NQE Administration

SG–2150 3.0

Portions of this product may still be in development. The existence of those portions still in development is not a commitment of actual release or support by Cray Research, Inc. Cray Research, Inc. assumes no liability for any damages resulting from attempts to use any functionality or documentation not officially released and supported. If it is released, the final form and the time of official release and start of support is at the discretion of Cray Research, Inc.

The UNICOS operating system is derived from UNIX® System V. The UNICOS operating system is also based in part on the Fourth Berkeley Software Distribution (BSD) under license from The Regents of the University of California.

New Features

NQE Administration SG–2150 3.0

The NQE 3.0 release has the following new features that affect administrators:

- The `nqeinfo` file is now supported on NQE clients. `NQS_SERVER` now may be set in the `nqeinfo` file, which eliminates the need for client users to set the `NQS_SERVER` environment variable if they want to use the `nqeinfo` file default setting.
- The `cqstat`(1) command was removed and replaced with the new NQE GUI `Status` window capability.
- Support was added for the NQE database, which is an mSQL database, and NQE scheduler . Section 8 of this manual and the new `nqedbmgr`(8) command are provided to enable you to administer the NQE database.
- The new NQE scheduler is written in Tcl. This new feature allows a site to create its own scheduler as opposed to using NLB destination selection and NQS. Section 9 of this manual and the new `nqedbmgr`(8) command are provided to help you to modify or write an NQE scheduler.
- Support was added for clusterwide rerun of requests submitted to the NQE database.
- Support was added for using the Distributed Computing Environment (DCE) processing capability when submitting a job to NQE; this is a licensed feature of NQE. The `NQE_AUTHENTICATION` variable and the `NQE_DCE_BIN` variable were added to support this feature. Section 13 of this manual documents this new feature for system administrators.
- FTA has been enhanced to support byte-level recovery. You can also use the new NQE GUI `Status` window to verify the status of your FTA file transfers. The following three FTA variables were added: `NQE_FTA_NLBCOLLECT`, which activates logging of transfer information to the NLB, `NQE_FTA_SMARTE_RESTART`, which enables FTA to do byte-level restarts of partially complete transfers, and `NQE_FTA_STAT_INTERVAL`, which determines the time interval (in seconds) between progress updates sent to the NLB.

- A new customized object feature allows you to store arbitrary information in the NLB. This information is periodically sent to the NLB, along with the data that is usually stored and updated by NQE. After the customized data is in the NLB, it can be used in policies or displays just as any other data. NQE collects and stores the customized data, but you define, generate, and update it. The new `ccollect -C` *filename* option and the new `NQE_CUSTOM_FILE_LIST` variable support this new feature. Section 7 of this manual and the `ccollect`(8) man page document this new feature.

- The new `ilb` utility executes a command on a machine chosen by the NLB. To use the utility, enter the `ilb` command followed by the command you want to execute. The NLB is queried to log you on to the appropriate machine. After the login process is complete, the command is executed and I/O is connected to your terminal or pipeline. The `ilb`(1) man page documents the `ilb` utility, and section 14 is included in this manual to enable you to administer this utility.

- The NQE documentation was revised to support the new features described on this New Features page.

Record of Revision

The date of printing or software version number is indicated in the footer.

Version	Description
1.0	December 1993. Original printing. This publication describes how to use the Cray 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.

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 from Cray Research that lets users submit, monitor, and control batch requests for execution on an NQS server in an execution server complex (also known as a *batch complex* or *cluster*).

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.

This publication is written for NQE administrators who are responsible for defining and maintaining the NQE network configuration.

Related publications

The following documents contain additional information that may be helpful:

- *Introducing NQE*, publication IN–2153, provides an overview of NQE functionality , describes how to access documentation online, and provides basic user-level troubleshooting information. This publication also may be accessed online by using the Cray Research CrayDoc online documentation reader.

- *NQE Installation*, publication SG–5236, describes how to install the NQE software in a customized environment.
- *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 document reader or by using the NQE GUI help function.
- *CrayDoc Reference Card*, publication SQ–6101 (also provided online)

For information about installation and configuration of NQE on UNICOS systems, see the *UNICOS NQS and NQE Administrator's Guide*, publication SG–2305.

Ordering publications

The *User Publications Catalog*, 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.

Licensing and ordering NQE product

To order additional copies of this product, or to increase the number of licensed users, contact the Cray Research, Inc. in one of the following ways:

1–800–289–2729

+1–612–683–3897

- Send mail to the following address:

  CrayDirect
  Cray Research, Inc.
  655A Lone Oak Drive
  Eagan, Minnesota 55121
  USA

- Send electronic mail from any system connected to Internet to the following address:

  `craysoft@cray.com`

- Send a facsimile (fax) to the following telephone number:

  +1–612–683–3559

- Contact Cray Research through the World Wide Web (WWW) at the following URL:

  `http://www.cray.com/craysoft`

Conventions

The following conventions are used throughout this manual:

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 2 System calls 3 Library routines, macros, and opdefs 4 Devices (special files) 4P Protocols 5 File formats 7 Miscellaneous topics 8 Administrator commands Some internal routines (for example, the `ddcnt`() 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.
`ABBreviation` or `abbreviation`	Capital letters of a `qmgr` command or underlining indicates the shortest possible abbreviation for a command; underlining also highlights a summary phrase or sentence.

Convention	Meaning
[ ]	Brackets enclose optional portions of a command line.
...	Ellipses indicate that a preceding command-line element can be repeated.

The following machine naming conventions may be used throughout this manual:

Term	Definition
Cray PVP systems	All configurations of Cray parallel vector processing (PVP) systems, including the following: CRAY C90 series (CRAY C916, CRAY C92A, CRAY C94, CRAY C94A, and CRAY C98 systems) CRAY C90D series (CRAY C92AD, CRAY C94D, CRAY C98D systems) CRAY EL series (CRAY Y-MP EL, CRAY EL92, CRAY EL94, and CRAY EL98 systems) CRAY J90 series (CRAY J916 and CRAY J932 systems) CRAY T90 series (CRAY T94, CRAY T916, and CRAY T932 systems) CRAY Y-MP E series (CRAY Y-MP 2E, CRAY Y-MP 4E, CRAY Y-MP 8E, and CRAY Y-MP 8I systems) CRAY Y-MP M90 series (CRAY Y-MP M92, CRAY Y-MP M94, and CRAY Y-MP M98 systems)
Cray MPP systems	All configurations of Cray massively parallel processing systems, including the CRAY T3D series (CRAY T3D MC, CRAY T3D MCA, and CRAY T3D SC systems), and CRAY T3E series

Term	Definition
All Cray Research systems	All configurations of Cray PVP and Cray MPP systems that support this release
SPARC systems	All SPARC platforms that run the Solaris operating system version 2.3 or later

The default shell in the UNICOS operating system, 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 operating system also supports the optional use of the C shell.

Cray UNICOS Version 9.0 is an X/Open Base 95 branded product.

The POSIX standard uses *utilities* to refer to executable programs that Cray Research documentation usually refers to as *commands*. Both terms may appear in this document.

Reader comments

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

- Send us electronic mail from a UNIX system, using the following UUCP address:

  `uunet!cray!publications`

- Send us electronic mail from any system connected to the Internet, using the following Internet address:

  `publications@timbuk.cray.com`

- Customers in the United States and Canada can call the Cray Research Customer Support Center at either of the following telephone numbers. These telephone numbers are also listed on the cs_support(7) man page and on the Service and Support card included in the product package.

  1–800–272–9349 (toll free from the United States and Canada)

  +1–404–631–2244

- Customers in other countries can call their local Cray Research support office. These telephone numbers are listed on the cs_support(7) man page and on the Service and Support card contained in the product package.

- Contact your Cray Research representative and ask that a Software Problem Report (SPR) be filed. 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" in Eagan, Minnesota, at the following fax number:

  +1–612–683–5599

- Use the postage-paid Reader's Comment Form at the back of the printed manual.

We value your comments and will respond to them promptly.

Contents

Page

Figures

Tables

NQE Overview [1]

The Network Queuing Environment (NQE) is a framework for distributing work across a network of heterogeneous systems. NQE consists of the Cray Network Queuing System (NQS), NQE clients, Network Load Balancer (NLB), the NQE database and its scheduler, and the File Transfer Agent (FTA).

Scope of this manual

1.1

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

- Section 1 is an overview of NQE.
- Section 2 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.
- Section 3 describes the NQE configuration file (`nqeinfo`) and changes you can make to it, including configuration of the NQE client component of NQE.
- Section 4 describes configuration and management of the NQS component of NQE. It describes the use of `qmgr` commands that are available only to NQS managers.
- Section 5 describes the `qmgr` commands that are available to NQS operators.
- Section 6 describes configuration and management of the NLB components of NQE.
- Section 7 describes how to implement load-balancing policies with NQE.
- Section 8 describes an overview of the NQE database and its components and describes management and configuration of the NQE database and the NQE scheduler.

- Section 9 describes how to modify the NQE scheduler to meet the needs of your site.
- Section 10 describes how to use the `csuspend`(8) command to make use of unused cycles on a server.
- Section 11 describes how the job dependency feature affects NQE administration.
- Section 12 describes configuration and management of the FTA component of NQE.
- Section 13 describes configuration of the Distributed Computing Environment (DCE) when using NQE.
- Section 14 describes configuration of the `ilb` command.
- Section 15 provides possible solutions to problems you may encounter as an NQE administrator.

NQE environment

1.2

NQE supports computing in a large network made up of clients and servers. This grouping of clients and servers is referred to as an *execution server complex* (also known as a *batch complex* or *cluster*). The servers provide reliable, unattended processing and management of the complex. Users who have long-running requests and a need for reliability can submit batch requests to an execution server complex.

NQE clients support the submission, monitoring, and control of work from the workstation for job execution of the batch requests on the servers. 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 network where users need an interactive interface to the batch complex. The NQE client provides the NQE GUI, which is accessed through the `nqe` command. It also provides a command-line interface. For a list of user-level commands, see appendix A, page 355. 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 the 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 the execution server complex. 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 network. 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 light-weight 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 section 8, page 197, and section 9, page 229.

The NLB provides status and control of work scheduling within the execution server complex. Sites can use the NLB to provide policy-based scheduling of work in the complex. The NQE graphical user interface (GUI) `Status` window provides refreshed status of work within the execution server complex. 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 and to the `cload` program, which display network request status 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 network contains the following components:

- 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.

- NQE *execution servers* (which are replicated over several machines) also contain NQE client software. In addition, they also contain the NQS, FTA, NLB collectors, and the NQE database LWS. These additional components have the following functions:

  – NQS schedules and initiates batch requests.

  – FTA transfers files.

  – NLB collectors 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 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.

- An NQE *master server* contains all NQE components. To distribute work among all NQE execution servers, a master server runs an NLB server; it also runs the NQE database and the NQE scheduler. A master server also contains all of the software that clients and execution servers contain, and they can function as execution servers in your execution server complex. Only one NQE master server is required in an execution server complex, although there may be more.

Figure 1 shows the location of the NQE components within the NQE network:

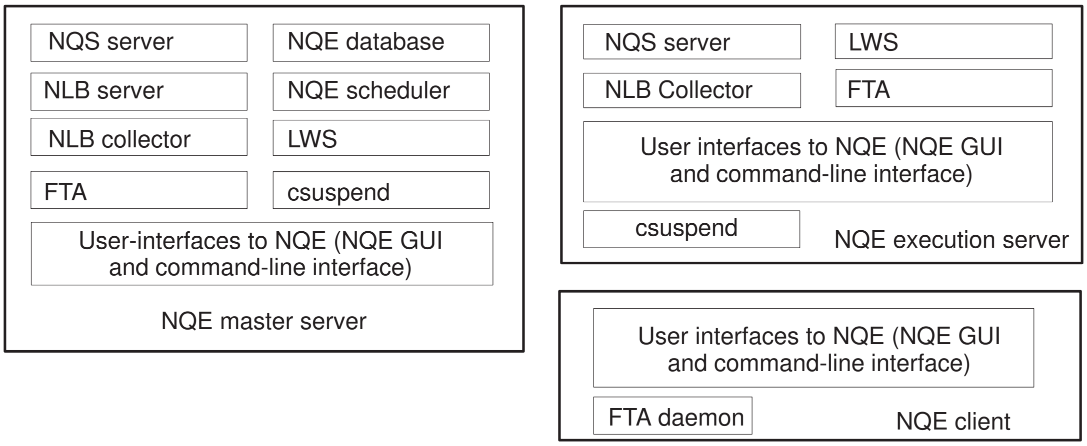

Figure 1. NQE component location within

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 any node in the execution complex, but they will only be initiated on an NQE server.

FTA can be used from any NQE 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 NQE 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 all NQE master servers, but it is easiest to run it initially on the first node where you install NQE. Redundant NQE master servers ensures greater availability of the NLB database if an NQE master server is unreachable through the network.

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

You can start the `csuspend`(8) utility on any NQE 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 require access to the following client commands: `nqe` (which invokes the NQE GUI), `cevent`, `cqsub`, `cqstatl`, `cqdel`, and `cload`. 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` utility so that a user may execute a load-balanced interactive command; for more information, see section 14, page 339.

Running NQE as an environment

1.3

The example given in this subsection shows how the NQE components work as an environment. For information on NQE user features, see *Introducing NQE,* publication IN–2153, and the *NQE User's Guide*, publication SG–2148. Both documents are available online through the same interface as this manual.

Figure 2 shows a possible NQE configuration. The user `ljgm` 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`). `ljgm` wants the output from her batch request to go to her research assistant (`fred`) at another client workstation, `gale`, which has an FTA connection to `latte`.

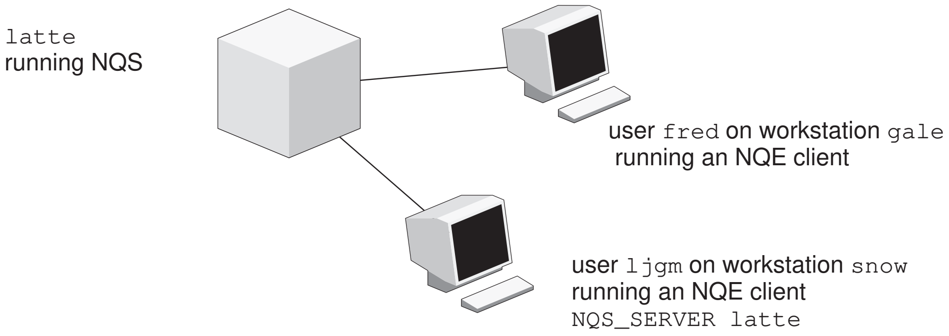

Figure 2. Sample NQE configuration

User `ljgm` has several batch requests to run. One of the requests (named `jjob`) looks like this:

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

The request script uses FTA (the rft command) to transfer its files. The FTA domain name nppa_latte and the option -nopassword indicate that peer-to-peer authorization is used, so ljgm does not need to specify a password:

```
rft -user ljgm -host nppa_latte -nopassword -function get jan.data nqs.data
```

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

```
cqsub jjob
```

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

The load-balancing policy for ljgm's site allows work to be shared among NQS systems on latte, pendulum, telltale, and gevalia. Because of the workload on the machines, ljgm'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 ljgm 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 ljgm can monitor the progress of all her requests. Because she used the embedded #QSUB -me option, she receives mail when the requests complete.

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.

Using the World Wide Web example

1.4

The NQE release contains an example World Wide Web (WWW) interface to NQE. You can access the interface through WWW clients such as Mosaic or Netscape. The interface lets IBM PC, Apple Macintosh, workstation, and superserver users access NQE. 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.

This interface is provided to NQE administrators as an example for site-specific modification and inclusion in your WWW server environment.

For information on installing and managing the interface, read the *target_root* `/craysoft/www/README` file. The default directory for *target_root* is `/usr` on most systems; on Solaris systems it is `/opt`.

Concepts and Terms [2]

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

Section 2

Batch requests

2.1

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 files 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 subsections.

Nice values

2.1.1

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.

Section 2

Shell interpretation
2.1.2

A batch request can specify the full path name of a shell to interpret its shell script. To specify the full path name of a shell, use the `General Options` selection of the `Configure` menu on the NQE GUI `Submit` window or use the `cqsub -s` user command option. If no shell is specified, 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 `qsub -s` option.

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 `qsub`.

