided with your NQE software:

User-level man page	Description
`cevent`(1)	Posts, reads, and deletes job-dependency event information.
`cqdel`(1)	Deletes or signals to a specified batch request.
`cqstatl`(1)	Provides a line-mode display of requests and queues on a specified host.
`cqsub`(1)	Submits a batch request to NQE.
`ftua`(1)	Transfers a file interactively (this command is issued on an NQE server only).
`ilb`(1)	Executes a load-balanced interactive command.
`nqe`(1)	Provides a graphical user interface (GUI) to NQE functionality.
`qalter`(1)	Alters the attributes of one or more NQS requests (this command is issued on an NQE server only).
`qchkpnt`(1)	Checkpoints an NQS request on a UNICOS, UNICOS/mk, or IRIX system (this command is issued on an NQE server only).
`qdel`(1)	Deletes or signals NQS requests (this command is issued on an NQE server only).
`qlimit`(1)	Displays NQS batch limits for the local host (this command is issued on an NQE server only).

`qmsg`(1)	Writes messages to `stderr`, `stdout`, or the job log file of an NQS batch request (this command is issued on an NQE server only).
`qping`(1)	Determines whether the local NQS daemon is running and responding to requests (this command is issued on an NQE server only).
`qstat`(1)	Displays the status of NQS queues, requests, and queue complexes (this command is issued on an NQE server only).
`qsub`(1)	Submits a batch request to NQS (this command is issued on an NQE server only).
`rft`(1)	Transfers a file in a batch request (this command is issued on an NQE server only).

A.2 Application Interfaces Man Pages

The following online man pages are provided with your NQE software to help those who must develop applications that others might use to perform work using NQE, such as writing a scheduler or modifying the NQE GUI:

Application interface man page	Description
`nqeapi`(3)	Describes general NQE functions for formatted lists.
`nqe_get_policy_list`(3)	Queries the NLB to retrieve a formatted list of hosts that match a specified policy.
`nqe_get_request_ids`(3)	Returns a list of all NQS request IDs known.
`nqe_get_request_info`(3)	Returns a list of all information known about a specific NQS request ID from a specified NLB server.
`nqe_request_delete`(3)	Deletes a specific NQS request.

`nqe_request_submit`(3)	Submits a batch request to NQS.

For more detailed information, you also may want to read a reference book about Tcl or Tk, such as the *Tcl and the Tk Toolkit* by John K. Ousterhout (Addison Wesley publisher) or *Practical Programming in Tcl and Tk* by Brent B. Welch (Prentice Hall publisher).

A.3 File Formats Man Pages for UNICOS Systems

The following online man pages describe the content of NQE configuration files and are provided with your NQE software:

Configuration file man pages	Description
`fta.conf`(5)	UNICOS systems that run only the NQE subset (NQS and FTA components) use the `fta.conf`(5) command, which is the file transfer agent (FTA) configuration file for these systems. The `fta.conf`(5) online man page documents the `fta.conf` file, which contains the list of file transfer services (FTSs) that FTA operates.
`nqeinfo`(5)	The `nqeinfo` file is the NQE configuration file. The `nqeinfo`(5) online man page documents all NQE configuration variables.

A.4 Product Support Man Page

The `cs_support`(7) online man page is provided with your NQE software; it includes product support information.

A.5 Administrator-level Man Pages

The following administrator-level online man pages are provided with your NQE software:

Administrator-level man page	Description
`ccollect`(8)	Daemon that provides NLB data to NQE.
`compactdb`(8)	Compacts the NQE database.

`csuspend`(8)	Suspends NQE batch activity.
`fta`(8)	Performs file transfer management.
`ftad`(8)	Invokes the FTA file transfer protocol server.
`nlbconfig`(8)	Configures and maintains the NLB server.
`nlbpolicy`(8)	Queries the NLB server.
`nlbserver`(8)	Starts an NLB server.
`nqeconfig`(8)	Configures the NQE `nqeinfo` variables.
`nqedbmgr`(8)	NQE database administration.
`nqeinit`(8)	Starts NQE.
`nqemaint`(8)	Performs NQE version maintenance.
`nqestop`(8)	Stops NQE.
`nqsdaemon`(8)	Starts the NQS main daemon.
`nqsfts`(8)	File transfer service for NQS.
`qconfigck`(8)	Verifies the NQS configuration of customized variables.
`qmgr`(8)	Enters the NQS queue manager subsystem.
`qstart`(8)	Starts NQS.
`qstop`(8)	Stops NQS.
`skey`(8)	FTA daemon network peer-to-peer authorization management.

NQE Sample Configurations [B]

This appendix describes sample NQE configurations for NQE administrators and discusses the following topics:

- NQE servers and clients
- NQE cluster configuration
- NQS queue structure
- NQE database and scheduler structure

B.1 NQE Servers and Clients

NQE consists of a cluster of servers and clients. The following components can be configured to run on nodes in an NQE cluster:

- The Network Load Balancer (NLB) server which receives and stores information from the NLB collectors in the NLB database which it manages.
- The NQE database server which serves connections from clients, the scheduler, the monitor and lightweight server (LWS) components in the cluster to add, modify, or remove data from the NQE database.
- The NQE scheduler which analyses data in the NQE database, making scheduling decisions.
- The NQE database monitor which monitors the state of the database and which NQE database components are connected.
- NQS which schedules and initiates batch requests on the local server.
- FTA which adds reliability through transfer retry and recovery and through network peer-to-peer authorization (NPPA). FTA transfers may be initiated on any server and those transfers may retrieve files from or copy files to any NQE server or client.
- NLB collectors which collect and communicate system load information to the NLB server so that the NLB can provide NQS with a list of servers, in order of preference, to run a request and provide status information upon request.
- The LWS which obtains request information from the NQE database, verifies validation, submits the copy of a request to NQS, and obtains exit status of

completed requests from NQS. The LWS may override NQS scheduling. For information about LWS attributes, see Chapter 9, page 231, and Chapter 10, page 261.

- NQE *clients* contain software so users may submit, monitor, and control requests by using either the NQE graphical user interface (GUI) or the command-line interface. From clients, users also may monitor request status, delete or signal requests, monitor machine load, and receive request output.

For more information on components which can be configured to run on nodes in an NQE cluster, see Section 1.2, page 2.

NQE uses the FLEXlm *license manager* to license the product. Before you can run NQE, a license manager must be installed and running. The license manager can run on any server or on another (nonserver) host.

The NLB collector is run on all NQS server nodes to collect the following:

- NQS request status data
- Host load information for destination selection

By default, the collector is configured to report the system batch queue as `nqebatch@` *hostname* and to send information to the NLB server.

The initial configuration of NQE starts the following components: NQS, NLB, and Collector

B.2 NQE Cluster Configuration

This section describes NQE clusters and gives you information about selecting servers. You can expand the examples to suit the needs of your network.

B.2.1 Simple Cluster

A *simple cluster* has one node to execute batch requests and has some clients, as shown in Figure 26.

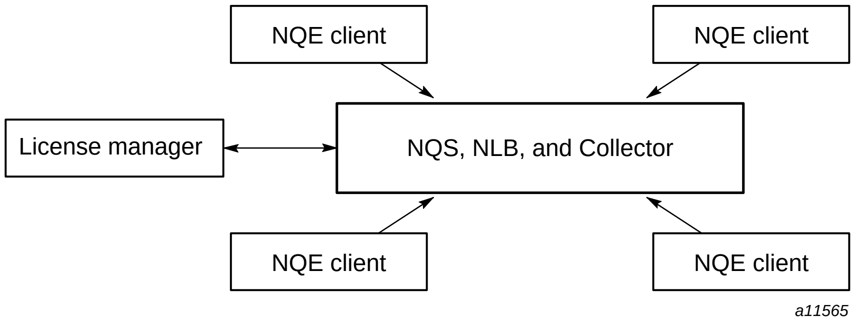


Figure 26. Simple NQE Cluster

B.2.2 NQE Load-balanced Cluster

An *NQE load-balanced cluster* has one NLB server, multiple NQS servers, and multiple clients, as shown in Figure 27. Clients submit work to an NQE load-balancing queue on any node. In this network, requests are sent to any node in an NQE cluster. To set up a quorum, the license manager is placed on each node.

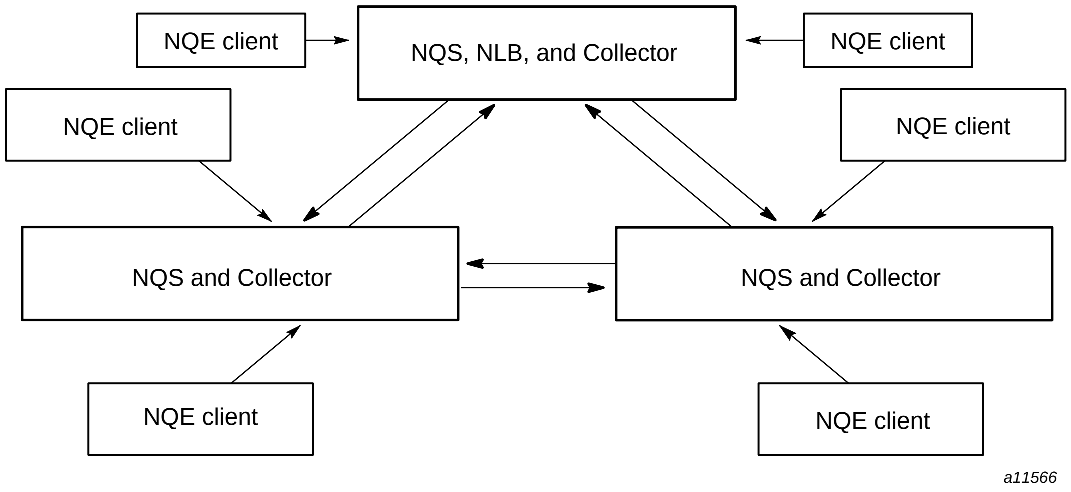


Figure 27. NQE Load-balanced Cluster

B.2.3 NQE Scheduler Cluster

An *NQE scheduler cluster* has one NQE database server, multiple NQS servers, one NLB server, and multiple clients, and is in this sense the same as the load-balanced cluster shown in Figure 27. The difference between the two clusters lies in how work is scheduled, as described in section Section B.4, page 393.

B.2.4 Server Considerations

The following types of NQE servers can exist in the NQE cluster:

- At least one node with high availability for the NQE cluster. The NLB load-balancing server and NQE database server run on this node.
- Typically, the NQE cluster includes additional NQS servers that have sufficient CPU cycles to be used for executing batch requests. However, in a simple network, this might be the same node as the NLB server.
- Typically, many workstations are designated as NQE clients, which do not provide batch execution cycles to the NQE cluster.

B.2.5 Load-balancing Policies

NQE comes configured with a default NQS load-balancing policy that assumes all of your NLB servers are of similar speed and that you want requests sent to the system that has the lowest CPU load. You can alter this policy or create others to suit your needs. For example, you could create policies that weigh and normalize differences in system CPU, or you could create policies that define sets of systems with important characteristics (some systems might run a given application, some systems might be for compiling, and so on). Chapter 8, page 193, describes how to define and implement load-balancing policies.

B.2.6 NQE Scheduler and NQE Database Options

In certain environments, it is not sufficient to choose a destination for a request by using NLB policies. For example, you may want to wait to send requests to a machine until its idle CPU is greater than 90%, or you may want to allow a user to run only one request at a time across the entire NQE cluster, regardless of which machine.

In such cases, you can use the NQE scheduler and NQE database. For a summary of how the NQE scheduler and NQE database are structured, see section Section B.4, page 393. For more information, see Chapter 9, page 231, and Chapter 10, page 261.

B.3 NQS Queue Configuration

NQS supports two types of queues: *batch* queues, which initiate requests; and *pipe* queues, which route work to appropriate destinations for execution or further routing. A *destination selection* queue is a pipe queue that uses load-balancing to route requests. These pipe queues are associated with policies.

When requests enter a destination-selection queue, NQS queries the NLB for a list of destinations. The list is ordered based on the policy. When multiple destinations fit the policy, NQS tries the first destination. If for some reason the first destination cannot accept the request, NQS tries the second, and so on.

When you submit a request, NQS places it in the default queue unless you specify otherwise. NQE is shipped with the following queues configured:

- The destination-selection queue `nqenlb`, which is the default for request submission.
- The batch queue `nqebatch`, which handles local batch request submissions and accepts requests from other destination-selection queues in the NQE cluster.

This configuration uses the NLB to send requests to the most appropriate system, based on an NLB policy. The default NLB policy, called `nqs`, sends batch requests to the system that has the lowest CPU load.

If users submit requests to the `nqebatch` queue, the request runs on the local NQS server.

Part of the configuration process is to define these queues and to interconnect the different instances of NQS to form the NQE cluster.

The `NQS_SERVER` environment variable points to the NQS server that handles NQE request submission. If you want your request run on another NQS server explicitly, you can change the `NQS_SERVER` environment variable to point to another host running NQS. You must have access to an account on this host and must have set up NQS validation as described in *Introducing NQE*, publication IN–2153.

Figure 28 shows how the default NQE configuration would look in a network that had one NLB server and two NQS servers. The `nqebatch` queue on each NQS system is configured by default to allow no more than six requests to run at one time and to allow no more than two requests from a user to run at one time on each NQS system. If the queue is full (running the predefined number of requests), requests wait in the `nqebatch` queue until an opening occurs, or until the queue limits are changed.