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 command-line interface 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
/usr/craysoft/nqe/3.0/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
```

Section 2

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
/usr/craysoft/nqe/3.0/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 using qsub -s, NQS invokes the request owner's UDB shell on UNICOS systems or their shell on UNIX systems.

To set a fixed shell path and invocation method, edit the /etc/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 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.

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.

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.

Intraqueue priority 2.1.3

The *intraqueue priority* determines the order in which requests are initiated and the order in which requests are displayed by the `cqstatl` command. 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).

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 = *(((share_wt)*share_pri) +*
*((time_wt)*time_in_que −*
*((cpu_wt)*cpu_requested) −*
*((mem_wt)*mem_requested/max_pri)*

The *share_wt* , *time_wt* , *cpu_wt* , and *mem_wt* are determined by the `qmgr` commands `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 *share_wt* value is valid only for UNICOS 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-que* value are from the batch request structure. The *max_pri* value is the largest calculated priority (that is, the largest numerator) for all the requests considered. *share_pri* is valid only for UNICOS systems.

If a user specifies a priority by using the `General Options` selection of the `Configure` menu on the NQE GUI `Submit` window or by using the `cqsub -p` command, the priority is displayed by `cqstatl` while the request resides in a pipe queue, and it continues to be in the request 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.

Resource limits
2.1.4

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 queue in which the request resides.

The limits supported by NQS are described in subsection 4.10, page 73.

Request states

2.2

The state of a request can be found by using the NQE GUI `Status` window or the `cqstatl -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` option command.

The current state of the request shown on the `cqstatl` 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) man page.

Section 2

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` option 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 subsection) 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

State	Description
`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
`suspended`	Processing temporarily suspended by an operator

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.

Queues

2.3

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 subsections 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.

Batch queues
2.3.1

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 assigns the limit for the request to the corresponding default limit of the queue. The `qmgr` commands define the queue default values.

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. The process and request limits that NQS enforces are described in the next subsection.

Every batch queue also has a set of associated queue limits, which are described in subsection 2.5, page 25.

Pipe queues
2.3.2

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), 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.

Queue, queue complex, and global limits

2.3.3

Queue, queue complex, and global limits (defined beginning with subsection 2.5, page 25) 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`.

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.

Request processing

2.4

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

Request destination

2.4.1

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 section 8, page 197.

Local submission
2.4.2

When a user submits a batch request to NQS using the NQE GUI Submit window or the cqsub command, 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.

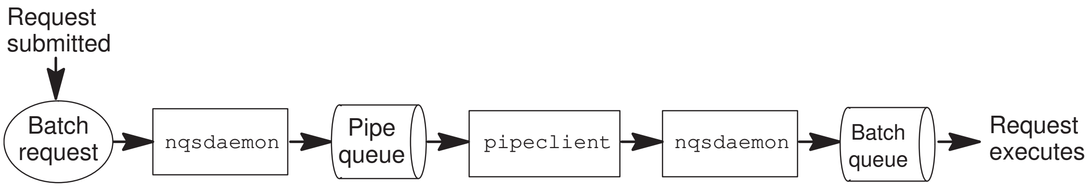

Figure 3. Request processing for local submission

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

NQE client submission
2.4.3

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

Client submission to NQS
2.4.3.1

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 to handle the request. The child process (that is, the netserver process) in turn 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.

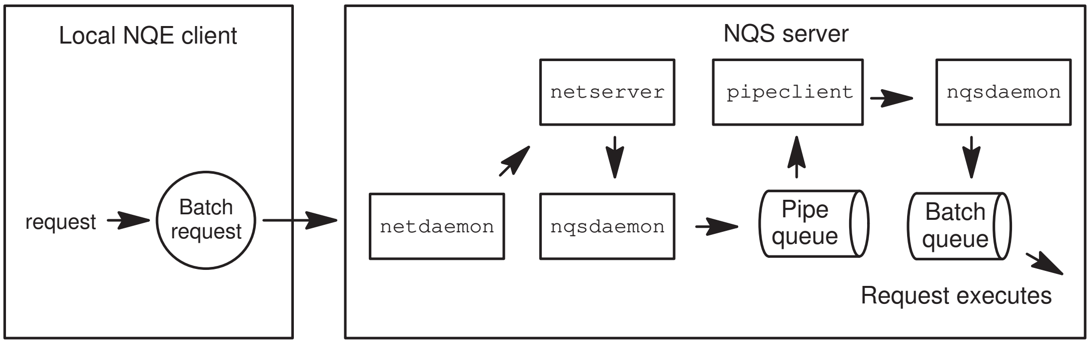

Figure 4. Request processing for client submission to NQS

Client submission to NQE database
2.4.3.2

When a user submits a request to NQS using the NQE GUI `Submit` window or the NQE `cqsub` command, the request is sent to the NQE database. The NQE scheduler determines when and where the request will execute. The NQE database routes a copy of the request to the light-weight server (LWS) on the selected execution 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.

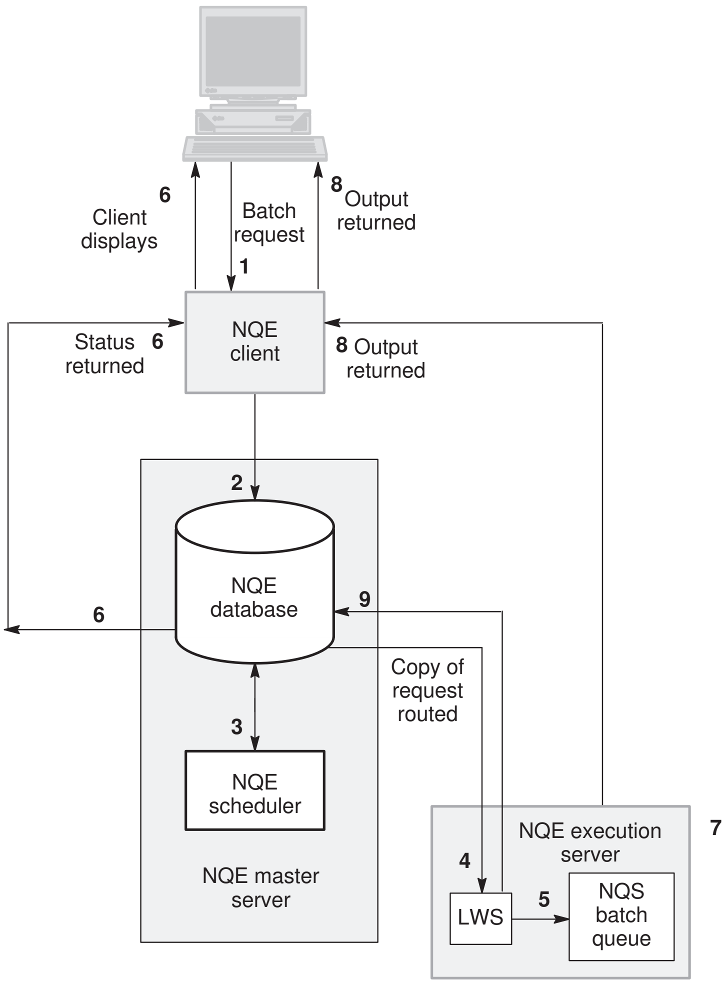

Figure 5. Request processing for client submission to the NQE database

Remote submission
2.4.4

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. Figure 6 shows this process.

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

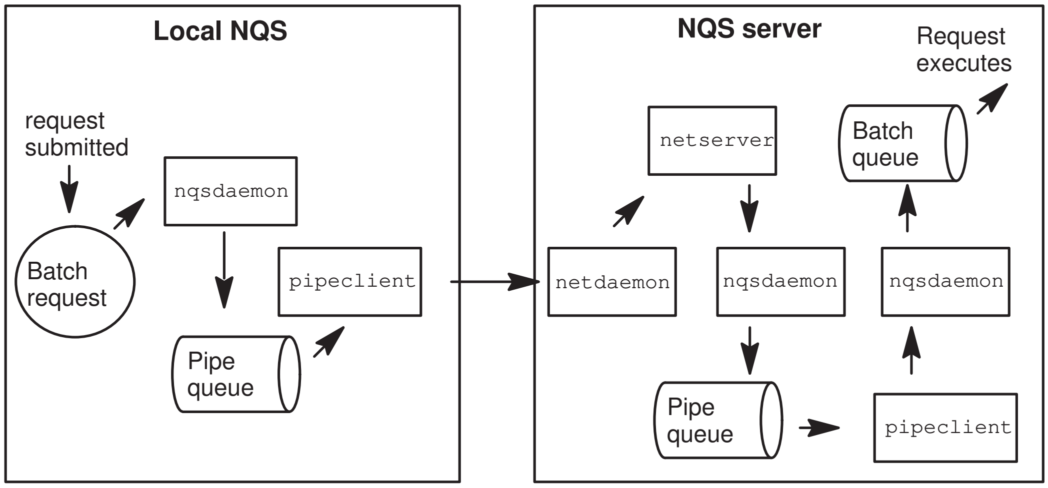

Figure 6. Request processing for remote submission

Daemon processing
2.4.5

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.

Controlling and deleting requests

2.4.6

NQS users can control or delete their own requests by using the NQE GUI `Status` window `Actions` menu options or by using the `cqdel` command. NQS operators and managers can control requests with the following `qmgr` commands (see subsection 5.6, page 125, 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 also can delete requests from any user by using the NQE GUI `Status` window `Actions` menu options or by using the `cqdel` command.

Displaying queue status

2.4.7

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

Queue attributes

2.5

Queue attributes define the properties of a queue. A queue attribute (*attribute_name*) 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 subsections describe possible attributes for NQS queues. For additional information about defining queue properties, see subsection 4.5, page 56.

2.5.1 Batch and pipe queue attributes

The queue attributes described in the following subsections 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.

Queue run limit
2.5.1.4

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

Batch queue attributes
2.5.2

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

`loadonly`
2.5.2.1

You may not submit requests directly to a `loadonly` queue. Requests may come from only another 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*.

Section 2

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.

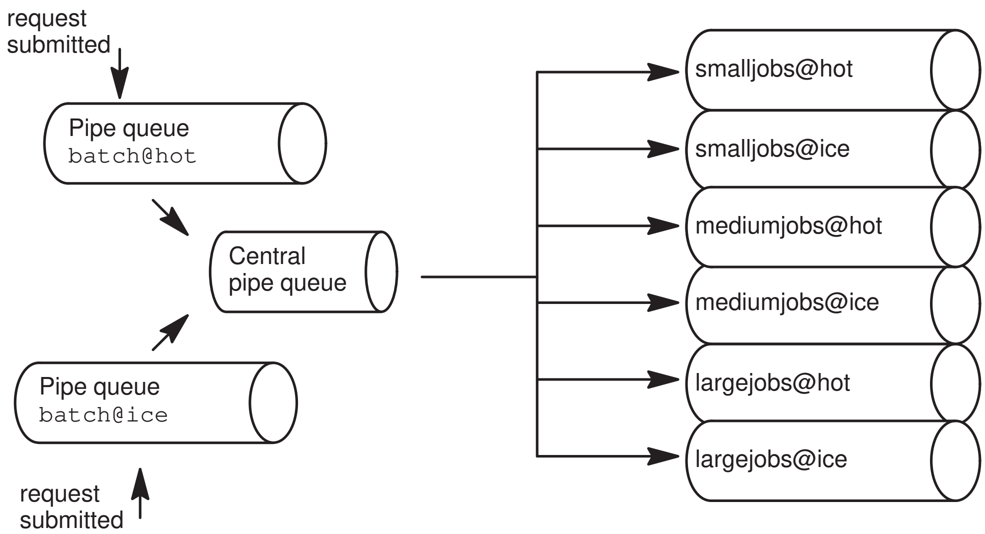

Figure 7. Queue configuration that uses the `loadonly` attribute

Queue group limit
2.5.2.2

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.

Queue memory limit
2.5.2.3

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.

Queue user limit
2.5.2.4

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.

Queue states
2.6

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.

Queue complexes

2.7

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 (see subsection 2.5, page 25, for information about queue attributes and subsection 2.8 for information about global limits). The following queue complex limits can be set:

- Group limits
- Memory limits
- Run limits
- User limits

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.

Subsection 4.8, page 69, describes how to set queue complex limits.

Global limits

2.8

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

Subsection 4.9, page 71, describes how to set global limits.

Load balancing and destination selection

2.9

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 an execution server complex. 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 subsections.

NLB
2.9.1

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

NLB servers
2.9.1.1

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 what permissions that particular user has. 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.

Section 2

Destination selection 2.9.1.2

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 execution server complex. It then provides NQS with a list of machines that is 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.

Policies 2.9.1.3

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.

NLB collectors 2.9.1.4

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 section 6, page 135.

NLB clients
2.9.1.5

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

NQE database
2.9.2

The NQE database provides the following:

- A mechanism for the client user to submit requests to a central storage database.
- An 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:

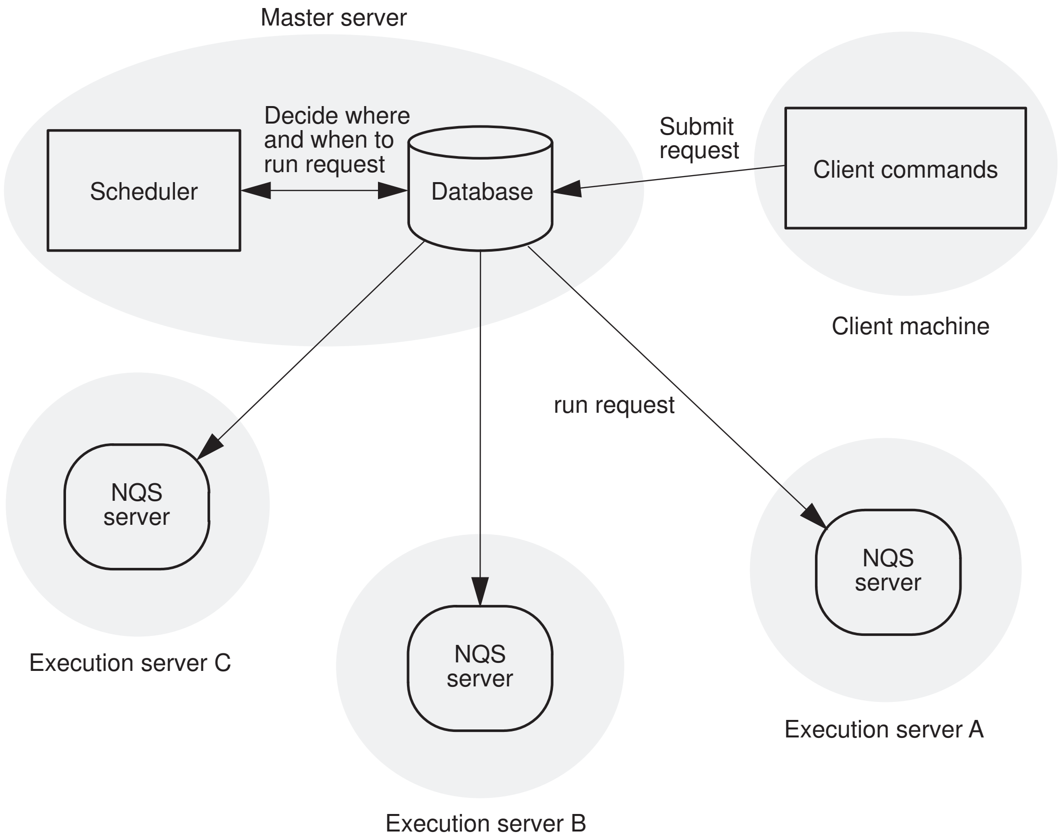

Figure 8. Basic concept of the NQE database

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

NQE database server (mSQL)

2.9.2.1

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

NQE database monitor

2.9.2.2

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.

Scheduler

2.9.2.3

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

- It keeps track of light-weight 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 section 9, page 229).

Light-weight server (LWS) (bridge to NQS)

2.9.2.4

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.

Section 2

File transfer

2.10

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

File transfer services

2.10.1

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

Domain names

2.10.2

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.

Section 2

FTA users

2.10.3

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.

Network peer-to-peer authorization

2.10.4

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.

File transfer requests

2.10.5

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.

Queue directories

2.10.6

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

The `nqeinfo` File [3]

This section discusses the `nqeinfo` file and variables you can use in NQE configuration.

The `nqeinfo` file is created by the NQE installation procedure. It contains the NQE configuration parameters. NQE master servers, execution servers, and clients must have a unique `nqeinfo` file. By default, the installation procedure places the `nqeinfo` file in the `$NQE_ROOT/craysoft/nqe/etc` directory and creates a link from `/etc/nqeinfo` to `$NQE_ROOT/craysoft/nqe/etc/nqeinfo`.

When the NQE installation procedure creates the `nqeinfo` file, the file has a set of configuration parameters defined that match the selections you made during installation. Comment lines describe each configuration parameter .

For a list of `nqeinfo` file variable names that were previously included in the `config.h` file, see appendix B, page 361.

> **Note:** Some variables may be set in the user's environment to indicate individual preferences. The `nqeinfo` file variables will be used only if an individual has not indicated a preference. For additional information about these variables, see the *NQE User's Guide*, publication SG–2148.

3.1 Configuration management

You can change the configuration by editing the `nqeinfo` file, but you should use caution. Even typographical errors in this file can cause NQE to fail.

> **Caution:** Always stop NQE before editing the `nqeinfo` file. Each NQE command and server program reads the `nqeinfo` file once during command or server initialization. Changes made to the `nqeinfo` file while NQE is running can result in NQE commands using different `nqeinfo` file values than the NQE server processes and will result in unpredictable problems.

To edit the nqeinfo file, perform the following steps:

1. Log in as root.
2. Stop NQE by using the following command:

   ```
   nqestop
   ```

3. Edit the nqeinfo file to contain the configuration variables you want to use.
4. Start NQE by using the following command:

   ```
   nqeinit
   ```

Do not edit the file to include more than one occurrence of a configuration variable. If two occurrences exist, only the first occurrence is used.

Section 3

Configuration variables not included in the nqeinfo file

3.2

The following configuration variables are not included in your nqeinfo file; you must add them to the file manually if you want your configuration to support their functions:

Configuration variable	Description
NQS_PASSWORD_NEEDED="*value*"	This variable should be added if you want users to be prompted for a password. If this variable is present, NQS will prompt for a password. The NQS_PASSWORD_NEEDED variable ensures that the user is prompted for a password if the user did not specify one when the request was submitted. The NQS_PASSWORD_NEEDED variable may be set to any value to turn on password prompting.

Configuration variable	Description
NQE_NQS_NQCACCT="*value*"	Defines validation behavior when a client user submits a batch request. It is read by the NQS server and is used to control how client users are authorized. You should use this variable if you have client user accounts that are different from your server accounts. When set to ORIGIN, NQS will always validate the originating client user account on the NQS server (that is, it authenticates the client user on the NQS server using the same account name). When set to TARGET, NQS validates the target user account when it receives a client batch request that designates a user name other than their default user name. If you do not include this variable in your nqeinfo file, the default action of your configuration will be as if you used this variable set to be ORIGIN.
NQE_AUTHENTICATION="*value*"	DCE variable; indicates that DCE authentication is available. If a user supplies a password with a job request, NQS will use it to attempt DCE authentication. NQS authentication will still be done according to the method indicated by qmgr or nqedbmgr. For additional information about configuring DCE, see subsection 13, page 335.
NQE_DCE_BIN="*value*"	DCE variable; tells NQS where to find the dce_login command. If it is not present, NQS attempts to execute /opt/dcelocal/bin/dce_login. Because DCE is optional on a system, no checks are performed on the value to insure the directory exists. For additional information about configuring DCE, see subsection 13, page 335.

Section 3

Common configuration variables you may want to change

3.3 The following configuration variables are ones that are commonly changed in an nqeinfo file:

Configuration Variable	Description
NQE_NQS_DEF_PATH="*value*"	Defines the default path NQS sets up before starting a request. The initial setting (default) for the path is /bin:/usr/bin:/usr/ucb. If you are running NQE on a UNICOS system running UNICOS 9.0.*x* or earlier or running UNICOS 9.1 and have access to source files, you may overwrite the default by modifying /usr/src/net/nqe/src/nqe/include/config.h and recompiling NQS.
NQE_NQS_DEF_PATH_ROOT="*value*"	Defines the default path NQS sets up before starting a request that is owned by root. The initial setting (default) for the path is /bin:/usr/bin:/usr/ucb:/etc. If you are running NQE on a UNICOS system running UNICOS 9.0.*x* or earlier or running UNICOS 9.1 and have access to source files, you may overwrite the default by modifying /usr/src/net/nqe/src/nqe/include/config.h and recompiling NQS.
NQE_NQS_SHELLPATH="*value*"	Defines a shell path for batch request processing. By default, NQS uses the /bin/sh shell. Users can override this value (either the default shell or the value set by using this parameter) by editing their request files. If the value of this parameter is any string other than a valid shell path, the nqsdaemon aborts on startup.

Configuration Variable	Description
NQE_NQS_SHELLINVOCATION="*value*"	Defines the number of shells spawned when NQS initiates a request. By default, NQS spawns a second shell when it initiates a request. If you set this parameter to a value of 1, NQS invokes one shell that is either /bin/sh or the value of the NQE_NQS_SHELLPATH parameter. If the value of this parameter is not 1 or 2, the nqsdaemon aborts on startup.
NQE_DEST_TYPE="*value*"	Designates the default destination of client requests; valid values are nqs and nqedb. The default value is nqs; this variable is not required if you do not use the NQE database.

Section 3

NQS Configuration [4]

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.

This section 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 section 5, page 101. This section discusses the following topics:

- Configuration guidelines
- Overview of manager actions
- Defining the NQS hosts in the network
- Defining managers and operators
- 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
- 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

Configuration guidelines

4.1

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 (see the `qstart`(8) command 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 subsection 2.1.3, page 14, for a description of intraqueue priority and subsection 4.11, page 78, for a description and examples of the `set sched_factor` commands.

Overview of manager actions

4.2

An NQS manager is responsible for the configuration of an NQS system. An NQS manager also can perform all of the operator actions; see section 5, page 101, 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 subsection 5.2, page 102, for a description of how to start `nqsdaemon`

Defining NQS hosts in the network

4.3

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.

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 subsection 4.3.1, page 50): `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.

Section 4

The *mid* and the *hostname* 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 subsection 4.3.1.

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.

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 *host name* in the network database. 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.

Configuring remote output agents

4.3.1

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 section 11, page 295.

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 `/etc/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 section 12, page 299, 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.

Section 4

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.

Section 4

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 section 12, page 299, and the `fta`(8) and `nqsfts`(8) man pages for details on how to configure FTA. NQS uses the FTA domain name `nqs-rqs` to spool output over TCP/IP.

Commands for defining machines
4.3.2

You can use the following `qmgr` command to add a new entry to the *mid* database (see Table 1, page 48, 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.

`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 subsection 4.3.1, page 50, 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 mid* | *hostname*
`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 subsection 5.4.6, page 122, for an example display.

4.3.3 TCP/IP network configuration example

Figure 9 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.

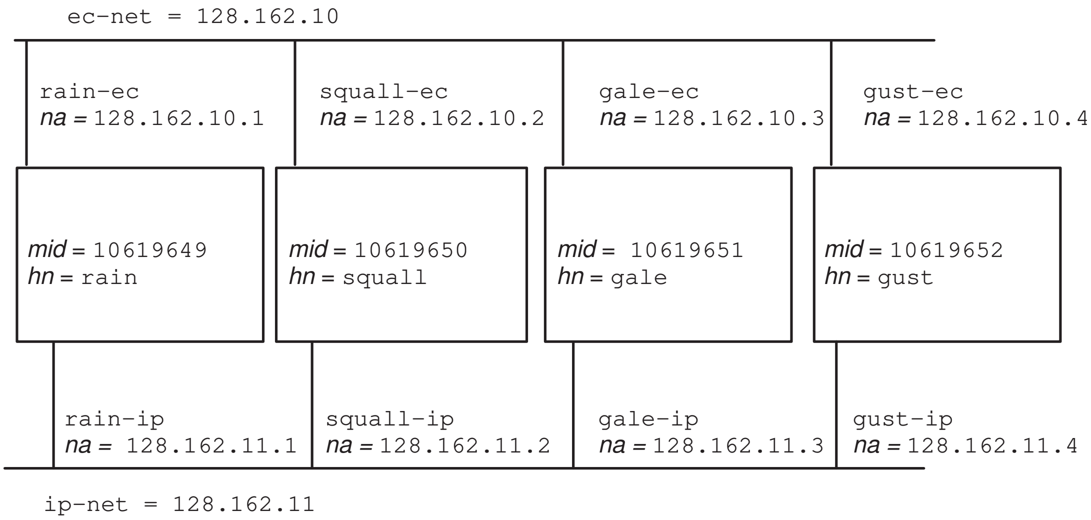

Figure 9. Example of a TCP/IP network

Section 4

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

Section 4

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
```