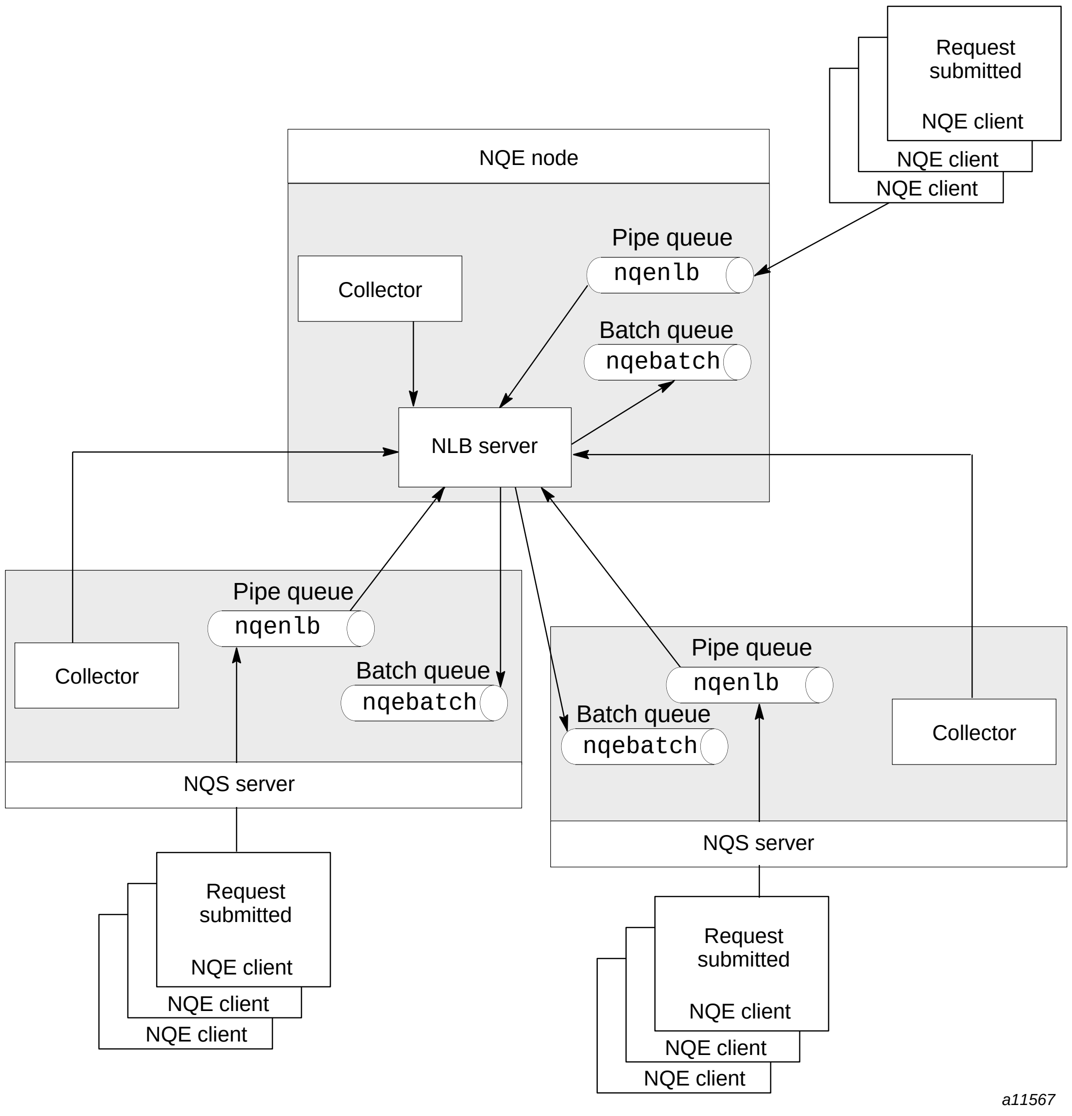


Figure 28. Default NQE Queue Configuration

B.4 NQE Database and Scheduler Structure

The NQE database and NQE scheduler let clients submit work to the NQE central storage database on the NQE database server. The scheduler analyzes the requests in the database, and selects when and on which NQS server the request will run.

The NQE scheduler and database components are shown in Figure 29.

The database server system task serves connections from clients (either locally or in the network) to access data in the database.

The monitor system task monitors the state of the databases and which system tasks are connected.

The scheduler system task analyzes data in the database and makes scheduling decisions.

The lightweight server (LWS) system task obtains request information from the database, checks authorization files locally, submits request to NQS, and obtains their exit status.

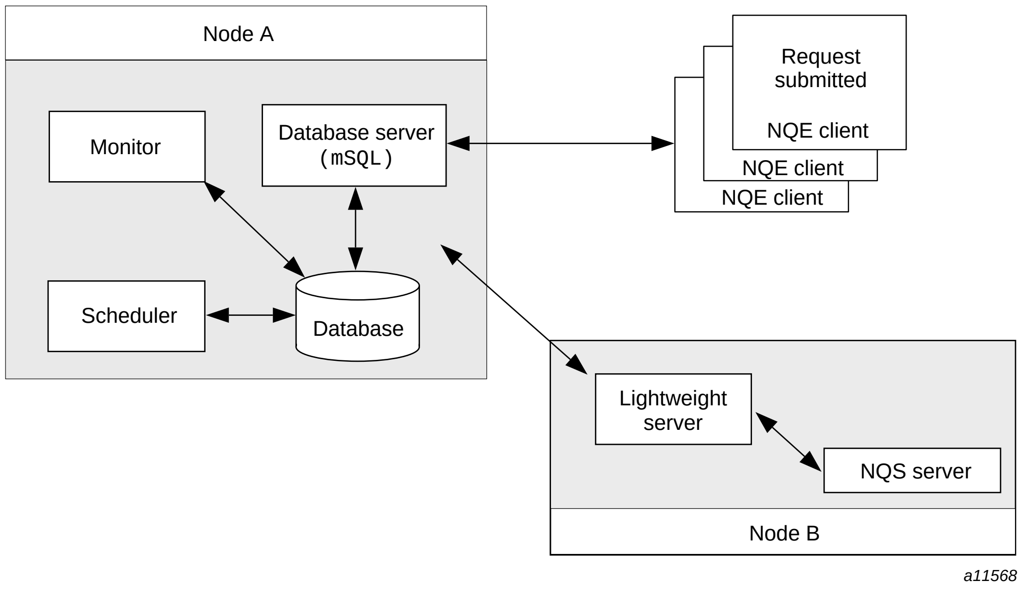


Figure 29. NQE Database and NQE Scheduler Components

config.h/nqeinfo File Variables [C]

This appendix applies only to UNICOS 9.0.*x* and UNICOS 9.1 systems.

NQS `config.h` file customization variables have been moved into the `nqeinfo` file. This appendix lists each new `nqeinfo` file variable name and its equivalent `config.h` file variable name, along with the default setting and a description of the variable's function. All `nqeinfo` file configuration variables are documented on the `nqeinfo`(5) man page.

Note: Use the NQE configuration utility (`nqeconfig`(8)) to add any of the `NQE_NQS_`*xxxx* variables listed in this appendix to the `nqeinfo` file.

`NQE_NQS_ABORT_WAIT IC_ABORT_WAIT`

Default is: 60

Specifies the default time, in seconds, to wait after sending a `SIGTERM` signal to all processes for each request running in a queue that has received a `PKT_ABOQUE` packet. At the end of this time period, any remaining queue processes receive a `SIGKILL` signal. This variable is the default grace period for the `qmgr` commands `abort queue`, `hold request`, and `preempt request`. This variable is an integer from 0 to 600.

`NQE_NQS_CHKPNT_DELAY IC_CHKPNT_DELAY`

Default is: 0

Defines the number of seconds to wait before a `chkpnt`(2) system call is retried during periodic checkpointing. This variable can be 0 or any positive integer.

`NQE_NQS_CHKPNT_DELAY_KILL IC_CHKPNT_DELAY_KILL`

Default is: 60

Defines the number of seconds to wait before a `chkpnt`(2) system call is retried during a hold or shutdown. This variable can be 0 or any positive integer.

`NQE_NQS_CHKPNT_TRIES IC_CHKPNT_TRIES`

Default is: 1

Defines the total number of `chkpnt`(2) system calls that will be retried. This variable is an integer; a value that is less than 1 is treated as a 1.

`NQE_NQS_CHKPNT_TRIES_KILL IC_CHKPNT_TRIES_KILL`

Default is: 3

Defines the total number of `chkpnt`(2) system calls that will be retried. This variable is an integer; a value that is less than 1 is treated as a 1.

`NQE_NQS_DEF_NETRETTIM IC_DEF_NETRETTIM`

Default is: 31

Sets the maximum number of seconds during which a network function may continue to fail, because of a temporary condition, before being marked as completely failed. This variable is an integer from 0 to 600.

`NQE_NQS_DEF_USRRETTIM IC_DEF_USRRETTIM`

Default is: 15

Defines the maximum time, in seconds, that is allowed for the `qdel`(1) or `qstat`(1) command to try to connect to a remote machine. This variable is an integer from 0 to 600.

`NQE_NQS_DLIM_PPCORECOEFF IC_DLIM_PPCORECOEFF`

Default is: 256

Defines the default per-process core size limit coefficient. The complete limit is expressed as `NQE_NQS_DLIM_PPCORECOEFF * NQE_NQS_DLIM_PPCOREUNITS`. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 0 to 2,147,483,647 ((2^{31})-1).

`NQE_NQS_DLIM_PPCOREUNITS IC_DLIM_PPCOREUNITS`

Default is: `QLM_MWORDS`

Defines the default per-process core size limit units. The complete limit is expressed as `NQE_NQS_DLIM_PPCORECOEFF * NQE_NQS_DLIM_PPCOREUNITS`. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is one of the following values:

QLM_BYTES	Allocate in units of bytes
QLM_WORDS	Allocate in units of words
QLM_KBYTES	Allocate in units of Kbytes
QLM_KWORDS	Allocate in units of Kwords
QLM_MBYTES	Allocate in units of Mbytes
QLM_MWORDS	Allocate in units of Mwords
QLM_GBYTES	Allocate in units of Gbytes
QLM_GWORDS	Allocate in units of Gwords

`NQE_NQS_DLIM_PPCPUMS IC_DLIM_PPCPUMS`

Default is: 0

Defines the time, in milliseconds, of the default per-process CPU limit. The complete limit is expressed as `NQE_NQS_DLIM_PPCPUSECS + NQE_NQS_DLIM_PPCPUMS`. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 0 to 999.

`NQE_NQS_DLIM_PPCPUSECS IC_DLIM_PPCPUSECS`

Default is: 720000

Defines the time, in seconds, of the default per-process CPU limit. The complete limit is expressed as `NQE_NQS_DLIM_PPCPUSECS + NQE_NQS_DLIM_PPCPUMS`. To change this variable for an individual queue, use the `qmgr set per_process cpu_limit` command; when no specific value is set during the creation of a queue, this variable is used as the default value creation of a queue. This variable is an integer from 1 to 2,147,483,647.

`NQE_NQS_DLIM_PPDATACOEFF IC_DLIM_PPDATACOEFF`

Default is: 256

Defines the default per-process data size limit coefficient. The complete limit is expressed as `NQE_NQS_DLIM_PPDATACOEFF`

* NQE_NQS_DLIM_PPDATAUNITS. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 0 to 2,147,483,647.

NQE_NQS_DLIM_PPDATAUNITS IC_DLIM_PPDATAUNITS

Default is: QLM_MWORDS

Defines the default per-process data size limit units. The complete limit is expressed as NQE_NQS_DLIM_PPDATACOEFF * NQE_NQS_DLIM_PPDATAUNITS. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is one of the following values:

QLM_BYTES	Allocate in units of bytes
QLM_WORDS	Allocate in units of words
QLM_KBYTES	Allocate in units of Kbytes
QLM_KWORDS	Allocate in units of Kwords
QLM_MBYTES	Allocate in units of Mbytes
QLM_MWORDS	Allocate in units of Mwords
QLM_GBYTES	Allocate in units of Gbytes
QLM_GWORDS	Allocate in units of Gwords

NQE_NQS_DLIM_PPMEMCOEFF IC_DLIM_PPMEMCOEFF

Default is: 256

Defines the default per-process memory size limit coefficient. The complete limit is expressed as NQE_NQS_DLIM_PPMEMCOEFF * NQE_NQS_DLIM_PPMEMUNITS. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 0 to 2,147,483,647.

NQE_NQS_DLIM_PPMEMUNITS IC_DLIM_PPMEMUNITS

Default is: QLM_MWORDS

Defines the default per-process memory size limit units. The complete limit is expressed as NQE_NQS_DLIM_PPMEMCOEFF * NQE_NQS_DLIM_PPMEMUNITS. When no specific value is set during the creation of a queue, this value is used as the default value. This variable is one of the following values:

`QLM_BYTES`	Allocate in units of bytes
`QLM_WORDS`	Allocate in units of words
`QLM_KBYTES`	Allocate in units of Kbytes
`QLM_KWORDS`	Allocate in units of Kwords
`QLM_MBYTES`	Allocate in units of Mbytes
`QLM_MWORDS`	Allocate in units of Mwords
`QLM_GBYTES`	Allocate in units of Gbytes
`QLM_GWORDS`	Allocate in units of Gwords

`NQE_NQS_DLIM_PPMPP_SECS IC_DLIM_PPMPP_SECS`

Default is: 10

Defines the default per-process massively parallel processing Cray MPP systems time limit. To change this variable for an individual queue, use the `qmgr set per_process mpp_time_limit` command. When no specific value is set during the creation of a queue, this variable is used as the default value. If you change this value, it will not alter all queue limits, but it will be used only when another queue is created. The variable is an integer from 0 to 2,147,483,647.

`NQE_NQS_DLIM_PPNICE IC_DLIM_PPNICE`

Default is: 0

Defines the default nice value assigned to requests in a queue. For more information, see the `qmgr set nice_increment` command. This variable is an integer from -20 to 19. You should specify only a positive value because a negative value is used for the initial queue nice value during the queue creation for all queues. To change this variable for an individual queue, use the `qmgr set nice_increment` command. When no specific value is set during the creation of a queue, this variable is used as the default value.

`NQE_NQS_DLIM_PPPFILECOEFF IC_DLIM_PPPFILECOEFF`

Default is: 100

Defines the default per-process permanent file size limit coefficient. To change this value for an individual queue, use the `qmgr set per_process permfile_limit` command. The complete limit is expressed as NQE_NQS_DLIM_PPPFILECOEFF * NQE_NQS_DLIM_PPPFILEUNITS. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 0 to 2,147,483,647.