Stand-alone configuration example

4.3.4

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.

Defining managers and operators

4.4

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. Only the `exit`, `help`, and `quit` and commands starting with `show` are available to users without special privileges.

When NQS is initially installed, only `root` is defined as the NQS manager (no operators are defined). Therefore, `root` must run `qmgr` first to define other users as either a manager or an operator. You cannot delete the user `root` from the list of managers.

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.

Section 4

Commands for defining managers and operators

4.4.1

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 subsection 5.4.2, page 112, for an example display.

Example of adding managers and operators
4.4.2

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
```

Defining NQS queues
4.5

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. Subsection 2.3, page 18, provides background information about queues and how they work.

NQE is shipped with one pipe queue and one batch queue configured on each NQS execution 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 execution server.

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

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

Table 3 show 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
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*`@`*remote_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

Section 4

Table 3. Definable queue characteristics (continued)

Characteristic	Description	Queue type
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
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
`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 subsection 2.5.2, page 27, for a description of how `loadonly` queues can be used.	Batch

Table 3. Definable queue characteristics (continued)

Characteristic	Description	Queue type
`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.	Pipe
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. The standard location is `pipeclient`; if you do not use the standard location, you must declare an absolute path name. A pipe queue can be defined as a destination-selection (load-balancing), as described in subsection 4.5.4, page 64. 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

Section 4

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 subsection 4.7, page 67.

- You can specify global system defaults for queues. See subsection 4.9, page 71.
- You can specify limits on the number and size of the requests in a queue, as described in the next subsection.

Setting queue limits
4.5.1

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.

Some queue limits are enforced only for certain machines. For more information on these limits, see Table 4, page 74.

Hints for defining queues
4.5.2

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.

Section 4

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 subsection 5.5.1, page 124). 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 subsection 5.5.2, page 124).

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

Commands used to define queues
4.5.3

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 57, 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 subsection 4.5.4, page 64, for information on configuring a destination-selection queue.

Section 4

Adding destinations to a pipe queue
4.5.3.1

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.

Section 4

Resetting destinations for a queue
4.5.3.2

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.

Deleting destinations from a queue
4.5.3.3

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.

Changing the interqueue priority for a queue
4.5.3.4

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

`set priority =` *priority* *queuename*

Changing the run limit for a queue
4.5.3.5

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

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

Changing the pipe client server for a queue
4.5.3.6

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*

Section 4

Commands for configuring destination-selection queues

4.5.4

Load balancing is described in section 7, page 155. The 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 queuename  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 (see the next subsection for an example). 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. 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
```

Section 4

Displaying details of the currently defined queues

4.5.5

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

`cqstatl -f` *queues*

See subsection 5.4.4, page 116, for an example display.

Examples of defining queues

4.5.6

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
```

Section 4

Example of changing the order of the destinations

4.5.6.1

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
```

Deleting a queue

4.5.7

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 subsection 4.6).

Section 4

Defining a default queue

4.6

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
```

Displaying the current default queue
4.6.1

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

```
show parameters
```

The setting is shown next to `Default batch_request queue`. See subsection 5.4.1, page 104, for an example display.

Example of defining a default queue
4.6.2

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

```
set default batch_request queue bqueue10
```

Restricting user access to queues
4.7

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.

Hints on restricting user access to queues
4.7.1

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`.

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`.

Commands available to restrict user access

4.7.2

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*

Displaying details of the currently defined queues

4.7.3

To display the current restrictions on user and group access to a queue, use the `cqstatl` command instead of a `qmgr` utility, as follows:

`cqstatl –f` *queues*

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

See subsection 5.4.4, page 116, for an example display.

Example of restricting user access

4.7.4

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
```

Defining queue complexes

4.8

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.

Section 4

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. It can 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
`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

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 74.

Queue complex examples
4.8.1

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

```
create complex=( bque11,bque12,bque13 night) 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
```

Defining global limits

4.9

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:

Limit	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
`pipe_limit`	Sets the maximum number of pipe requests allowed to run concurrently at the host
`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
`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 74.

Section 4

Displaying the current global limits
4.9.1

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 subsection 5.4.1, page 104, for an example. Some parameters are not enforced if the operating system does not support the feature, such as MPP processor elements.

Section 4

Example of setting limits
4.9.2

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
```

Defining per-request and per-process limits

4.10

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 shows the limits that can be enforced on NQS requests. This table uses the following notation:

Symbol	Meaning
✓	Limit is supported and enforced on this platform
✗	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.

Section 4

Table 4. Platform support of NQS limits

Description	Limit type	qsub qualifier	Run-time enforcement on executing NQS: UNICOS	Solaris	SunOS	AIX	DEC OSF	HP-UX	IRIX
Core file size	PP	`-lc`	✓	✓	✓	✓	✓	✗	✓
Data segment size	PP	`-ld`	✗	✓	✓	✓	✓	✗	✓
Permanent file size	PP	`-lf`	✓	✓	✓	✓	✓	✗	✓
	PR	`-lF`	✓	✗	✗	✗	✗	✗	✗
Memory size	PP	`-lm`	✓	✗	✗	✗	✗	✗	✗
	PR	`-lM`	✓	✗	✗	✗	✗	✗	✗
Nice value	PP	`-ln`	✓	✓	✓	✓	✓	✓	✓
Quick file size	PP	`-lq`	✓	✗	✗	✗	✗	✗	✗
	PR	`-lQ`	✓	✗	✗	✗	✗	✗	✗
Stack segment size	PP	`-ls`	✗	✓	✓	✓	✓	✗	✓
CPU time	PP	`-lt`	✓	✓	✓	✓	✓	✗	✓
	PR	`-lT`	✓	✓ †	✓ †	✓ †	✓ †	✗	✓ †
Tape drive limit	PR	`-lU`	✓	✗	✗	✗	✗	✗	✗
Temporary file size	PP	`-lv`	✗	✗	✗	✗	✗	✗	✗
	PR	`-lV`	✓	✗	✗	✗	✗	✗	✗

Table 4. Platform support of NQS limits (continued)

Description	Limit type	qsub qualifier	UNICOS	Solaris	SunOS	AIX	DEC OSF	HP-UX	IRIX
Working set size	PP	`-lw`	✗	✓	✓	✓	✓	✗	✓
MPP processing elements	PR	`-l mpp_p`	✓	✗	✗	✗	✗	✗	✗
MPP residency time (T3D systems)	PP	`-l p_mpp_t`	✓	✗	✗	✗	✗	✗	✗
	PR	`-l mpp_t`	✓	✗	✗	✗	✗	✗	✗

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

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

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. 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.

Section 4

Option	Description
`memory_limit`	Sets the per-process or per-request maximum memory size limit for the specified 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.
`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.

Displaying per-process and per-request limits

4.10.1

To check the per-process and per-request limits assigned to a queue, use the following command:

```
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 subsection for a partial example, or subsection 5.4.4, page 116, for a full example display.

Example of setting per-process and per-request limits

4.10.2

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, it appears in the `cqstatl` 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>
.
.
.
```

Section 4

Some parameters are not enforced if the operating system does not support the feature; see Table 4, page 74, for a list of limit enforcements.

How queue limits affect requests

4.10.3

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).

Defining job scheduling parameters

4.11

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
```

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	Valid only on Cray UNICOS systems
cpu	Per -request CPU limit requested
memory	Per -request memory limit requested
time	Time the request has been waiting in the queue for initiation

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

Section 4

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.

4.11.1 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.

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:

```
set sched_factor share = 0
set sched_factor cpu = 100
set sched_factor memory = 0
set sched_factor time = 100
```

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

Section 4

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.

The `qmgr` command `set sched_factor share` is valid only on UNICOS systems.

4.11.2 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      = 100
   Requested Memory        = 0
   Time-in-queue           = 100
```

4.12 Setting the log file

Section 4

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

Section 4

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`. See subsection 4.13 for more information.

Displaying information on the current log file
4.12.1

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 subsection 5.4.1, page 104, for an example display.

4.12.2 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 subsection 4.12, page 80.

4.13 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.

Section 4

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 subsection 5.4.1, page 104, for an example display.

4.14 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.

4.14.1 Log file location

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

On UNICOS, NQS daemon accounting information is set using the standard UNICOS accounting system.

On UNIX, NQS simply writes accounting records to the file `private/root/nqsacct` 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 systems, it is up to the UNICOS 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`.

Accounting records

4.14.2

Accounting records are written to the NQS log and/or the NQS accounting file. The following table 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

Section 4

Table 5. Accounting records (continued)

Type	Subtype	Format	Description
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

Table 5. Accounting records (continued)

Type	Subtype	Format	Description
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
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

Format	Description
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 job
cstime	Sum of system CPU time for all processes in job

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

Section 4

Format	Description
subtype	Record subtype
host_name	Host name on which the record was written
request_name	NQE request name
queue_name	NQE queue name

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 runtime alterable.

The following table describes the unit value of `sysconf (_SC_CLK_TCK)` for each workstation type. Time in seconds can be computed by dividing the times reported in the accounting record and dividing by the unit.

Table 6. Unit value of `sysconf (SC_CLK_TCK)`

Platform	Unit
AIX	100
HP-UX	100
IRIX	100
OSF/1	60
SunOS 4	60
Solaris	100

Section 4

Setting the validation strategy for remote access

4.15

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 execution server complex can set a validation type. To avoid confusing users, all NQS servers in the execution server complex 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.

Validation type	Description
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 execution 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 `cqsub`, `cqstatl`, and `cqdel` commands). 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.

User validation commands
4.15.1

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 and validation file checking

Section 4

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 subsection 5.4.1, page 104, for a full example display.

Example of setting user validation
4.15.2

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

```
set validation password
```

Defining other system parameters
4.16

This subsection describes miscellaneous NQS configuration settings.

Setting retry limits
4.16.1

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.

Displaying the current retry limits
4.16.1.1

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 subsection 5.4.1, page 104, for a full example display.

Example of setting retry limits
4.16.1.2

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
```

Defining the sender of NQS mail
4.16.2

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.

4.16.2.1 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 subsection 5.4.1, page 104, for a full example display.

4.16.2.2 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.

4.16.3 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
```

4.16.3.1 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 subsection 5.4.1, page 104, for a full example display.

4.17 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 subsection 4.17.1 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.

Section 4

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*

4.17.1 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 subsection 5.4.1, page 104, 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.

Section 4

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 /usr/craysoft/nqe/database/spool/snapfile | qmgr > outfile
.
.
.
```

Section 4

Output validation

4.18 The NQS server returns output from requests to the NQC 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):

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 specified when using the NQE GUI to submit a request. FTA transfers the file according to the rules configured for the *domain*

Section 4

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.

Operating NQS [5]

This section 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 section.

This section 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

Overview of operator actions

5.1

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` utility to display the following:
  - The current values of system parameters
  - The status of NQS queues

Section 5

  - A detailed description of the characteristics of NQS queues
  - The status of individual requests in NQS queues

  You must use `cqstatl` 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

Starting NQS

5.2

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`(8) command (as follows), which then automatically starts the other NQS processes.