NQE_NQS_DLIM_PPPFILEUNITS IC_DLIM_PPPFILEUNITS

Default is: QLM_MBYTES

Defines the default per-process permanent file size limit units. To change this value for an individual queue, use the qmgr set per_process permfile_limit command. The complete limit is expressed as NQE_NQS_DLIM_PPPFILECOEFF * NQE_NQS_DLIM_PPPFILEUNITS. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is one of the following values:

QLM_BYTES	Allocate in units of bytes
QLM_WORDS	Allocate in units of words
QLM_KBYTES	Allocate in units of Kbytes
QLM_KWORDS	Allocate in units of Kwords
QLM_MBYTES	Allocate in units of Mbytes
QLM_MWORDS	Allocate in units of Mwords
QLM_GBYTES	Allocate in units of Gbytes
QLM_GWORDS	Allocate in units of Gwords

NQE_NQS_DLIM_PPQFILECOEFF IC_DLIM_PPQFILECOEFF

Default is: 0

Defines the default per-process quick-file size limit coefficient. The complete limit is expressed as NQE_NQS_DLIM_PPQFILECOEFF * NQE_NQS_DLIM_PPQFILEUNITS. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 0 to 2,147,483,647.

NQE_NQS_DLIM_PPQFILEUNITS IC_DLIM_PPQFILEUNITS

Default is: QLM_MBYTES

Defines the default per-process permanent quick-file size limit units. The complete limit is expressed as NQE_NQS_DLIM_PPQFILECOEFF * NQE_NQS_DLIM_PPQFILEUNITS. When no specific value is set during the creation of a queue, this variable is used as the default value during the creation of a queue. This variable is one of the following values:

QLM_BYTES	Allocate in units of bytes
QLM_WORDS	Allocate in units of words
QLM_KBYTES	Allocate in units of Kbytes
QLM_KWORDS	Allocate in units of Kwords
QLM_MBYTES	Allocate in units of Mbytes
QLM_MWORDS	Allocate in units of Mwords
QLM_GBYTES	Allocate in units of Gbytes
QLM_GWORDS	Allocate in units of Gwords

NQE_NQS_DLIM_PPSTACKCOEFF IC_DLIM_PPSTACKCOEFF

Default is: 256

Defines the default per-process stack size limit coefficient. The complete limit is expressed as NQE_NQS_DLIM_PPSTACKCOEFF * NQE_NQS_DLIM_PPSTACKUNITS. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 0 to 2,147,483,647.

NQE_NQS_DLIM_PPSTACKUNITS IC_DLIM_PPSTACKUNITS

Default is: QLM_MWORDS

Defines the default per-process stack size limit units. The complete limit is expressed as NQE_NQS_DLIM_PPSTACKCOEFF * NQE_NQS_DLIM_PPSTACKUNITS. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is one of the following values:

QLM_BYTES	Allocate in units of bytes
QLM_WORDS	Allocate in units of words
QLM_KBYTES	Allocate in units of Kbytes

`QLM_KWORDS`	Allocate in units of Kwords
`QLM_MBYTES`	Allocate in units of Mbytes
`QLM_MWORDS`	Allocate in units of Mwords
`QLM_GBYTES`	Allocate in units of Gbytes
`QLM_GWORDS`	Allocate in units of Gwords

`NQE_NQS_DLIM_PPTFILECOEFF IC_DLIM_PPTFILECOEFF`

Default is: 0

Defines the default per-process temporary file size limit coefficient. The complete limit is expressed as `NQE_NQS_DLIM_PPTFILECOEFF * NQE_NQS_DLIM_PPTFILEUNITS`. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 0 to 2,147,483,647.

`NQE_NQS_DLIM_PPTFILEUNITS IC_DLIM_PPTFILEUNITS`

Default is: `QLM_MBYTES`

Defines the default per-process temporary file size limit units. The complete limit is expressed as `NQE_NQS_DLIM_PPTFILECOEFF * NQE_NQS_DLIM_PPTFILEUNITS`. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is one of the following values:

`QLM_BYTES`	Allocate in units of bytes
`QLM_WORDS`	Allocate in units of words
`QLM_KBYTES`	Allocate in units of Kbytes
`QLM_KWORDS`	Allocate in units of Kwords
`QLM_MBYTES`	Allocate in units of Mbytes
`QLM_MWORDS`	Allocate in units of Mwords
`QLM_GBYTES`	Allocate in units of Gbytes
`QLM_GWORDS`	Allocate in units of Gwords

`NQE_NQS_DLIM_PPWORKCOEFF IC_DLIM_PPWORKCOEFF`

Default is: 256

Defines the default per-process work size limit coefficient. The complete limit is expressed as `NQE_NQS_DLIM_PPWORKCOEFF * NQE_NQS_DLIM_PPWORKUNITS`. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 0 to 2,147,483,647.

`NQE_NQS_DLIM_PPWORKUNITS IC_DLIM_PPWORKUNITS`

Default is: `QLM_MWORDS`

Defines the default per-process work size limit units. The complete limit is expressed as `NQE_NQS_DLIM_PPWORKCOEFF * NQE_NQS_DLIM_PPWORKUNITS`. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is one of the following values:

`QLM_BYTES`	Allocate in units of bytes
`QLM_WORDS`	Allocate in units of words
`QLM_KBYTES`	Allocate in units of Kbytes
`QLM_KWORDS`	Allocate in units of Kwords
`QLM_MBYTES`	Allocate in units of Mbytes
`QLM_MWORDS`	Allocate in units of Mwords
`QLM_GBYTES`	Allocate in units of Gbytes
`QLM_GWORDS`	Allocate in units of Gwords

`NQE_NQS_DLIM_PRCPUMS IC_DLIM_PRCPUMS`

Default is: 0

Defines the milliseconds portion of the default per-request CPU limit. To change this variable for an individual queue, use the `qmgr set per_request cpu_limit` command. The complete limit is expressed as `NQE_NQS_DLIM_PRCPUMS + NQE_NQS_DLIM_PRCPUSECS`. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 0 to 999.

`NQE_NQS_DLIM_PRCPUSECS IC_DLIM_PRCPUSECS`

Default is: 720000

Defines the seconds portion of the default per-request CPU limit. To change this variable for an individual queue, use the qmgr set per_request cpu_limit command. The complete limit is expressed as NQE_NQS_DLIM_PRCPUMS + NQE_NQS_DLIM_PRCPUSECS. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 1 to 2,147,483,647.

NQE_NQS_DLIM_PRDRIVES IC_DLIM_PRDRIVES

Default is: 0

Defines the per-request tape drive limit. To change this variable for an individual queue, use the qmgr set per_request tape_limit command. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 0 to 255.

NQE_NQS_DLIM_PRMEMCOEFF IC_DLIM_PRMEMCOEFF

Default is: 256

Defines the default per-request memory size limit coefficient. To change this variable for an individual queue, use the qmgr set per_request memory_limit command. The complete limit is expressed as NQE_NQS_DLIM_PRMEMCOEFF * NQE_NQS_DLIM_PRMEMUNITS. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 0 to 2,147,483,647.

NQE_NQS_DLIM_PRMEMUNITS IC_DLIM_PRMEMUNITS

Default is: QLM_MWORDS

Defines the default per-process memory size limit units. The complete limit is expressed as NQE_NQS_DLIM_PRMEMCOEFF * NQE_NQS_DLIM_PRMEMUNITS. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is one of the following values:

QLM_BYTES	Allocate in units of bytes
QLM_WORDS	Allocate in units of words
QLM_KBYTES	Allocate in units of Kbytes
QLM_KWORDS	Allocate in units of Kwords
QLM_MBYTES	Allocate in units of Mbytes
QLM_MWORDS	Allocate in units of Mwords

QLM_GBYTES	Allocate in units of Gbytes
QLM_GWORDS	Allocate in units of Gwords

NQE_NQS_DLIM_PRMPP_PES IC_DLIM_PRMPP_PES

Default is: 0

Defines the default per-request Cray MPP systems processing element (PE) limit. To change this variable for an individual queue, use the qmgr set per_request mpp_pe_limit command. When no specific value is set during the creation of a queue, this variable is used as the default value. If you change this value, it will not alter all queue limits, but it will be used only when another queue is created. The variable is an integer from 0 to NQE_NQS_MAX_MPPPES.

NQE_NQS_DLIM_PRMPP_SECS IC_DLIM_PRMPP_SECS

Default is: 10

Defines the default per-request Cray MPP systems time limit. To change this variable for an individual queue, use the qmgr set per_request mpp_time_limit command. When no specific value is set during the creation of a queue, this variable is used as the default value. If you change this value, it will not alter all queue limits, but it will be used only when another queue is created. The variable is an integer from 0 to 2,147,483,647.

NQE_NQS_DLIM_PRPFILECOEFF IC_DLIM_PRPFILECOEFF

Default is: 0

Defines the default per-request permanent file size limit coefficient. To change this variable for an individual queue, use the qmgr set per_request permfile_limit command. The complete limit is expressed as NQE_NQS_DLIM_PRPFILECOEFF * NQE_NQS_DLIM_PRPFILEUNITS. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 0 to 2,147,483,647.

NQE_NQS_DLIM_PRPFILEUNITS IC_DLIM_PRPFILEUNITS

Default is: QLM_BYTES

Defines the default per-request permanent file size limit units. To change this variable for an individual queue, use the qmgr set per_request permfile_limit command. The complete limit is expressed as `NQE_NQS_DLIM_PRPFILECOEFF * NQE_NQS_DLIM_PRPFILEUNITS`. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is one of the following values:

`QLM_BYTES`	Allocate in units of bytes
`QLM_WORDS`	Allocate in units of words
`QLM_KBYTES`	Allocate in units of Kbytes
`QLM_KWORDS`	Allocate in units of Kwords
`QLM_MBYTES`	Allocate in units of Mbytes
`QLM_MWORDS`	Allocate in units of Mwords
`QLM_GBYTES`	Allocate in units of Gbytes
`QLM_GWORDS`	Allocate in units of Gwords

`NQE_NQS_DLIM_PRQFILECOEFF IC_DLIM_PRQFILECOEFF`

Default is: 0

Defines the default per-request quick-file-size limit coefficient. To change this variable for an individual queue, use the `qmgr set per_request quickfile_limit` command. The complete limit is expressed as `NQE_NQS_DLIM_PRQFILECOEFF * NQE_NQS_DLIM_PRQFILEUNITS`. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 0 to 2,147,483,647.

`NQE_NQS_DLIM_PRQFILEUNITS IC_DLIM_PRQFILEUNITS`

Default is: `QLM_BYTES`

Defines the default per-request quick-file size limit units. To change this variable for an individual queue, use the `qmgr set per_request quickfile_limit` command. The complete limit is expressed as `NQE_NQS_DLIM_PRQFILECOEFF * NQE_NQS_DLIM_PRQFILEUNITS`. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is one of the following values:

`QLM_BYTES`	Allocate in units of bytes
`QLM_WORDS`	Allocate in units of words
`QLM_KBYTES`	Allocate in units of Kbytes

QLM_KWORDS	Allocate in units of Kwords
QLM_MBYTES	Allocate in units of Mbytes
QLM_MWORDS	Allocate in units of Mwords
QLM_GBYTES	Allocate in units of Gbytes
QLM_GWORDS	Allocate in units of Gwords

NQE_NQS_DLIM_PRSHM_COEFF IC_DLIM_PRSHM_COEFF

Default is: 0

Defines the default per-request shared memory size limit coefficient. The complete limit is expressed as NQE_NQS_DLIM_PRSHM_COEFF * NQE_NQS_DLIM_PRSHM_UNITS. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 0 to NQE_NQS_MAX_SHM_COEFF.

NQE_NQS_DLIM_PRSHM_SEGS IC_DLIM_PRSHM_SEGS

Default is: 0

Defines the default per-request shared memory segment limit. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 0 to NQE_NQS_MAX_SHM_SEGS.

NQE_NQS_DLIM_PRSHM_UNITS IC_DLIM_PRSHM_UNITS

Default is: QLM_MWORDS

Defines the default per-request shared memory size limit units. The complete limit is expressed as NQE_NQS_DLIM_PRSHM_COEFF * NQE_NQS_DLIM_PRSHM_UNITS. When no specific value is set during the creation of a queue, this variable is used as the default value.

NQE_NQS_DLIM_PRTFILECOEFF IC_DLIM_PRTFILECOEFF

Default is: 0

Defines the default per-request temporary file size limit coefficient. To change this variable for an individual queue, use the `qmgr set per_request tempfile` command. The complete limit is expressed as NQE_NQS_DLIM_PRTFILECOEFF * NQE_NQS_DLIM_PRTFILEUNITS. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is an integer from 0 to 2,147,483,647.

NQE_NQS_DLIM_PRTFILEUNITS IC_DLIM_PRTFILEUNITS

Default is: QLM_BYTES

Defines the default per-request temporary file size limit units. To change this variable for an individual queue, use the qmgr set per_request tempfile command. The complete limit is expressed as NQE_NQS_DLIM_PRTFILECOEFF * NQE_NQS_DLIM_PRTFILEUNITS. When no specific value is set during the creation of a queue, this variable is used as the default value. This variable is one of the following values:

QLM_BYTES	Allocate in units of bytes
QLM_WORDS	Allocate in units of words
QLM_KBYTES	Allocate in units of Kbytes
QLM_KWORDS	Allocate in units of Kwords
QLM_MBYTES	Allocate in units of Mbytes
QLM_MWORDS	Allocate in units of Mwords
QLM_GBYTES	Allocate in units of Gbytes
QLM_GWORDS	Allocate in units of Gwords

NQE_NQS_ER_WAIT_TIME IC_ER_WAIT_TIME

Default is: 60

Defines the time interval (in seconds) to wait when checking whether external resources are available again after being unavailable. If the time interval elapses and the external resources are still unavailable, the timer is reset to wait again. Otherwise, the scheduler is called to process any queued requests that require the resources. This variable is an integer from 1 to 300.

NQE_NQS_EVENT_LOG_MODE IC_EVENT_LOG_MODE

Default is: 1

Defines the configuration switch that controls whether the request-related event messages will be logged automatically to the job log area in the tail of the request's control file. If this variable is set to 0, no messages are generated automatically. If this variable is set to 1, the messages are logged automatically to the request's control file as events happen to the request.