`% qstart` *options*

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`(8) and `qstart` commands check whether the NQS database has been created. If it has not, the commands create it. During its

Section 5

creation, default values are taken from one of the following two files, whichever exists:

```
$NQE_ETC/config/nqs_config
$NQE_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, `$NQE_ETC` and `$NQE_ETC/config` are 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` command to save the current configuration. To update the standard configuration file, you can replace the `nqs_config` file with the snap file.

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.

NQS shutdown and restart

5.3

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, 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. The `logdaemon` stops only when you use `nqestop`.

When the `qmgr shutdown` command shuts down NQS, all processes that make up a restartable running batch request on the local host are suspended. On UNICOS systems, an image of the request is saved by NQS using the `chkpnt`(2) system call in a file in the checkpoint directory.

Monitoring NQS

5.4

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` command to provide valuable information about requests and queues.

Displaying system parameters

5.4.1

To display the current values for system parameters, use the following `qmgr` command:

```
show parameters
```

Section 5

An example of the display follows:

```
Qmgr: sho parameters
sho para

  Checkpoint directory =
/usr/craysoft/nqe/3.0/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
  Job Scheduling:   Configured = nqs normal   Active = nqs normal
  Log_file = /usr/craysoft/nqe/3.0/pendulum/database/spool/../log/nqslog
  MESSAGE_Header = Short
```

(continued)

Section 5

```
  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 = /usr/craysoft/nqe/bin/netdaemon
  Network client = /usr/craysoft/nqe/bin/netclient
  Network retry time = 31 seconds
  Network server = /usr/craysoft/nqe/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 = /usr/craysoft/nqe/3.0/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 UNICOS 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`).

Section 5

Display entry	Description
`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.
`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 (†`set global pipe_limit`).
`Global quick-file limit`	The maximum quickfile space that can be allocated to all batch requests running concurrently on UNICOS NQS (†`set global quickfile_limit`).

Section 5

Display entry	Description
`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 Factors`	The weight of the specified factor in selecting the next request initiated in a queue (`set sched_factor cpu`| `memory`| `share`| `time`).
`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`).

Section 5

Display entry	Description
`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 systems) (`set periodic_checkpoint concurrent_checkpoints`).
`Periodic_checkpoint cpu_interval`	The default CPU time interval for periodic checkpointing (valid only on UNICOS 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 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 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`).

Section 5

Display entry	Description
Periodic_checkpoint min_cpu_limit	The minimum CPU limit for a request that can be periodically checkpointed (valid only on UNICOS 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 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 systems) (set periodic_checkpoint status off| on).
Periodic_checkpoint time_interval	The default wall-clock time interval for periodic checkpointing (valid only on UNICOS 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 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).

Section 5

Display entry	Description
`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`).

Displaying a list of managers and operators

5.4.2

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).

Displaying a summary status of NQS queues
5.4.3

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 a `cqstatl -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 a `cqstatl` command.

Section 5

```
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.

Section 5

Column name	Description
	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.

5.4.4 Displaying all the characteristics defined for an NQS queue

To display details of all the characteristics defined for NQS queues, use the `cqstatl -f` command.

You can restrict the display to specific queues, as follows:

`cqstatl -f` *queues*

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
<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>
        /opt/craysoft/nqe/bin/pipeclient CRI_DS

<ACCESS>
        Route:  Unrestricted            Users:  Unrestricted

<CUMULATIVE TIME>
        System Time:    269.92 secs     User Time:      95.21 secs
```

The `cqstatl` 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.

Section 5

Displaying summary status of user requests in NQS queues

5.4.5

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 ljgm     nqebatch@latte   10644  --- 262144     ** R
```

Section 5

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) man page for a list of the minor values.

Section 5

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     ljgm     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). For example, the `cqstatl` command can be used with the `-a` option, as follows:

```
cqstatl -a
```

Note: For information about displaying requests in the NQE database, see section 8, page 197.

Displaying the mid database
5.4.6

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 six systems are currently defined; the descriptions of these entries are explained in subsection 4.3, page 48.

The following `qmgr` command displays information on a specific *mid* or host:

`show mid` *hostname*|*mid*

Section 5

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
```

Controlling the availability of NQS queues

5.5

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.

Enabling and disabling queues

5.5.1

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 subsection 4.7, page 67).

Example of enabling and disabling queues

5.5.1.1

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
```

Starting and stopping queues

5.5.2

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 subsection 5.6.4, page 129.

Example of starting and stopping queues
5.5.2.1

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
```

Manipulating requests
5.6

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.

Moving requests between NQS queues
5.6.1

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 subsection 5.4.5, page 119). 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*

Example of moving requests between queues
5.6.1.1

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.

Moving requests within a queue
5.6.2

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 `first` or `next` 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.

Example of moving requests within a queue
5.6.2.1

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.

Holding and releasing requests
5.6.3

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 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.

Section 5

Example of holding and releasing requests
5.6.3.1

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
```

Suspending, resuming, and rerunning executing requests
5.6.4

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 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 `cqsub -nr` option.

Example of suspending and resuming requests
5.6.4.1

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`

Modifying the characteristics of requests in queues
5.6.5

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 (`rtime_limit` and `ptime_limit`).
Intraqueue priority	NQS intraqueue priority of a request. This has no effect on local queues (`priority_limit`).
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`).

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 systems and 2 bytes on UNIX systems.

Suffix	Units
`b`	Bytes (default)
`w`	Words
`kb`	Kilobytes (2^{10} bytes)
`kw`	Kilowords (2^{10} bytes)
`mb`	Megabytes (2^{20} bytes)
`mw`	Megawords (2^{20} bytes)

Suffix	Units
gb	Gigabytes (2^{30} bytes)
gw	Gigawords (2^{30} bytes)

Example of modifying a queued request
5.6.5.1

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
```

Deleting requests in NQS queues
5.6.6

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` command to delete requests, as follows:

`cqdel -u` *username requestids*

The *username* argument is the name of the user who submitted the request.

Example of deleting a queued request
5.6.6.1

To delete all requests waiting in the queue `bqueue15`, use the following `qmgr` command:

```
purge queue bqueue15
```

Deleting or signaling requests in remote NQS queues
5.6.7

After a request is in an NQS queue at a remote system, you can use the NQE GUI `Status` window or the `cqdel` 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 section 8, page 197, 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. If you use the `cqdel` command, see the `cqdel`(1) man page for more information about using the `cqdel` command to delete or signal requests on NQS.

NLB Administration [6]

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 subsection describes NLB administration. The following topics are discussed:

- Overview of NLB
- Overview of NLB configuration
- Editing NLB configuration file
- Starting the NLB server
- Specifying the NLB server location
- 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. (The same machine load information may be displayed by using the `cload` command.)

For information about the NQE database and NQE scheduler, see section 8, page 197.

NLB overview

6.1

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 the subsection 7.1, page 155.)

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 and `cload` program to indicate an NQE server is no longer talking to the NLB. It may also be used for destination selection.

Object type	Description
FTA_CONFIG and FTA_DOMAIN	FTA configuration objects. See subsection 12.2.8, page 319 for information on defining these objects.
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.

Overview of NLB configuration

6.2

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 subsection 7.2, page 156. 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 /etc/nqeinfo file.

Server database integrity

6.2.1

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 subsection. 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 subsection 6.7.5, page 148.)

The restart sequence, in combination with collectors, provides a high degree of consistency between NLB database contents and the information sent to the server.

Editing the NLB configuration file

6.3

The NLB configuration file, `config`, is a text file with 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.

KEY	Description of value
`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
```

Starting the NLB server

6.4

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.

Specifying the NLB server location

6.5

All programs that communicate with the NLB server (the collector, the NQE GUI `Status` function, and the `cload`, `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 `/etc/nqeinfo` file at installation
4. The `.Xdefaults` file

In each of these cases, the syntax is as follows:

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. 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"
```

The default value of *service_name* is `nlb`.

The service name is the string contained in the configuration file (see subsection 6.3, page 138) as a value for the keyword `PORT`.

If you place this 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.

If you have multiple servers, specify multiple *hostname* [:*service_name*] pairs.

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 subsection 11.2, page 296.

Starting NLB collectors

6.6

An NLB collector runs on each NQE master server and NQE execution server . 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 subsection 11.2, page 296.

A number of attributes are collected (see subsection 7.5, page 169). 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 `/nqe/etc` directory, which is `$NQE_ETC` as defined in the `nqeinfo` file, or the `/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.

Option	Description
`-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*	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`.

Option	Description
`-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. See subsection 7.3.5, page 161, for more information on averaging data.
`-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.

Managing the NLB server

6.7

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 subsections describe the `nlbconfig` command.

Specifying the server on the command line

6.7.1

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 subsection 6.5, page 141). 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 subsection 6.5, page 141, the NLB server will not start.

Verifying the NLB server is running

6.7.2

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.

Shutting down the server

6.7.3

To shut down the server, use the following command:

```
nlbconfig -kill
```

Rereading the `policies` file

6.7.4

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 located by default in `/usr/spool/nqe/nlbdir/policies`. Do not confuse this file with the `policies` file located in the `/usr/craysoft/nqe/etc` directory, which is used only upon NQE initialization. If you make changes to the policies file, make them to the file in `/usr/spool/nqe/nlbdir`.

Configuring name maps, objects, and ACLs
6.7.5

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 139 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 subsection 6.1, page 135):

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.

Option	Description
`-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.

Examples of configuring name maps and objects
6.7.6

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 subsection 6.5, page 141):

```
nlbconfig -odel -t C_OBJ -n gust
```

Changing ACL permissions
6.7.7

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`. To change the master ACL, edit the NLB `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.

By default, all users can monitor all requests (through the NQE GUI `Status` window) in the execution server complex. It may be better for your site to control access to NQS request data, for example, by allowing certain users to see all requests and limiting most users to seeing only their own requests. The following commands specify that only the owner of a request can see its status, and then the collector accounts (`luigi` and `mario`) are given delete (`d`), read (`r`), and update (`u`) privileges:

```
nlbconfig -s cool -aput -t NJS -u OWNER -a r
nlbconfig -s cool -aput -t NJS -u luigi -a dru
nlbconfig -s cool -aput -t NJS -u mario -a dru
```

In the preceding list of commands, the user `OWNER` is not a user account. Rather , it is a keyword that the NLB server uses to specify that any user can see his or her own objects, but others cannot.

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
```

To change the permissions so that users can see only their own requests, you can edit the final line (which allowed `WORLD` read privilege) so that `OWNER` has read privilege, as follows:

```
type          user          host          priv
----          ----          ----          ----
NJS           luigi         *             dru
NJS           root          *             dru
NJS           mario         *             dru
NJS           OWNER         *             r
```

Note: Editing this line does not actually delete the `WORLD` entry. To delete the `WORLD` entry, you must use the `-adel` option on the `nlbconfig` command.

The following `nlbconfig` command places the information in the server:

```
nlbconfig -aput -f acl_file
```

Implementing NLB Policies [7]

This subsection 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)

Defining load-balancing policies

7.1

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.

Section 7

The `policies` file

7.2

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 `/usr/spool/nqe/nlbdir/policies`. Do not confuse this file with the `policies` file located in the `/usr/craysoft/nqe/etc` directory, which is used only on NQE start-up (after an install) if `/usr/spool/nqe/nlbdir/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

Section 7

Operator	Description
<=	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 `qsub -la` option can also be used in policies.

Attributes provided at startup are described in Table 8, page 173. 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 subsection 7.8, page 194.

Example of how policies are applied

7.3

The following subsection describes how a policy for NQS load balancing is applied. For information about how to specify a policy name, see subsection 4.5.4, page 64.

Step 1: the name

7.3.1

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.

Step 2: the constraints

7.3.2

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 subsection 6.6, page 142). 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
```

Step 3: the sort
7.3.3

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.

Step 4: the count
7.3.4

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.

Approaches to designing policies: averaging data
7.3.5

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 subsection 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 the `cload` program) 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 10 and Figure 11.

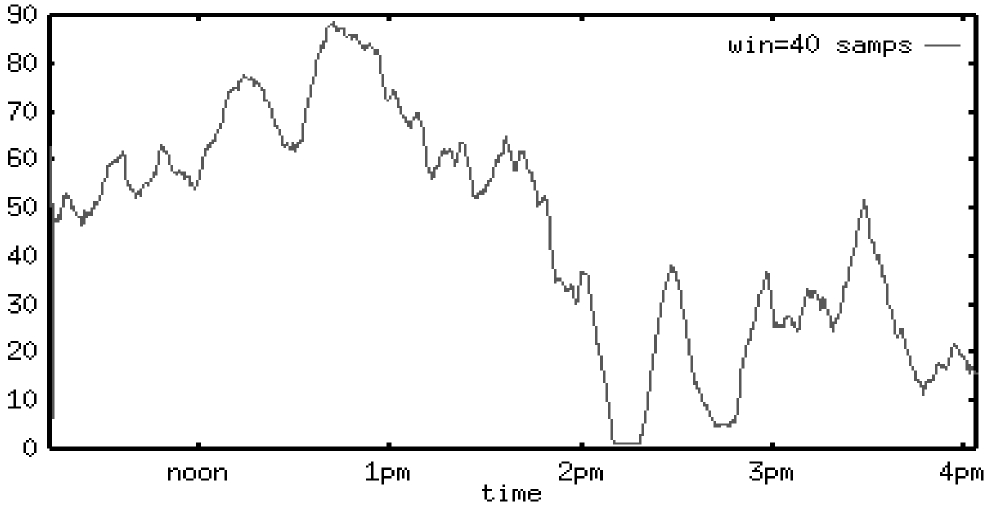

Figure 10. Average machine load data

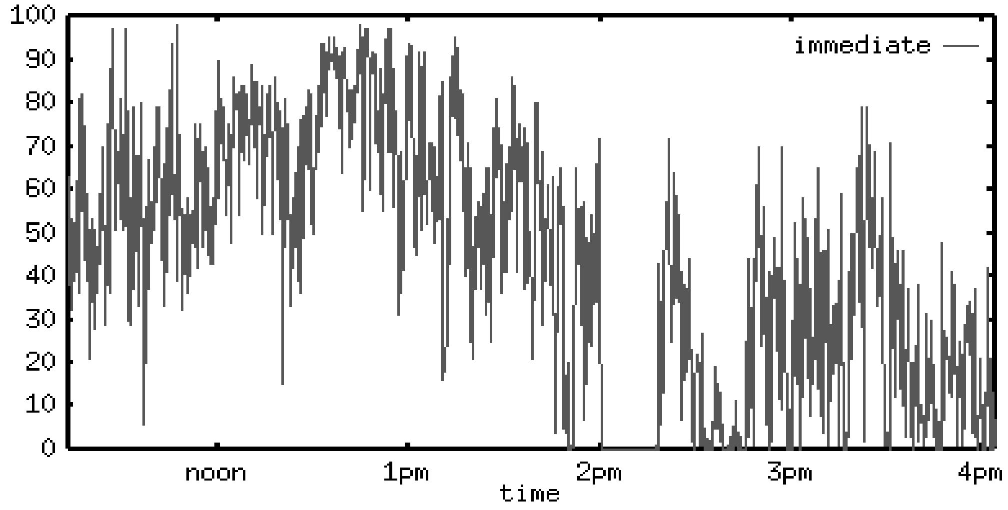

Figure 11. Instantaneous machine load data

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.

Other factors: weighting machines

7.3.6

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 subsection 7.3.9, page 166), 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.

7.3.7 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.

Policy examples
7.3.8

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 subsection 7.3.6, page 163).

```
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
```

7.3.9 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.

Section 7

Testing policies

7.4

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 [-c count ] [-d] [-s server ] [-p policy ] [-h host ]
```

The `nlbpolicy` command sends the server a policy query. The `-c` option controls the maximum number of hosts returned, the `-p` option selects a policy by name, and 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 definition 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).

File formats

7.5

The following two subsections describe the NLB name map and the object definition file formats.

The name map file

7.5.1

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);
```

(continued)

```
        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 vailidation 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 (and `cload`) displays and in policy definitions.

Section 7

Table 7 lists the allowed attribute types:

Table 7. 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 or the `cload` command). For an example of writing a load-balancing policy, see subsection 7.3.9, page 166.

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 8 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 8. 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

Section 7

Table 8. Reserved host object attributes (continued)

Attribute	Description	Type
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
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)

Table 8. Reserved host object attributes (continued)

Attribute	Description	Type
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

Section 7

Table 9 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 systems.

Table 9. Attribute implementation across platforms

Attribute	AIX	DEC OSF	HP-UX	SGI	Solaris	SunOS	UNICOS
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

Table 9. Attribute implementation across platforms (continued)

Attribute	AIX	DEC OSF	HP-UX	SGI	Solaris	SunOS	UNICOS
NLB_SWAPPING	Y	Y	Y	Y	Y	–	Y
NLB_SYSCPU	Y	Y	Y	Y	Y	Y	Y
NLB_RUNQOCC	Y	–	–	Y	Y	–	Y
NLB_PHYSMEM	Y	Y	Y	Y	Y	Y	Y
NLB_FREEMEM	Y	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	Y
NLB_SWAPFREE	Y	Y	Y	Y	Y	Y	Y
NLB_IOTOTAL	Y	–	–	Y	Y	–	Y
NLB_IOREQS	Y	–	Y	Y	Y	–	Y
NLB_SYSCALLS	Y	Y	Y	Y	Y	Y	Y
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 -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);
.
.
.
```

The object data file
7.5.2

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 definition file that is downloaded to the `nlbserver` with the following command:

`nlbserver -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”] ||\
  [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
```

Implementing NQS destination-selection policies

7.6

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 subsections 7.6.5 and 7.6.6, pages 185 and 186, show how sites might configure NQS destination-selection queues to determine how batch requests are routed.

Configuring NQS destination selection
7.6.1

The following steps comprise the configuration process for an NQS destination selection environment:

1. Start an NLB server on any system in the network. NQS does not need to be running on this system.

2. Define a pipe queue as a destination selection pipe queue on each system that will route requests using NLB (as described in subsection 7.6.5, page 185). 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 subsection 4.5.4, page 64.)

3. Run NLB collectors on each system that will accept batch requests from the NLB. By default, batch requests using destination-selection will be sent to the `nqebatch` queue on the selected host. To change this, see the `ccollect`(8) man page.

NQS user interface to NLB
7.6.2

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.

Policy definition
7.6.3

The following example could be used as a policy definition in the NLB `policies` file:

```
#       Policy for load-balancing between Cray systems
#       Cray hardware responding within last 5 minutes
#       Memory oversubscription of less than 4 - Ratio of memory
#       used (real and swap) divided by real memory
#       System CPU less than 20% per CPU for 1 or 2 CPU hosts
#       System CPU less than 3.5% for 3 or more hosts
#
#       Sorted by average percentage of idle cpu per cpu
#
policy: AGE
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_A_RUNQLEN * NLB_NCPUS
```

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) / 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
```

NQS processing
7.6.4

This subsection 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 subsection 7.7, page 188.