`NQE_NQS_FAIR_RATIO IC_FAIR_RATIO`

Default is: 0

Defines the configuration flag that controls which fair-share algorithm will be used. This variable can be set to 0 for the standard fair-share algorithm, or 1 for the share/usage ratio algorithm. When the kernel fair-share process scheduling is enabled, you should use the standard algorithms. You should use the alternative share/usage ratio algorithm only when the kernel is collecting share data.

`NQE_NQS_IPC_TIMEOUT IC_IPC_TIMEOUT`

Default is: 90

Defines the minor time-out interval that is used for the interprocess communication time-out for NQS daemon-to-daemon requests. When this timer expires, the NQS daemon is queried to determine whether it is still running and available to process requests. This timer is set again and a wait occurs. This continues until `NQE_NQS_IPC_WAIT_TIME` expires, which causes the request to fail. This is important for long-running tasks, such as suspend or checkpoint, so that they do not time out. This variable is an integer from 1 to 86,400.

`NQE_NQS_IPC_TIMEOUT_I IC_IPC_TIMEOUT_I`

Default is: 15

Defines the minor time-out interval that is used for the interprocess communication time-out for NQS user commands-to-daemon requests. When this timer expires, the NQS daemon is queried to determine whether it is still running and available to process requests. If the daemon is still available, the following message is displayed on the user's screen: Waiting on local NQS daemon to complete transaction. This timer is set again and a wait occurs. This continues until `NQE_NQS_IPC_WAIT_TIME` expires, which causes the request to fail. This is important for long-running tasks, such as suspend or checkpoint so that they do not time out. This variable is an integer from 1 to 86,400.

`NQE_NQS_IPC_WAIT_TIME IC_IPC_WAIT_TIME`

Default is: 5400

Defines the major time-out interval that is used for the interprocess communication time-out for NQS commands and daemons. This time-out value is used with the `NQE_NQS_IPC_TIMEOUT` and `NQE_NQS_IPC_TIMEOUT_I` variables. If this time-out value expires, the current task ends with a time-out response. This variable is an integer from 1 to 86,400.

`NQE_NQS_MAC_COMMAND IC_MAC_COMMAND`

Default is: 0

Enables mandatory access control (MAC) checking for job status and deletion. If enabled, `qstat` lets users display only job information that their MAC label dominates. If enabled, `qdel` lets users delete jobs that are equal to their MAC label (user-owned jobs only). To disable MAC checking, set this variable to 0; to enable MAC checking, set it to 1.

`NQE_NQS_MAC_DIRECTORY IC_MAC_DIRECTORY`

Default is: 0

Enables multilevel directory (MLD) support for NQS spool directories that hold files with different MAC labels. To specify wildcard directories, set this variable to 0; to specify MLDs, set it to 1.

`NQE_NQS_MAX_ENVIRONMENT IC_MAX_ENVIRONMENT`

Default is: 5120

Defines the maximum amount of environment space, in bytes, for a request. If this value is reduced, requests that are queued before the change may fail. This variable is an integer from 5120 to 50,000.

`NQE_NQS_MAX_ENVIRONVARS IC_MAX_ENVIRONVARS`

Default is: 400

Defines the maximum number of environment variables for a request server. If this value is reduced, requests that are queued before the change may fail. This variable is an integer from 100 to 25,000.

`NQE_NQS_MAX_EXTREQUESTS IC_MAX_EXTREQUESTS`

Default is: 250

Defines the maximum number of requests from other remote machines that can be queued at a given time on this machine.

The `NQE_NQS_MAX_EXTREQUESTS` value must always be less than or equal to `NQE_NQS_MAX_TRANSACTS`. Any change in `NQE_NQS_MAX_TRANSACTS` must be considered carefully. If any NQS requests are on the local host, you can only increase the value of `NQE_NQS_MAX_TRANSACTS`. If one or more requests are on the local host, you cannot decrease the value of `NQE_NQS_MAX_TRANSACTS`. Any change to `NQE_NQS_MAX_EXTREQUESTS` also must be carefully considered. Any increase in this value is always safe. A danger, however, in decreasing this value exists. You can reduce `NQE_NQS_MAX_EXTREQUESTS` only if you know that the number of requests currently queued on the local system is less than or equal to the reduced value of `NQE_NQS_MAX_EXTREQUESTS`. If this condition is not true, NQS cannot requeue all of the requests on rebooting. If a value smaller than the current number of transactions in the NQS database is specified, the value will be increased to the current read count.

This variable is an integer from 1 to 65,536. If you change this variable when queued requests are in NQS, the queued requests will be deleted when the NQS databases are re-created.

`NQE_NQS_MAX_GBLBATLIMIT IC_MAX_GBLBATLIMIT`

Default is: 20

Defines the global maximum number of batch requests that are running simultaneously. To change this variable for an individual queue, use the `qmgr set global batch_limit command`. This variable is an integer from 1 to 65,536.

`NQE_NQS_MAX_GBLNETLIMIT IC_MAX_GBLNETLIMIT`

Default is: 20

Defines the global maximum number of network queue servers that are running simultaneously. This variable is an integer from 1 to 65,536.

`NQE_NQS_MAX_GBLPIPLIMIT IC_MAX_GBLPIPLIMIT`

Default is: 20

Defines the global maximum number of pipe queue servers that are running simultaneously. This value acts only as an initial value. You can configure other values through the `qmgr` program. To change this variable for an individual queue, use the `qmgr set global pipe_limit` command. This variable is an integer from 1 to 65,536.

`NQE_NQS_MAX_GBLTAPLIMIT IC_MAX_GBLTAPLIMIT`

Default is: 100

Defines the global maximum number of tape drives that can be allocated simultaneously. To change this variable for an individual queue, use the `qmgr set global tape_limit` command. This variable is an integer from 1 to 65,536.

`NQE_NQS_MAX_MPPPES IC_MAX_MPPPES`

Default is: 4096 on Cray MPP systems running UNICOS/mk or Origin 64-bit systems, 64 on all other platforms

Defines the maximum number of MPP processing elements (PEs) on a Cray MPP system or the maximum number of CPUs on an Origin 64-bit system that are accessible at the local host. This variable is an integer from 0 to 65,533.

`NQE_NQS_MAX_NETRETTIM IC_MAX_NETRETTIM`

Default is: 300

Sets the maximum number of seconds (retry time) during which a network function can continue to fail, because of a temporary condition, before being marked as completely failed. To change this variable for an individual queue, use the `qmgr` set network retry_time command. This variable is an integer from 1 to 86,400.

NQE_NQS_MAX_QPRIORITY IC_MAX_QPRIORITY

Default is: 63

Defines the maximum priority that can be assigned to an NQS queue. This variable is an integer from 0 to 63.

NQE_NQS_MAX_SHM_COEFF IC_MAX_SHM_COEFF

Default is: MAX_NQS_LIM_SHM_L

Defines the maximum finite value of the shared memory limit coefficient. This variable is an integer from 0 to MAX_NQS_LIM_SHM_L.

NQE_NQS_MAX_SHM_SEGS IC_MAX_SHM_SEGS

Default is: MAX_NQS_LIM_SHM_S

Defines the maximum finite value of the number of shared memory segments that can be created. This variable is an integer from 0 to MAX_NQS_LIM_SHM_S.