Example of load balancing using the default policy
7.6.5

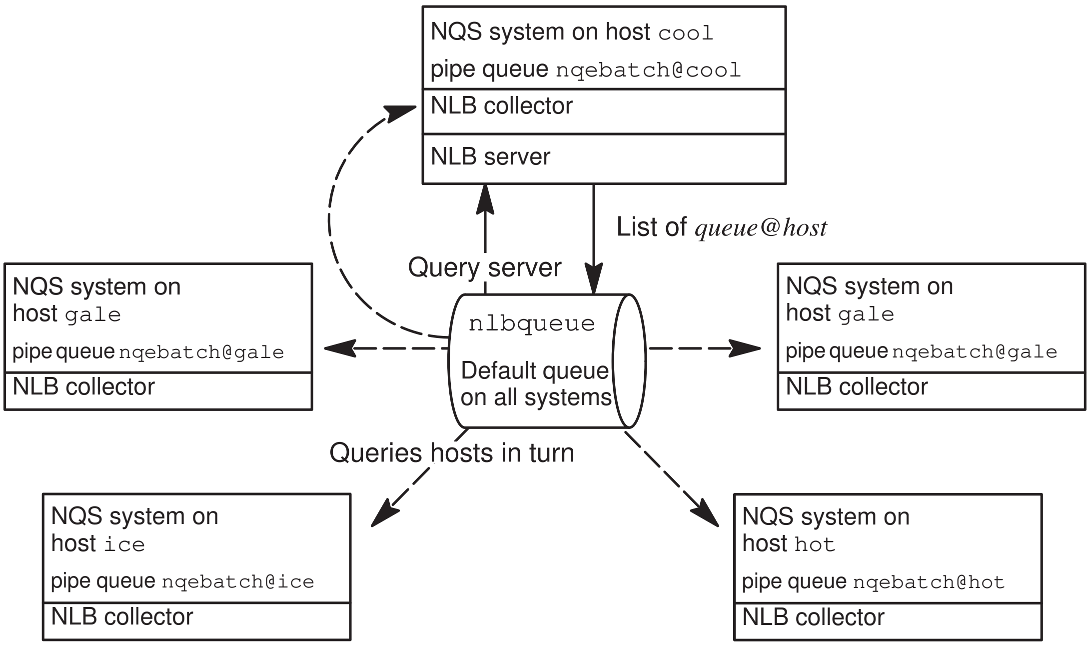

Figure 12. NQS configuration with load balancing by default

The following qmgr commands create the queue nlbq and define it as the default queue:

```
% qmgr
Qmgr: create pipe_queue nlbq server=(pipeclient CRI_DS) priority=33
Qmgr: set default batch_request queue nlbqueue
Qmgr: quit
```

Section 7

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.

7.6.6 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 13. One is a standard pipe queue, called standard. The second, called nlbqueue, is configured to perform destination selection.

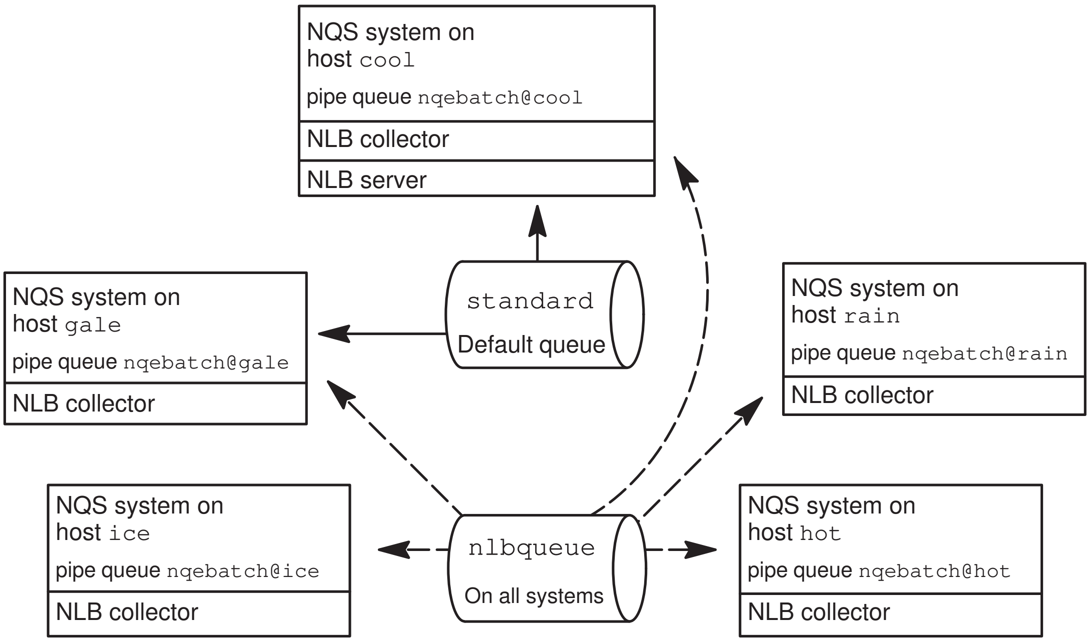

Figure 13. NQS configuration with load balancing by selection

The following qmgr commands create these queues and set the default queue to standard so that requests are routed by standard NQS routing. This queue explicitly defines a policy name, NLB host, and NLB port.

```
Qmgr: create pipe_queue standard server=(pipeclient) \
destination=(batch@cool,batch@gale) priority=30
Qmgr: create pipe_queue nlbq \
server=(pipeclient CRI_DS IOBOUND cool:nlb) priority=30
Qmgr: set default batch_request queue standard
```

Requests that are submitted to the queue nlbq are routed to the most lightly loaded system.

Using batch request limits in policies

7.7

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 10. 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 10. 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`

Section 7

7.7.1 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 subsection. 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 definition file and downloading this into the server.

Section 7

```
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.

7.7.2 Testing the policy

To test your policy, use the `nlbpolicy -a` option. Its syntax is as follows:

```
nlbpolicy -a attribute =value ,[attribute =value ]
```

You can specify any attribute described in Table 10, page 189, 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.

The output from the command when you use only the `-a` option is a list of *queue* @*hostname* values. For example, the following command tests which queues in the complex the NLB server would recommend if you had a request that required 1000 seconds of CPU time:

```
# nlbpolicy -p nqs -a JOB_PRCPUTIME=1000
nqebatch@ice
nqebatch@gust
nqebatch@storm
```

You can combine the `-a` with other `nlbpolicy` options to specify policies, the number of hosts or queues, a particular host, or a particular server.

Section 7

Section 7

7.8 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 definition file
- `nlbconfig` command
- `nqeinfo` file; which is the site NQE configuration file, and it is generated at installation time (see section 3, page 39)

The following steps are necesssary 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. Make the new attributes known to the NLB server by downloading the updated name map using the `nlbconfig` command (see subsection 6.7.5, page 148).

2. 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 definition file. This program or script will periodically update this object definition file with new values for the attributes that have been defined.

3. Instruct the NQE collector program to read from the new object definition file. This is done by using either the `-C` option of the `ccollect` program or by using the variable `NQE_CUSTOM_FILE_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. For syntax and additional information about how to tell the collector to read from a custom object file, see the `ccollect`(8) man page.

An additional feature, called *customized collector startup*, can help automate the generation of the extended object definition files. This feature allows the administrator to specify one or more programs that will be automatically started by the NQE collector. The program can be either an executable file or a script file. Each program can have multiple command-line arguments. The NQE collector detects when any of the customized collectors has terminated and will restart it during its next cycle.

To specify which customized collectors are to be started by the NQE collector , use the variable `NQE_CUSTOM_COLLECTOR_LIST` either in the `nqeinfo` file or as an environment variable. For the syntax of this variable, see the `ccollect`(8) man page.

NQE Database [8]

This section provides basic information about the NQE database and its components. This section includes the following information:

- Why is the NQE database needed?
- What is it?
- NQE database components
- NQE database concepts and terms
- Starting the NQE database
- Stopping the NQE database
- Obtaining status
- Security
- Configuring the NQE database components

Section 9, page 229, describes the NQE scheduler in detail and including a tutorial for writing a simple NQE scheduler.

Section 8

Why is the NQE database needed?

8.1

In certain environments, using NLB policies to determine the destination server for a job does not provide sufficient flexibility in scheduling. For example, the following types of scheduling are not possible with NLB:

- 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.

What is it?

8.2

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 14 shows an example of the layout of NQE database. Jobs are submitted using client commands from client machines. The jobs are placed in the NQE database on one machine (the master server). A scheduler program determines which machine should run the job. In the example shown in Figure 14, the job is sent to execution server A where it is submitted and run in NQS.

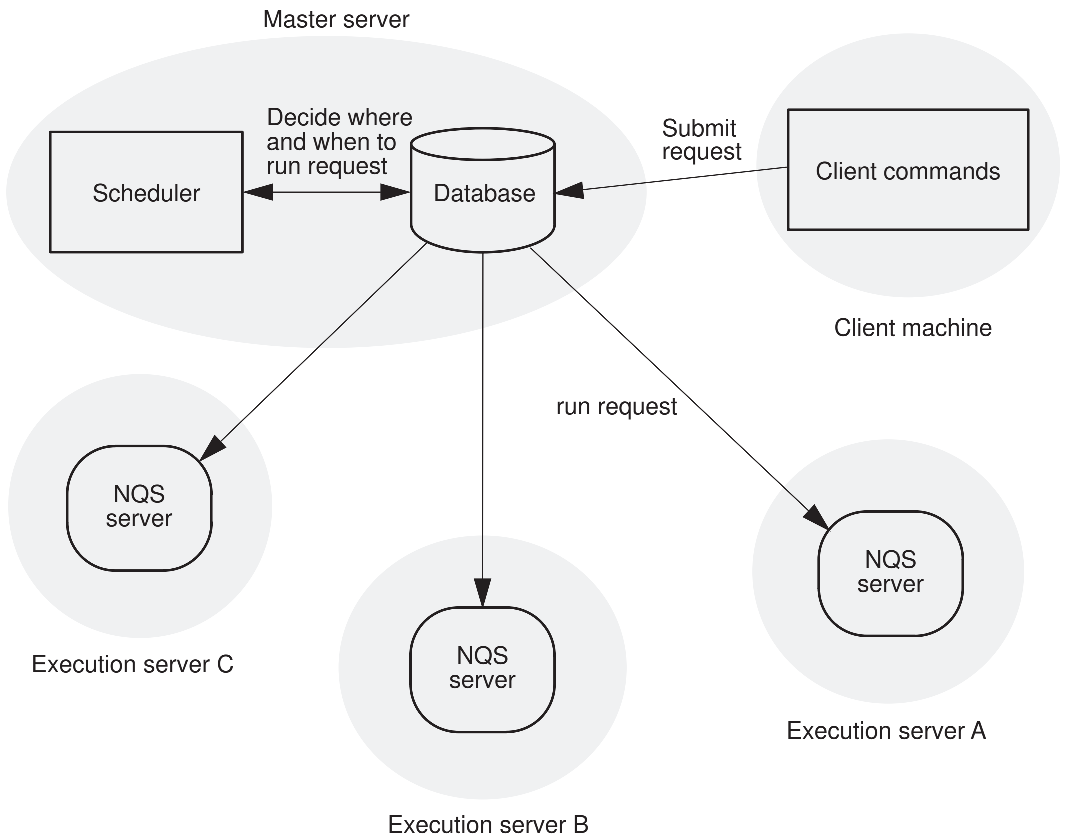

Figure 14. NQE database layout

NQE database components

8.3

Figure 15 shows in greater detail the processes and components of the NQE database. The components are described in the following subsections.

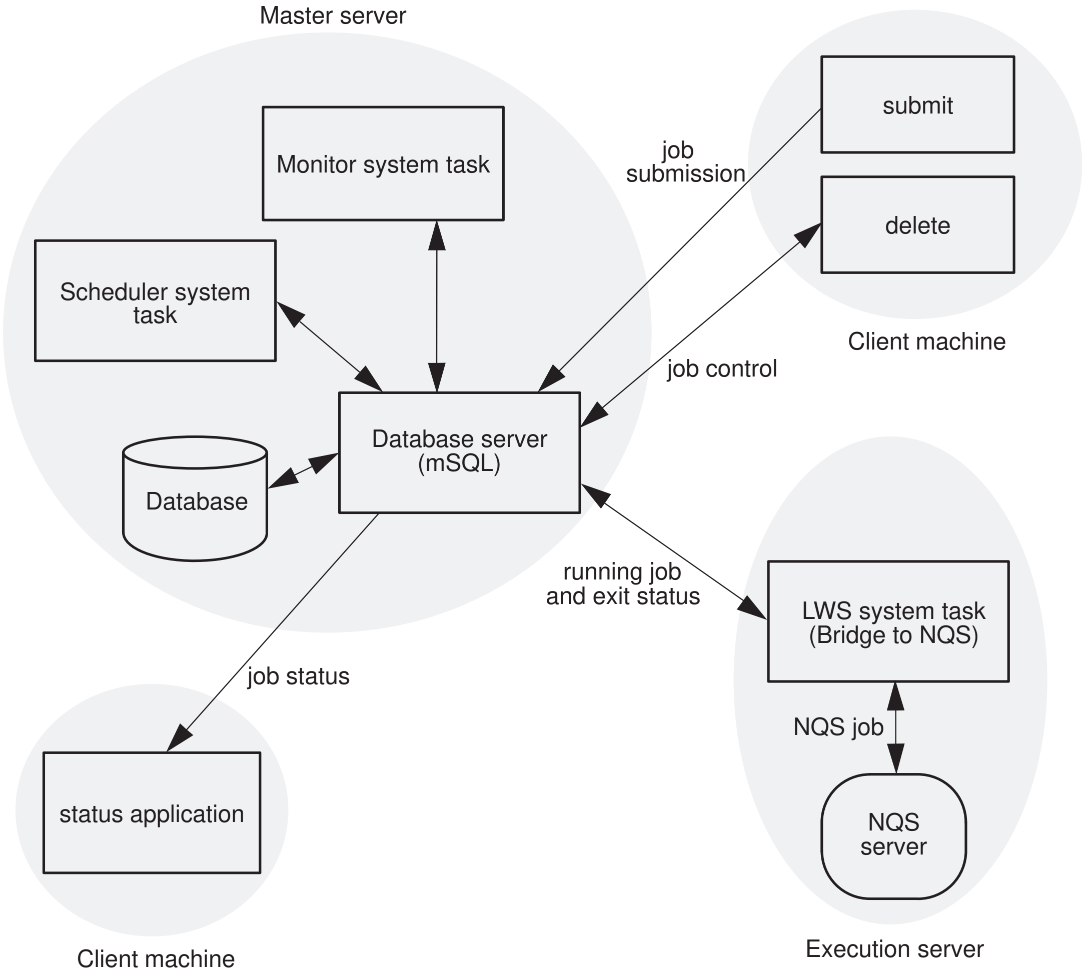

Figure 15. NQE database components

Database server (mSQL)

8.3.1

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

Monitor system task

8.3.2

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.

Scheduler system task

8.3.3

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

- It keeps track of light-weight 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 section 9, page 229).

LWS system task (bridge to NQS)

8.3.4

The light-weight server (LWS) system task 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.

Client commands

8.3.5

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: By using 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, 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: By using 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.

- By using 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 subsections.

Section 8

NQE database concepts and terms

8.4

This subsection describes the following NQE database terms and concepts:

- Objects
- Task owners
- Task states
- Events

Objects

8.4.1

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.

Attribute name	Description
`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 22, page 268.
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 11, page 209. For example, a user may insert a `SIGDEL` event targeted at a particular task object. (This is equivalent to performing a `cqdel` command.) 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.

Section 8

Object type (and abbreviation)	Description
	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 etc. For a list of configuration attributes, see subsection Table 15, page 222.

Task owners
8.4.2

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. Three classes 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. A LWS's default name is the host name 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.

Task states

8.4.3

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 16 shows some ownership changes and state changes possible for a user task.

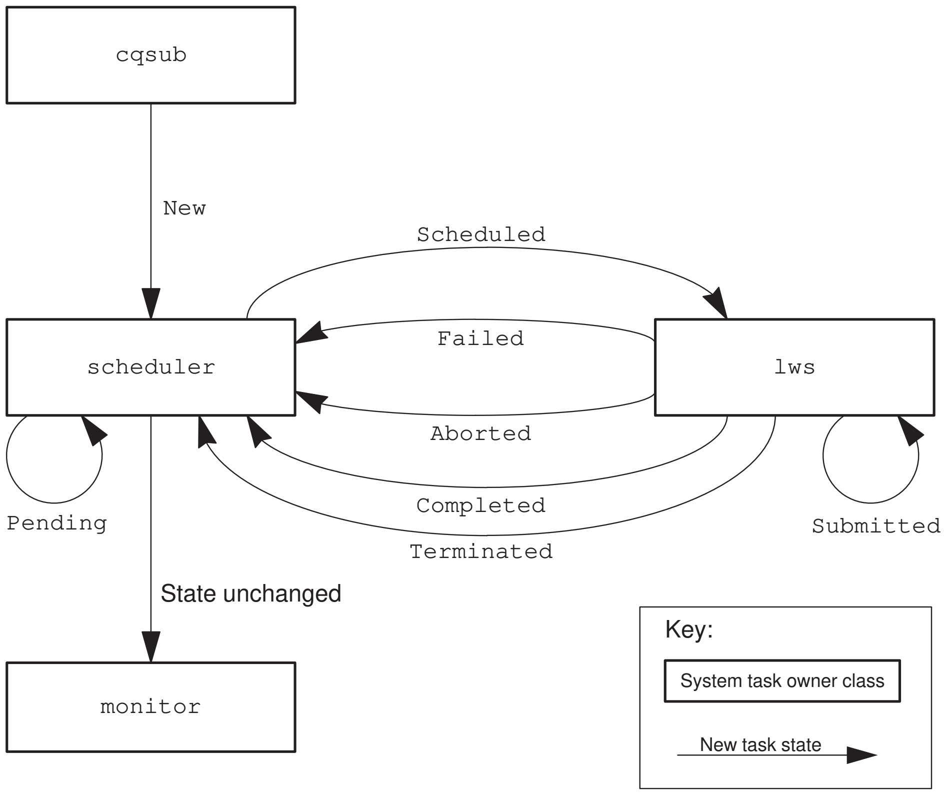

Figure 16. User task owners and states

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.)

Section 8

4. When scheduled to run, `scheduler.main` assigns the task to one of the `lws` system tasks with state `Scheduled`. There is, in general, one `lws` system task for each node in the network. 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. On 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.

Events
8.4.4

Events form the mechanism for communication between 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 tasks 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) 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 11 lists the standard events:

Table 11. 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.

Table 11. Standard events (continued)

Event name	Usual target	Description
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.

Starting the NQE database

8.5

During installation, one master server should have been defined to run the mSQL database server. If this has not been done, simply ensure that on one master server, the `MSQL_SERVER` `nqeinfo` file variable is set to the name of the local machine. `MSQL_SERVER` should have the same value across all master servers, execution servers, and clients in the network.

On a master server and execution server, `nqeinit` always starts the LWS daemon.

On the master server designated the NQE database server, the following UNIX processes are also started:

- mSQL database server, `msqld`
- Monitor daemon
- Scheduler daemon

Note: No startup is required on NQE clients.

The `nqedbmgr` utility allows finer control over process startup. Table 12 describes commands available for this control; these commands are also described on the `nqedbmgr`(8) man page:

Table 12. `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.	`db create database`

Section 8

Table 12. nqedbmgr process startup control commands (continued)

Command	Description	Example
	You must be root and be on the NQE database server node to perform this command.	
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
create systask *name* [*tclarrayname*]	Creates 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

Section 8

Table 12. `nqedbmgr` process startup control commands (continued)

Command	Description	Example

`start` *systask*	Starts up the system task.
	
	This command starts the named system task process. On the `MSQL_SERVER`, `nqeinit` will start a `monitor`, `scheduler`, and `lws` process. On other NQE servers, `nqeinit` will start and create the system task object on the `MSQL_SERVER`.
		`start lws`
		`start monitor`
		`start scheduler`

The `nqeinit` startup script uses an `nqedbmgr` Tcl input script (`nqedb_config.tcl`) to perform all the startup operations. See that input script for an example 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.

Stopping the NQE database

8.6

The nqestop script stops all system tasks and shuts down the NQE database.

The nqedbmgr utility allows finer control over process shutdown. Table 13 describes commands available for this control:

Table 13. 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 a nqedbmgr Tcl input script (nqedb_stop.tcl) to perform all the shutdown operations. See that input script for an example of how the individual components are stopped.

Section 8

Obtaining status

8.7

The nqedbmgr utility provides commands for determining the status of the NQE database and examining data in the database. Table 14 describes commands available to obtain status; these commands are also described on the nqedbmgr(8) man page:

Table 14. nqedbmgr status commands

Command	Description	Example
status [all|db]	Provides a status of the components of NQE database. status by itself displays which system tasks are running, their PIDs, and host on which they are running. status db obtains version information from the mSQL server. status all displays everything.	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 man page for details.	select –f t

Section 8

Security

8.8

Authorization is performed at the following points:

- Connection: On 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 subsections.

Connection

8.8.1

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 `/etc/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 (`dbusers` 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 disallow the dbuser explicitly. 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, by default, using a `dbuser` of `root`.

Section 8

Section 8

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, using the cqsub, cqstatl, cqdel, and NQE GUI (nqe) 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 connect using their UNIX names as the `dbuser` name, except `bob` from `badclient` (`bob` can connect from anywhere except from `badclient`) and except `amy`, who cannot connect from any host.

Target

8.8.2

The administrator may define how the LWS validates jobs from the NQE database before submitting them to NQS locally. Valid values are one of the following:

- `FILE`. The LWS checks for a file called `~tuser/.nqshosts` or `~tuser/.rhosts`. Within the file will be a list containing which client users from which client hosts are allowed or disallowed access.
- `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
```