NQE_NQS_MAX_TRANSACTS IC_MAX_TRANSACTS

Default is: 500

Defines the maximum number of concurrent transactions. Each request on the system has a transaction descriptor. Therefore, the value of this parameter sets a limit on the maximum number of requests that can be queued on the local host.

The NQE_NQS_MAX_EXTREQUESTS value must always be less than or equal to NQE_NQS_MAX_TRANSACTS. You must consider carefully any change in NQE_NQS_MAX_TRANSACTS. If any NQS requests are on the local host, you can only increase the value of NQE_NQS_MAX_TRANSACTS. If one or more requests are on the local host, you cannot decrease the value of NQE_NQS_MAX_TRANSACTS. You also must consider carefully any decrease to NQE_NQS_MAX_EXTREQUESTS. A danger, however, in decreasing this value exists. You can reduce NQE_NQS_MAX_EXTREQUESTS only if you know that the number of requests currently queued on the local system is less than or equal to the reduced value of

`NQE_NQS_MAX_EXTREQUESTS`. If this condition is not true, NQS will not requeue all of the requests on rebooting.

This variable is an integer from 1 to 65,536. If you change this variable when queued requests are in NQS, the queued requests will be deleted when the NQS databases are re-created.

`NQE_NQS_MPP_CHECK IC_MPP_CHECK`

Default is: 0 for a UNIX system, 1 for CRAY T3D systems running UNICOS

Enables checking for CRAY T3D availability. If enabled, NQS checks for the availability of the CRAY T3D device before it initiates a request that requires the device. If it is disabled, no checking is performed. To disable this variable, set it to 0; to enable it, set it to 1. This variable is intended for use only in a test environment; you should not change it in a production environment.

`NQE_NQS_MSG_CAT_MODE IC_MSG_CAT_MODE`

Default is: 1

Defines the default mode in which NQS will use the UNICOS message catalog for its user and log messages. This variable can be set to either 0 to reduce the memory that is required at the expense of additional I/O, or set to 1 to reduce I/O at the expense of additional memory.

`NQE_NQS_NQS_NICE IC_NQS_NICE`

Default is: -5

Defines the nice value of the NQS daemons. This variable is an integer from -19 to 20, however, NQS should always run at a more desirable nice value than all user processes and at a less desirable nice value than the kernel.

`NQE_NQS_PRIV_FIFO IC_PRIV_FIFO`

Default is: 0

Enables use of privilege through privilege assignment lists (PALs) to read and write on the NQS protocol FIFO named pipe and the `NQS LOGFIFO` pipe. If this variable is not enabled, these two pipes are created with a wildcard label. To use a wildcard label, set this variable to 0; to use the privilege mechanism, set it to 1.

`NQE_NQS_QSTAT_SHOWALL IC_QSTAT_SHOWALL`

Default is: 0

Defines the behavior of a `qstat -a` command, and lets the site configure it to display either all jobs or just the user’s jobs. You can set this variable to either 0 to display only the user’s own jobs, or 1 to display all jobs.

`NQE_NQS_SHELLINVOCATION IC_SHELL_INVOCATION`

Default is: 2

Defines the invocation method used to initiate batch jobs. This variable can be set either to 1 for the one-shell invocation method or to 2 for the two-shell invocation method.

`NQE_NQS_SHELLPATH IC_SHELL_PATH`

Default is: Null string

Defines the shell used to initiate a batch job. The default value, a null string, indicates that the user’s UDB shell should be used. This variable can be any valid shell path name.

`NQE_NQS_SHUTDOWN_WAIT IC_SHUTDOWN_WAIT`

Default is: 60

Defines the default time, in seconds, to wait after sending a `SIGTERM` signal to all processes for each request that runs in any queue after it receives a `qmgr shutdown` command. At the end of this time period, all remaining processes of any remaining NQS requests receive a `SIGKILL` signal. This variable is an integer from 1 to 86,400.

`NQE_NQS_TAPED_CHECK IC_TAPED_CHECK`

Default is: 0 for a UNIX system, 1 for a UNICOS or UNICOS/mk system

Enables checking for tape daemon availability. If enabled, NQS checks for the availability of the tape daemon before it initiates a request that requires a tape device. If it is disabled, no checking is performed. To disable checking, set this variable to

0; to enable checking, set it to 1. This variable is intended for use only in a test environment; you should not change it in a production environment.

`NQE_NQS_TCP_SERVICE IC_TCP_SERVICE`

Default is: `nqs`

Defines the TCP/IP port or service name from the `/etc/services` file that will be used for NQS. This variable must be a valid TCP/IP port or service name that is defined in the `/etc/services` file.

`NQE_NQS_TMPDIR IC_TMPDIR`

Default is: `JTMPDIR`

Defines the name of the temporary directory for NQS batch requests. A subdirectory is created in `NQE_NQS_TMPDIR` for each NQS job. The name of the NQS temporary directory must be a valid directory; NQS does not create this directory. On UNICOS and UNICOS/mk systems, valid values are `JTMPDIR`, which translates into the use of `/tmp`, or `JTMPDIRNQS`, which translates into the use of `/nqstmp`. On UNIX systems, this variable can be set to any valid directory path.

If a UNICOS NQS administrator wishes to use a directory other than `/tmp` or `/nqstmp`, such as `/scratch`, a solution is to create a link between `/nqstmp` and `/scratch` (that is, `ln -s /scratch /nqstmp`) and then set the `NQE_NQS_TMPDIR` value to `JTMPDIRNQS`.

The environment variable `TMPDIR` is created and sent to the user shell.

`NQE_NQS_TMPMODE IC_TMPMODE`

Default is: `JTMPMODE`

Defines the permissions (mode) for the temporary directory that is created for each NQS job that is initiated. This mode setting is for each user’s temporary directory. The name of that temporary directory will be generated by `NQE_NQS_TMPDIR`. This does not set the mode for `TMPDIR` itself. The default value for `NQE_NQS_TMPMODE` comes from `/usr/include/tmpdir.h`; it uses the value of the variable `JTMPMODE` as a default. The value of `JTMPMODE` is 0700. This variable is any valid mode bit mask.

`NQE_NQS_TREE_TIMER IC_TREE_TIMER`

Default is: 15

Controls the frequency, in minutes, that the NQS share tree for fair-share scheduling is regenerated. After NQS startup, the share tree is regenerated periodically with any changes related to user IDs, account IDs, shares, or resource groups. If fair-share scheduling is not used for NQS job scheduling, or there are few changes to the UDB values, you can specify a long frequency. If there are many changes or you want to keep current values, you can specify a short frequency. This variable is an integer from 5 to 1440.

`NQE_NQS_URM_RETRY IC_URM_RETRY`

Default is: 120

For UNICOS sytems, defines the default Unified Resource Manager (URM) connection retry time, in seconds. If the connection between NQS and URM is broken, NQS retries the connection every `IC_URM_RETRY` seconds. This variable is an integer from 60 to 2,147,483,647 ((2^{31})-1).

`NQE_NQS_URM_SERVICE IC_URM_SERVICE`

Default is: `urm`

For UNICOS systems, defines the URM port or service name from `/etc/services` that NQS will use to interface to URM. This variable string must be a valid TCP/IP port or service name that is defined in `/etc/services`.

Index