Section 8

Events
8.8.3

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 21 lists the predefined events you may want to send.

A `dbuser` connected with the privilege type of `user` may only send events to jobs created by the same `dbuser`. Those connected with the privilege type of `system` may send events to any object, including system tasks.

Status
8.8.4

A `dbuser` connected with `system` privilege may read any object in the database. That is, any task may be examined.

A dbuser connect with `user` privilege is restricted to using the client interfaces (such as `cqsub`, `cqstatl`, `cqdel`, and the NQE GUI (`nqe`) 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}
```

Configuring the database components

8.9

Various global NQE database and scheduling configuration attributes can be set and/or updated using the following `nqedbmgr` command syntax:

```
update config_obj {attribname attribvalue ...}
```

For example:

```
update g1 {status.access all}
```

System tasks must be re-initialized or restarted in order to read any updated global configuration attributes.

To re-initialize a system task, such as the `scheduler`, enter the `nqedbmgr` command that inserts a `TINIT event` object into the database, which is read by the scheduler and acknowledged:

```
ask scheduler tinit
```

The following subsections describe the predefined global NQE database and scheduling configuration, NQE scheduler, and LWS attributes that you can use.

Section 8

Section 8

8.9.1 Global NQE database and scheduling configuration attributes

Table 15 lists the predefined global NQE database and scheduling configuration attributes (this table is also defined on the `nqedbmgr`(8) man page):

Table 15. `nqedbmgr` predefined global NQE database and scheduling 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 with 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`

Table 15. nqedbmgr predefined global NQE database and scheduling configuration attributes (continued)

Attribute name	Description	Default if not specified
`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` Allow user to select summary information from all tasks `own` Allow user to select only tasks submitted by the user. A user may only see full information for tasks submitted by that user.	`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.	`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`

Section 8

Table 15. nqedbmgr predefined global NQE database and scheduling configuration attributes (continued)

Attribute name	Description	Default if not specified
sys.st_rate	Rate at which system tasks check for new tasks, changes in task state etc. This attribute may be a mathematical expression. Units are seconds.	10
sys.upd_rate	Rate at which LWS system tasks checks for messages from NQS. This attribute may be a mathematical expression. Units are seconds.	10

Scheduler attributes

8.9.2

The default scheduler file is situated in the NQE bin directory and is called local_sched.tcl. The default scheduler has the following properties:

1. The number of tasks that may run on an LWS can be limited.
2. The number of tasks per user that may run on an LWS can be limited.
3. LWS selection can be performed by a simple round-robin or by using an NLB policy (passing memory and CPU requirements).
4. Tasks are ordered by time in.
5. 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.
6. A simple cluster rerun facility is implemented.

The default scheduler can interpret the following predefined attributes. Table 16 lists these attributes (this table is also defined on the nqedbmgr(8) man page).

Table 16. nqedbmgr pre-defined 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 job to the scheduler without completing) before 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 User-scheduler file. The File name may include absolute path name or is 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

Section 8

Table 16. nqedbmgr pre-defined scheduler attributes (continued)

Attribute name	Description	Default if not specified
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

LWS attributes
8.9.3

Table 17 lists the predefined LWS attributes (this table is also defined on the nqedbmgr(8) man page):

Table 17. nqedbmgr predefined LWS attributes

Attribute Name	Description	Default if not specified
default_queue	The default NQS queue name 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

Table 17. nqedbmgr predefined LWS attributes (continued)

Attribute Name	Description	Default if not specified
run_mechanism	Method by which LWS checks the authorization for submitting new jobs to NQS. The mechanisms available are: File Use NQS file validation. Passwd Use NQS password validation.	File

Section 8

Writing an NQE Scheduler [9]

The scheduling algorithm used by the NQE scheduler is designed to be easily modified. Also, it is possible to completely rewrite the scheduler. This section describes the NQE database and scheduler in greater detail than section 8, 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 re-started 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 section contains the following information:

- Scheduler system task
- Writing a simple scheduler
- Attribute names and `nqeinfo` file variable names
- Tcl built-in functions

The scheduler system task

9.1

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 17 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.

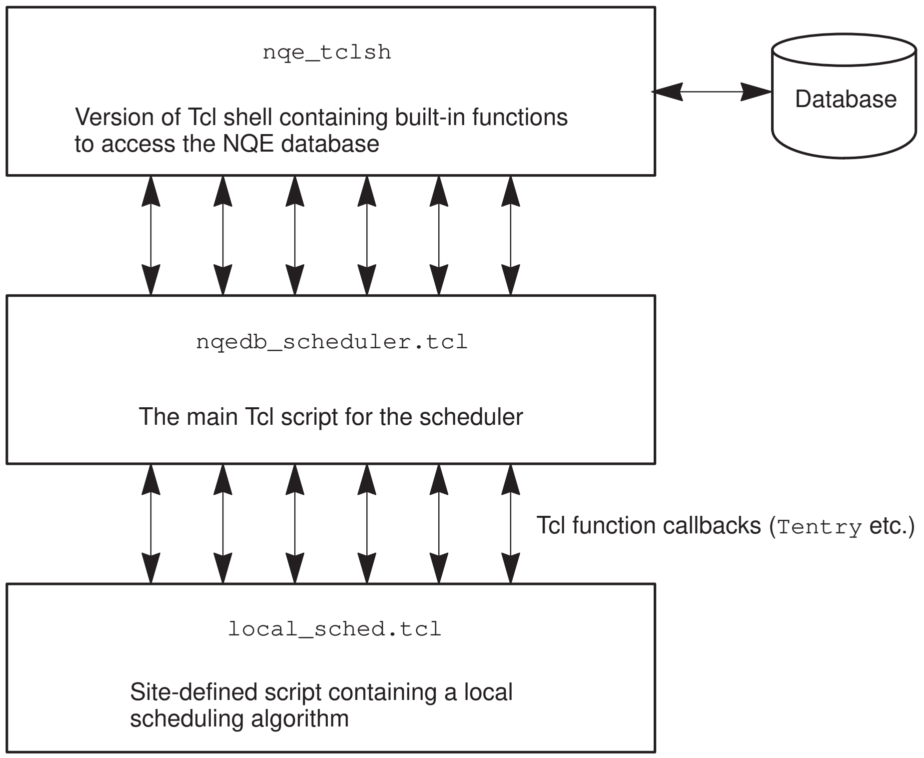

Figure 17. Scheduler program structure

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 subsection 9.2, page 232, for an example of how to do this.

Writing a simple scheduler

9.2

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 subsection provides a tutorial in writing an NQE scheduler.

Requirements

9.2.1

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 master server machine running the mSQL daemon and NQE scheduler system task. Only one master server machine 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 daemon, check that the `nqeinfo` file variable `NQE_TYPE` is set to `MASTER_SERVER` 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 and NQE scheduler must be running.

  `nqeinit` process should start these system by default. To check that system tasks are running, perform 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 master server named `latte` with another server named `pendulum`:

Section 9

```
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
```

Section 9

Goal
9.2.2

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 execution 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 subsections:

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
9. Add configuration enhancements to the `Tinit` callback function

Step 1: Create a new scheduler system task object

9.2.3

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 may vary depending on 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` should never be issued while the `simple` scheduler is still running.

For additional information about the `nqedbmgr shutdown` command, see subsection 8.6, page 214.

Step 2: Create the simple scheduler file

9.2.4

The following local scheduler files are provided with NQE:

- template_sched.tcl (in the craysoft/nqe/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 (in the craysoft/nqe/bin directory). This is the default scheduler supplied. (For more information about the default scheduler , see subsection 8.9.2, page 224.)
- simple_sched.tcl (in the craysoft/nqe/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:

- `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 describe their purpose.

Step 3: Write a `Tinit` callback function

9.2.5

Write a `Tinit` callback function.

The `Tinit` callback is called when the following occurs:

- When the scheduler starts up.
- Whenever 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
```

- Upon reconnecting to the NQE database after a previous unexpected disconnection.

Section 9

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 wit a value of 0, and the scheduler will shut 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 subsection 9.3.5, page 280. 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 pre-defined attributes, see subsection 9.3.4, page 277.

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`:

Section 9

```
#
# 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

}
```

Section 9

9.2.6 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:

- Whenever 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.
- Whenever a user task is completed or fails on an LWS.

The `Tentry` callback should be used to update local lists and counts, depending on 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 18 lists the states that must be recognized by the `Tentry` callback:

Table 18. 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. This state is detected by the LWS system task 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 state is never associated to the scheduler and does not need to be trapped.

Section 9

Table 18. Mandatory task states recognized by the `Tentry` callback (continued)

Task state	Description
`Completed`	Indicates that the job request has completed successfully .
`Failed`	Indicates that the job request cannot be submitted to NQS.
`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 a user task has been terminated by using 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
         }
        }
```

Section 9

```
         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_$objid`. Each attribute of each user task object owned by the scheduler is represented in Tcl by an array with a name comprised of `Obj_$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 the Tcl function `upvar #0`. See standard Tcl documentation for a full description of this function.

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.

Section 9

Tcl function	Description
`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 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.

9.2.7 Step 5: Write a `Tpass` callback function

Write the `Tpass` scheduling algorithm.

The `Tpass` callback is called when the following occurs:

- 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 on when the state of any LWS changes. This means that the scheduler may not react to the start up 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)
       }
```

Section 9

```
        # 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)
}
```

Section 9

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:

Section 9

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 LWS by round-robin (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]] \
      ]

}
```

Section 9

Step 6: Write a `Tnewowner` callback function

9.2.8

Write the `Tnewowner` callback function.

The `Tnewowner` callback is called when the following occurs:

- 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"
```

Section 9

```
        # 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
}
```

Section 9

The following additional functions were introduced in this subsection (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*	Replaces an element in 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.

9.2.9 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 { [lsesearch -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
        }
```

Section 9

```
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.

Step 8: Run the new scheduler
9.2.10

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 simple
   ```

   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.

Section 9

Before invoking the scheduler, set a trace level so that scheduling trace messages are reported; this will aid debugging. The environment variable settings are as follows:

```
ksh: export NQEDB_TRACE_LEVEL=sched
csh: setenv NQEDB_TRACE_LEVEL sched
```

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.

Tcl errors
9.2.10.1

It is very possible that the Tcl interpreter will encounter syntax errors while a local schedule is being developed.

All Tcl errors are trapped. If the error was due to a connection failure 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 in Tcl is the ability to quickly correct this type of problem.

Section 9

Trace levels
9.2.10.2

As shown in previous subsections, it is possible to increase the amount of information logged by the scheduler by setting the `NQEDB_TRACE_LEVEL` environment variable. Its value is a mask of the types of tracing required. 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.

Trace-level name	Usage
`gen`	General system task tracing.
`db`	NQE database function tracing.
`0`	Disable tracing.

Values may be separated by commas. For example, to set `sched` and `gen` tracing, use the following command:

```
ksh: export NQEDB_TRACE_LEVEL=sched,gen
csh: setenv NQEDB_TRACE_LEVEL sched,gen
```

You can change the trace level while the scheduler is running by sending a `TRACE_LEVEL` event to the scheduler. For example, to set the trace level to `sched` on the simple scheduler as it runs, use the following command:

```
nqedbmgr ask scheduler.simple trace_level sched
```

9.2.10.3 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.

Section 9

Step 9: Add configuration enhancements

9.2.11

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:\
         $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.

Section 9

Tracking down problems

9.2.12

This subsection describes some common errors you may experience and suggested solutions.

`WARN: Callback Tpass is not defined. No regular scheduling passes will occur`

Your local scheduler has not registered a `Tpass` function.

`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:

1. 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.
2. The mSQL server is not running; check that the `msqld` process is running and that the `nqedbmgr` command `status db` executes successfully .
3. Connection was refused by the mSQL server; check the `msqld` log file. It may give a security authorization reason for the connection failure.

`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.

Section 9

```
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.

```
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 follows 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 `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 sxx 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.

Section 9

`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.

`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.

9.3 Attribute names and `nqeinfo` file variable names

This subsection lists and describes all predefined attribute names for the `nqeinfo` file and for NQE database objects.

9.3.1 nqeinfo variables

Table 19 describes variables that may be defined in the `nqeinfo` file. The table also indicates its default value if it has one.

Section 9

Table 19. 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, it is no longer required.	Default value is $NQE_BIN/nqedb.struct. Needed by master 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. Needed by master server only.
MSQL_HOME	mSQL database files location. 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 master server only.

Section 9

Table 19. Variables that may be defined in the nqeinfo file (continued)

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 which 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 the environment variable MSQL_SERVER. Needed by all NQE machines.

Section 9

Table 19. Variables that may be defined in the nqeinfo file (continued)

Name of variable	Description	Notes
NQE_SEC_RUN_ MECHANISM	Type of validation performed by LWS. This variable defines the default type of validation of client to target user to be performed by the LWS when submitting a request to NQS. Valid types are as follows: File: LWS checks for local ~/.nqshosts or~/.rhosts file. Passwd: LWS checks for target password.	Default is File. This variable is overridden by a value set in the run_mechanism attribute in the LWS's system task object. Used by master and execution servers.

Section 9

Table 19. Variables that may be defined in the nqeinfo file (continued)

Name of variable	Description	Notes
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 master and execution servers.
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 master server only.

Section 9

Event objects
9.3.2

The event object type is `e`. Event objects are used to communicate among tasks (both system and user). Attributes are described in Table 20:

Table 20. Event objects attributes

Name of attribute	Description	Notes
`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 21 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 of this event. It may be a user task or system task ID.	

Section 9

Table 20. Event objects attributes (continued)

Name of attribute	Description	Notes
`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`*nnn* where *nnn* 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 subsection 8.8, page 216.
`timein`	Time object was inserted into NQE database.	
`timeup`	Time object was last updated.	
`type`	Object type (`e`)	

Section 9

Table 21 lists the events that are defined by default:

Table 21. 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 it. This event is posted by qdel/cqdel.
SHUTDOWN	None	Shutdown 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. Its principle use of this event is to restart the scheduler.

User task attributes

9.3.3

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 22:

Table 22. User task object attributes

Name of attribute	Description	Notes
`att.account`	Account 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`	Output file specification name	`cqsub` option: `-o`
`file.2.name`	Standard error output file name	`cqsub` option: `-e`
`file.3.name`	Job log output file name	`cqsub` option: `-j`
`file.eo`	Merge error and output files	`cqsub` option: `-eo`
`file.jo`	Job log control (`-J m|n|y`)	`cqsub` option: `-J`
`file.ke`	Keep standard error output 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`

Section 9

Table 22. User task object attributes (continued)

Name of attribute	Description	Notes
`file.kj`	Keep job log output 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 on execution machine. The option is set if *value* is 1 and is not set if *value* is 0 or attribute is not set.the	`cqsub` option: `-ko`
`id`	Unique ID for the task. In mSQL, this will be of the form: `t`*nnn* where *nnn* 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_p`	Per -request MPP processor elements.	`cqsub` option: `-l mpp_p`
`lim.mpp_t`	Per -request T3D MPP time limit.	`cqsub` option: `-l mpp_t`
`lim.p_mpp_t`	Per -process T3D MPP time limit	`cqsub` option: `-l p_mpp_t`
`lim.ppcpu`	Per -process CPU time limit	`cqsub` option: `-lt`

Section 9

Table 22. User task object attributes (continued)

Name of attribute	Description	Notes
`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`
`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`

Section 9

Table 22. User task object attributes (continued)

Name of attribute	Description	Notes
`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`

Table 22. User task object attributes (continued)

Name of attribute	Description	Notes
`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.user`	Send mail to stated user.	`cqsub` option: `-mu`
`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`
`nqs.export`	Export environmental 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`

Section 9

Table 22. User task object attributes (continued)

Name of attribute	Description	Notes
`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.	`cqsub` option: `-s`
`sec.ins.clienthost`	Client host name.	
`sec.ins.clientuser`	Client UNIX user name.	
`sec.ins.dbuser`	NQE database user name of originator of this task.	See subsection 8.8, page 216.
`sec.ins.hash`	Hash value for `sec.ins.*` values.	
`sec.ins.tuser`	Target user specification.	See subsection 8.8, page 216.

Section 9

Table 22. User task object attributes (continued)

Name of attribute	Description	Notes
`sec.ins.tuser.pwe`	Target user specification (encrypted).	See subsection 8.8, page 216.
`sec.target.user`	Target user for a particular LWS.	This attribute is created by the lws after submitting the task locally.
`script.0`	Script file contents.	This may be a very long attribute.
`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.	

Section 9

Table 22. User task object attributes (continued)

Name of attribute	Description	Notes
`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 any failures 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`.	may be `""`.
`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.	It is unset when task has not yet been submitted to NQS.
`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.	

Table 22. User task object attributes (continued)

Name of attribute	Description	Notes
`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 which 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 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`

Section 9

System task object

9.3.4

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 execution server. Attributes are described in Table 23:

Table 23. System task object attributes

Name of attribute	Description	Notes
`default_queue`	Provides the default NQS queue name to which jobs are submitted.	Used by the LWS only.
`id`	Unique ID for the object. In mSQL, this will be of the form `s`*nnn* where *nnn* 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.	
`run_mechanism`	Defines the validation mechanism used to check the target user against the client user for local submission to NQS.	Used by the LWS only. Default is `File` `NQE_SEC_RUN_MECHANISM` variable. See Table 19, page 261.

Section 9

Table 23. System task object attributes (continued)

Name of attribute	Description	Notes
s.class	The class name of this system task. Possible values at present are as follows: lws (LWS system task) There may be >1 instances of these. scheduler (Scheduler system task) monitor (The 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, by default, the host name for the LWS.	
s.pid	UNIX PID of system task.	Set to −1 when system task exits.
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 page 236 for usage.

Section 9

Table 23. System task object attributes (continued)

Name of attribute	Description	Notes
`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 subsection 8.8, page 216.

Global config object

9.3.5

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 24:

Table 24. 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 `e`*nnn* where *nnn* is an integer.	Allocated when the object is first inserted into the NQE database.

Section 9

Table 24. Global configuration object attributes (continued)

Name of attribute	Description	Notes
`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 subsection 8.8, page 216.

Section 9

Table 24. Global configuration object attributes (continued)

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`.

Table 24. Global configuration object attributes (continued)

Name of attribute	Description	Notes
`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.	

Tcl built-in functions

9.4

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 25 provides a brief description of these functions.

Table 25. Tcl built-in functions

Tcl built-in function	Description

`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 an event-driven characteristics.
`nqe_curtime`	Returns the elapsed seconds since UNIX Epoch.

Section 9

Table 25. Tcl built-in functions (continued)

Tcl built-in function	Description
nqe_strtime	Convert UNIX Epoch to a human-readable string.
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	Return effective/real UIDs and GIDs.
nqe_gethostname	Return local host name.

Section 9

Table 26 lists other useful functions implemented in Tcl:

Table 26. Additional functions implemented in Tcl

Tcl built-in 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*	Replace 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*]	Log 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 value.

Section 9

Table 26. Additional functions implemented in Tcl (continued)

Tcl built-in function	Description
`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. There is no return value.
`sched_set_callbac` *callback* *functionname*	Register 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.

Section 9

Table 26. Additional functions implemented in Tcl (continued)

Tcl built-in function	Description
	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.

Section 9

Using `csuspend` [10]

NQE lets you put to use the unused cycles on servers generally reserved for interactive use. This process is sometimes called *cycle stealing*. This section 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

`csuspend` options

10.1

The syntax of the `csuspend` command is as follows:

```
csuspend [-i value ] [-o value ] [-s] [-t interval ]
```

Section 10

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.
`-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.
`-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.

`csuspend` functionality

10.2

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

Section 10

below the thresholds specified, batch jobs are resumed. 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.

10.3 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  1 00:07:28 - auto suspend processing cycle initiated
  Mar  1 00:07:28 - input character rate=4, output character rate=0
  Mar  1 00:07:29 - found 2 pipe queues to process
  Mar  1 00:07:29 - processing pipe queue nqenlb
  Mar  1 00:07:30 - processing pipe queue gate
  Mar  1 00:07:31 - found 1 batch queues to process
  Mar  1 00:07:31 - processing batch queue nqebatch
```

Job Dependency Administration [11]

This section 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

Managing the NLB database for job dependency

11.1

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
```

Multiple NLB servers and job dependency

11.2

A potential problem exists when you use job dependency and multiple NLB servers. This subsection 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 NQE master or execution server 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:

```
#
# 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).
- If you have an NFS environment, you can create a cevent script. In the script, before invoking the actual cevent command for a given platform, explicitly set the NLB_SERVER environment variable to be one host. Then invoke cevent on behalf of the user. This guarantees that cevent will use the NLB server that you specify in the script.

FTA Administration [12]

Section 12

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 section provides the following information about managing the FTA:

- FTA overview
- Configuration
- Administration

FTA overview

12.1

The FTA facility consists of the three components shown in Table 27.

Table 27. File transfer components

Component	Program	Location of description
User interface	`ftua`(1) and `rft`(1)	Subsection 12.1.1, page 300
Common services	`fta`(8)	Subsection 12.1.2, page 300
File transfer services	`ftad`(8) and `nqsfts`(8)	Subsection 12.1.3, page 301

The three-tier stack of file transfer components consists of the file transfer user interface, the File Transfer Agent (FTA), 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 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.

User interface

12.1.1

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.

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.

Common services

12.1.2

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.

The `fta`(8) man page contains complete reference information.

File transfer services
12.1.3

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` 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 18 shows the relationships among the FTA components.

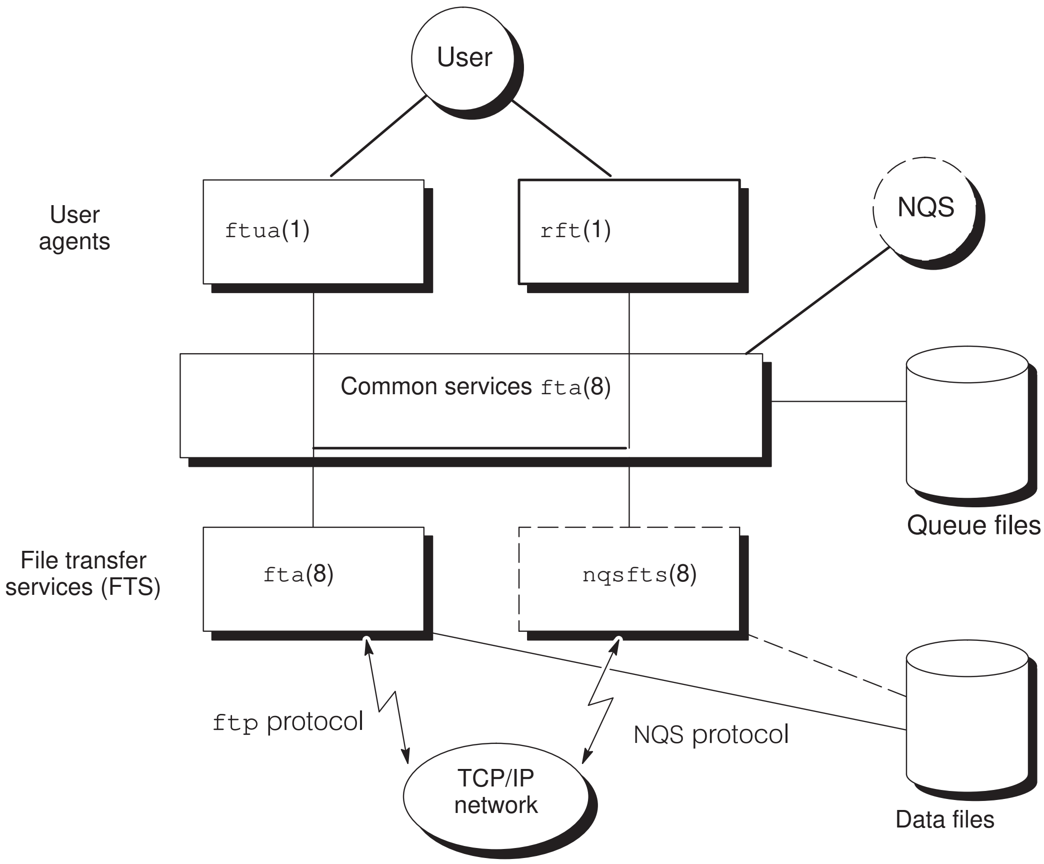

Figure 18. Component relationships in FTA

Configuration

12.2

FTA is configured in the Network Load Balancer (NLB) during NQE installation. This FTA configuration applies to all hosts in the NQE except UNICOS hosts. The NPPA S-keys must be configured on each separate system. For information about configuring S-keys, see subsection 12.2.6.1, page 315.

Note: FTA must be configured separately on each UNICOS host. For further information, see subsection 12.2.3, page 311.

FTA configuration information is contained in the NLB database. Two FTA object types exist in the NLB database: `FTA_CONFIG` and `FTA_DOMAIN`.

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 subsection 12.3, page 321.

This subsection explains how to view and change the FTA configuration.

Viewing the FTA configuration

12.2.1

FTA configuration is contained in the NLB. FTA configuration keywords map to attributes in the NLB object files.

FTA configuration keywords are `FTA` or `FTS`. `FTA` defines configuration parameters for FTA, and `FTS` defines configuration parameters for domains. Each `FTS` domain name must be unique.

The FTA keywords map to the attributes in the NLB object files, as follows:

Keyword	Attribute type
`FTA`	`FTA_CONFIG`
`FTS`	`FTA_DOMAIN`

FTA configuration may be viewed by using the the `nlbconfig` command.

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: The `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 declarer is always required. You can change this parameter in the `/etc/nqeinfo` configuration file.

Section 12

Parameter	Description
`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.

Section 12

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 (`FTS`) configuration parameters depends on the `FTA_FTS` class used. This parameter is required. See Table 28 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.

Parameter	Description
`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 29 for a list of flags that can be used with this parameter.

Section 12

The *class* associated with the FTA_FTS parameter can be any of the following, as shown in Table 28:

Table 28. 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 29:

Table 29. 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. NPP A requires this flag.

Table 29. FTA_FLAGS parameters (continued)

Flag	Description
`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.

Changing the FTA configuration

12.2.2

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
   ```


12.2.3 FTA configuration for UNICOS systems

For UNICOS systems, FTA configuration is not maintained in the NLB. Instead, UNICOS FTA configuration is maintained in the FTA configuration file. To view `fta` configuration for UNICOS systems, use the `fta -config` command.

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.

```
% fta -config
FTA
queue="/usr/craysoft/nqe/2.0/latte/database/ftaqueue"
mode=0600
admin_users="root"
admin_groups="bin"
restartlimit=30
FTS
domain="inet"
fts=ftp
restartlimit=30
port=605
flags="nopassword"
```

Table 30 describes how the `fta -config` output corresponds to the output from the `nlbconfig` commands.

Table 30. `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.

For further information about FTA configuration on UNICOS systems, see the *FTA User and Administrator Manual*, publication SG–2144, and the `fta.conf`(5) man page.

12.2.4 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.

12.2.5 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:

```
/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  /usr/craysoft/nqe/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
```

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 Guide*, publication SG–5236.

Configuring network peer-to-peer authorization

12.2.6

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.

NPP A 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. NPP A 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 subsection 12.2.8, page 319.

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.

Configuring S-keys 12.2.6.1

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`.

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`.

NPPA example
12.2.7

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 31. 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 19 shows this configuration.

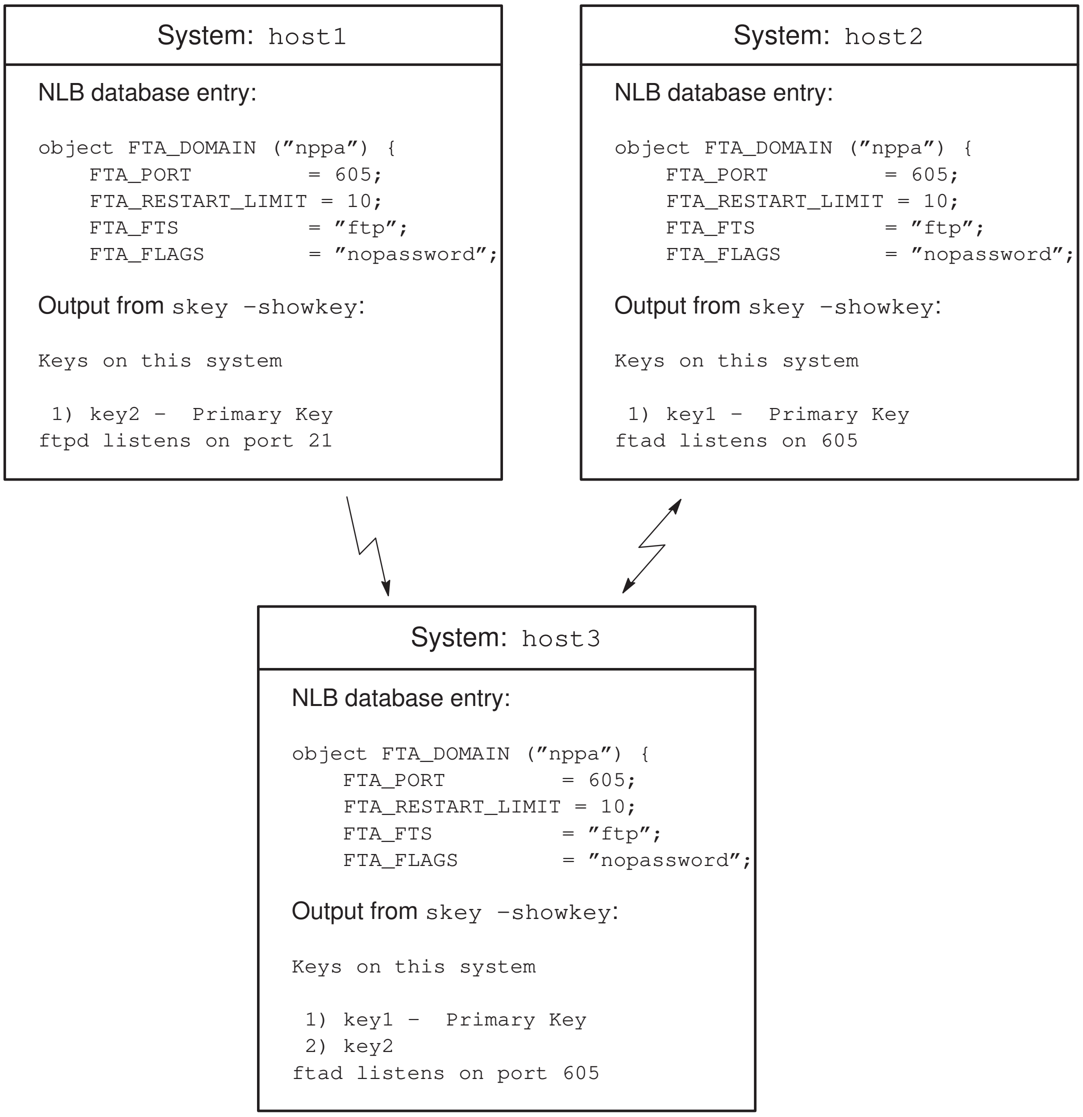

Figure 19. NPPA configuration example

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`.

Modifying the NLB database for NPPA

12.2.8

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; subsection 12.2.8.1 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 subsection 12.2.1, page 303.

Example of modifying the NLB database 12.2.8.1

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 subsection 12.2.2, page 309.

Administration

12.3

FTA startup, logging, and recovery functions are not entirely contained within the bounds of the `fta`(8) program. This subsection 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.

FTA system startup

12.3.1

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
```

FTA recovery

12.3.2

The following subsections describe FTA recovery.

Byte-level recovery of binary transfers

12.3.2.1

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 `/etc/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.

12.3.2.2 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 /usr/spool/fta/fqueue
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...
```

(continued)

Section 12

```
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
```


Types of failure
12.3.2.3

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
```

FTA recovery limits
12.3.2.4

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. To do so, define a recovery limit in the configuration file by using an `fta restart limit` parameter. The number of requests recovered at any one time, regardless of domain, is controlled by this limit. The following configuration example defines the restart limit as `30`:

```
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*. Subsection 12.2.2, page 309, 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.

FTA recovery strategy
12.3.2.5

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.

Information log
12.3.3

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 /etc/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.

Section 12

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
```

FTA management functions and status
12.3.4

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 example, user `joe` and the members of the group `operator` are authorized to perform FTA management functions:

```
fta { admin user joe, 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 example, user `joe` and the members of the group `staff` can display the status of any request:

```
fta { status user joe, 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
```

Section 12

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 /usr/spool/fta/fqueue
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
```

Section 12

```
% 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 /usr/spool/fta/fqueue
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
```

Transfer status

12.3.5

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 `/etc/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 [13]

This section describes how you administer Distributed Computing Environment (DCE) work when using NQE.

> **Note:** DCE support is optional for an installation; it is a licensed feature of NQE.

The following topics are described in this section:

- Enabling DCE support
- Obtaining DCE credentials
- Checkpoint and restart support
- User requirements

Section 13

Enabling DCE support

13.1

Before enabling DCE support, system administrators should note the following:

- The NQE Distributed Computing Environment/Distributed File Service (DCE/DFS) feature is restricted to operate within a single DCE cell.
- Ticket refreshing is supported on NQE UNICOS platforms, but not on NQE UNIX platforms for the NQE 3.0 release.
- NQE support of DCE/DFS does not support an installation running DFS-only file space. The spool and binary trees, among other components, must reside in UNIX file space.

To enable DCE support on an NQE master or execution server, manually edit the `/etc/nqeinfo` file as described:

- Add the `NQE_AUTHENTICATION` variable to the server `/etc/nqeinfo` file. Set the `NQE_AUTHENTICATION` variable to `dce`.

- If the path to the kdestroy command is not the default /opt/dcelocal/bin path, then add the NQE_DCE_BIN variable to the /etc/nqeinfo file and set it to the correct full path name.

  Note: No changes to the /etc/nqeinfo file are needed on NQE clients.

Obtaining DCE credentials

13.2

When the NQE_AUTHENTICATION variable is set to dce, NQS performs supplemental user authentication to obtain DCE credentials for the request owner . This operation is not done to authenticate the user to NQS, but to obtain DCE credentials. Any existing DCE credential the user may have at request submission time is not sent along with the request.

Ticket forwarding is not supported with this release of NQE. New credentials are obtained by NQS on the execution server host on behalf of the request owner just prior to initiating the request.

Note: This action is separate from the user authentication done by NQS when queueing a remotely submitted request or validating an alternate user identity. So, even though a password is required to obtain DCE credentials, the NQS user validation type defined within qmgr can remain at its current value. The NQS user validation type does not have to be changed to password validation.

The integrated login feature is used on NQE UNICOS servers to obtain the DCE credentials. This single DCE credential is used for both initiating the request and returning request output files.

Because there is not an integrated login feature available on NQE UNIX servers, NQS makes DCE security login calls to obtain a separate set of DCE credentials when initiating the request and when 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 can use the `klist` command within a request script file to verify that DCE credentials were obtained.

Messages are written to the NQS log file for both successful and failing attempts to obtain DCE credentials. A sample message indicating a successful attempt to obtain DCE credentials follows:

```
Request <1927.latte>: Acquired DCE credentials for user
<kohnke>,  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 <kohnke>
```


Checkpoint and restart support

13.3

On UNICOS platforms, 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` subcommands to 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. As a result, DCE services are affected but not DFS, which continues to work with the new credentials.

User requirements

13.4

Users must provide a password with the request if DCE credentials are desired.

Section 13

Although use of the DCE/DFS feature is transparent to users (except for the need to supply a password), they need to be aware of the following:

- A user's home directory can be within DFS space on both UNIX and UNICOS platforms.
- The request script file can be within DFS space.
- The user password supplied must be the same for both UNIX or UNICOS and the DCE registry password for that user. 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 `-u` option of the `cqsub` command.
- After a request completes, NQS uses `kdestroy` to destroy any credentials obtained by NQS on behalf of the request owner.

  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.

  On UNIX platforms, there is not an integrated login system feature available. NQS on UNIX platforms obtains separate DCE credentials for returning request output. Therefore, including a `kdestroy` command within a request script file running on an NQE UNIX server will not affect the return of request output files into DFS space.
- NQS supports DFS path name formats as follows so that request output files can be returned to DFS:
  - Directs the standard error file produced by the request to the specified file destination (this is specified by the `qsub -o` option if using the command-line interface)
  - Directs the standard output file to the specified file destination (this is specified by the `qsub -e` option if using the command-line interface)
  - Directs the job log to the specified file destination (this is specified by the `qsub -j` option if using the command-line interface)

Configuring `ilb` [14]

The `ilb` utility executes a command on a machine chosen by the Network Load Balancer (NLB). To use the utility, enter the `ilb` 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:

- `/etc/ilbrc` file (requires a specific format); see subsection 14.1
- `$HOME/.ilbrc` file (requires a specific format); see subsection 14.2, page 341
- User 's environment variables; see subsection 14.3, page 341

`/etc/ilbrc` file

14.1

The `ilbrc` file is located in `/opt/craysoft/nqe/etc` on Solaris systems and in `/usr/craysoft/nqe/etc` on all other platforms. There are two categories of information contained in the `ilbrc` file: LOGIN blocks and SYSTEM blocks.

Section 14

LOGIN blocks

14.1.1

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.
$PROMPT	Specifies a user's prompt; it contains a regular expression and may be changed by setting the environment variable ILB_PROMPT.

SYSTEM blocks

14.1.2

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`.

`$HOME/.ilbrc` file

14.2

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 subsection 14.1, page 339.

User environment variables

14.3

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.

Section 14

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 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 .

Section 14

Problem Solving [15]

This section helps you to identify and resolve problems while administering NQE. The following topics are discussed:

- Common problems with NQS, NQE clients, the NLB, and the NQE database, along with their possible causes and resolutions
- Networking debugging

For information on DCE/DFS issues, see section 13, page 335.

For information on obtaining support from customer service, see *NQE Installation*, publication SG–5236.

NQS troubleshooting

15.1

This subsection describes some common problems users may encounter in running NQS, along with their possible causes and resolutions.

A request has disappeared

15.1.1

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` command displays all requests on the server defined by the `NQS_SERVER` environment variable and 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* 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.

A queued request will not run
15.1.2

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 following command:

`cqstatl -f -h` *target_host* *requestids*

Compare these limits with the limits defined for the batch queue by using the following command to receive details about the queue:

`cqstatl -f` *queues*

Standard output and standard error files cannot be found
15.1.3

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 execution server on the user's submitting system.
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 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.

Syntax errors
15.1.4

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 .

`File not found` errors in standard error file
15.1.5

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.

Section 15

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.

Queues shut off by NQS

15.1.6

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.

Unexpected changes in request states

15.1.7

Requests may unexpectedly switch states from `queued` to `wait`. A checkpointed UNICOS 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.

Requests are queued waiting for license

15.1.8

If no license can be obtained, a request may switch to the `wait` state. NQS continues to try to obtain a license at regular intervals. If this occurs often, you may have to consider purchasing more licenses.

Problems issuing `qmgr` commands

15.1.9

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.
```

Section 15

Client troubleshooting

15.2

This subsection 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.

Client users cannot access the NQS server

15.2.1

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 subsection 15.5, page 351).

A request is accepted by NQS but not run

15.2.2

A request that remains in an NQS queue for a long time without executing usually means 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` command line. Ensure that they match the attributes of the batch queue.

15.2.3 `No account authorization at transaction peer` messages

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.

15.3 NLB troubleshooting

This subsection describes some common problems you may encounter in running NLB, along with their possible causes and resolutions.

15.3.1 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.

Section 15

Collector is running, but no data appears in server

15.3.2

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.

`nlbconfig -pol` reports errors

15.3.3

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.

Collectors on Solaris systems report incorrect data

15.3.4

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.

15.3.5 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.

15.3.6 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.

15.4 NQE database troubleshooting

This subsection describes some common problems you may encounter in running the NQE database, along with their possible causes and resolutions.

NQE database authorization failures

15.4.1

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 section 8, page 197.

NQE database connection failures

15.4.2

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 subsection 15.5.

Network debugging

15.5

The following subsections discuss how to diagnose network problems.

Checking connectivity

15.5.1

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.

15.5.2 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. An example display follows.

The most important information to check is the protocol type (`Proto`), your local address (`Local Address`), and the status of the connection (`state`).

```
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp        0      0  abcd13.6000            abcd13.1308            ESTABLISHED
tcp        0      0  abcd13.1308            abcd13.6000            ESTABLISHED
tcp        0      0  abcd13.1023            abcd.login             ESTABLISHED
tcp        0      0  abcd13.6000            abcd13.1267            ESTABLISHED
tcp        0      0  abcd13.1267            abcd13.6000            ESTABLISHED
tcp        0   1632  abcd13.6000            abcd13.1219            FIN_WAIT_1
tcp     4096      0  abcd13.1219            abcd13.6000            ESTABLISHED
tcp        0      0  *.6000                 *.*                    LISTEN
tcp        0      0  *.nqs                  *.*                    LISTEN
tcp        0      0  *.chargen              *.*                    LISTEN
tcp        0      0  *.daytime              *.*                    LISTEN
tcp        0      0  *.discard              *.*                    LISTEN
tcp        0      0  *.echo                 *.*                    LISTEN
tcp        0      0  *.time                 *.*                    LISTEN
tcp        0      0  *.finger               *.*                    LISTEN
tcp        0      0  *.uucp                 *.*                    LISTEN
tcp        0      0  *.exec                 *.*                    LISTEN
tcp        0      0  *.login                *.*                    LISTEN
tcp        0      0  *.shell                *.*                    LISTEN
tcp        0      0  *.printer              *.*                    LISTEN
tcp        0      0  *.telnet               *.*                    LISTEN
tcp        0      0  *.ftp                  *.*                    LISTEN
tcp        0      0  *.702                  *.*                    LISTEN
tcp        0      0  *.smtp                 *.*                    LISTEN
tcp        0      0  *.*                    *.*                    CLOSED
tcp        0      0  *.1024                 *.*                    LISTEN
tcp        0      0  *.sunrpc               *.*                    LISTEN
Active UNIX domain sockets
Address  Type   Recv-Q Send-Q    Vnode     Conn     Refs  Nextref Addr
ff64c98c dgram       0      0        0        0        0        0
ff65648c stream      0      0 ff090600        0        0        0 /tmp/.X11-unix/X0
ff64f50c stream      0      0        0 ff66698c        0        0
ff65700c stream      0      0        0 ff657d0c        0        0 /tmp/.X11-unix/X0
ff64fa8c stream      0      0        0 ff678e8c        0        0
ff65770c stream      0      0        0 ff64d20c        0        0 /tmp/.X11-unix/X0
ff657c0c stream      0      0        0 ff65348c        0        0
ff65b70c stream      0      0        0 ff64fe0c        0        0 /tmp/.X11-unix/X0
ff64f18c stream      0      0        0 ff64f48c        0        0
ff65340c stream      0      0        0 ff66cc0c        0        0 /tmp/.X11-unix/X0
ff672b8c stream      0      0        0 ff65628c        0        0
ff656a0c stream      0      0        0 ff64e20c        0        0 /tmp/.X11-unix/X0
ff65310c stream      0      0        0 ff653d0c        0        0
ff65380c stream      0      0        0 ff64ff8c        0        0 /tmp/.X11-unix/X0
ff65b88c stream      0      0        0 ff65660c        0        0 /tmp/.X11-unix/X0
ff64d70c stream      0      0 ff0d2620        0        0        0 /dev/printer
ff65360c stream      0      0        0 ff67870c        0        0
ff67e80c stream      0      0        0 ff64f00c        0        0
ff666c8c stream      0      0        0 ff658e0c        0        0 /tmp/.X11-unix/X0
ff64408c dgram       0      0 ff089d28        0        0        0 /dev/log
ff64970c dgram       0      0        0        0        0        0
```

Section 15

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 `/usr/craysoft/nqe/man`, except on Solaris systems where it is `/opt/craysoft/nqe/man`.

User-level man pages

A.1

The following user-level online man pages are provided with your NQE software:

User -level man page	Description
`cdoc`(1)	Brings up the CrayDoc online documentation reader , from which you can access the NQE documentation
`cevent`(1)	Posts, reads, and deletes job-dependency event information
`cload`(1)	Provides an X Window System display of machine load 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

User-level man page	Description
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 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)

Application interfaces man pages

A.2

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).

Product support man page

A.3

The cs_support(7) online man page is provided with your NQE software; it includes international product support telephone numbers.

Administrator-level man pages

A.4

The following administrator-level online man pages are provided with your NQE software:

Administrator -level man page	Description
addnode(8)	Copies NQE server or client software to additional systems
ccollect(8)	Starts and manages an NLB or network job status (NJS) collector
csuspend(8)	Suspends NQE batch activity
fta(8)	Performs file transfer management
ftad(8)	Invokes the FTA file transfer protocol server
inst(8)	NQE installation tool
nlbconfig(8)	Configures and maintains the NLB server
nlbpolicy(8)	Queries the NLB server
nlbserver(8)	Starts an NLB server
nqedbmgr(8)	NQE database administration
nqeinit(8)	Starts NQE
nqestop(8)	Stops NQE
nqsfts(8)	File transfer service for NQS

Administrator-level man page	Description
`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

config.h/nqeinfo File Variables [B]

Appendix B

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.

> **Note:** The nqeinfo file must be manually edited to add any of the NQE_NQS_*xxxx* variables listed in this appendix.

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 memory 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 core 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.

Appendix B

`NQE_NQS_DLIM_PPDATAUNITS    IC_DLIM_PPDATAUNITS`

Default is: `QLM_MWORDS`

Defines the default per-process memory 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 (MPP) CRAY T3D 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 core 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 memory 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 core 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 memory 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 core file 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 MPP 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 MPP CRAY T3D 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: 128

Defines the maximum number of MPP processing elements (PEs) that are accessible at the local host. This variable is an integer from 0 to 2048.

`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 a UNICOS system

Enables checking for MPP availability. If enabled, NQS checks for the availability of the MPP 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_SHELL_INVOCATION 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_SHELL_PATH 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 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_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. 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

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

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

D

Index

E

F

Index

G

H

I

J

K

L

Index

Index

Index

Index

Index

Index

Q

Index

R

T

U

V

W

Index

Reader's Comment Form

NQE Administration SG–2150 3.0

Your reactions to this manual will help us provide you with better documentation. Please take a moment to complete the following items, and use the blank space for additional comments.

List the operating systems and programming languages you have used and the years of experience with each.

Your experience with Cray Research computer systems: _____0-1 year _____1-5 year _____5+years

How did you use this manual: _____in a class _____as a tutorial or introduction _____as a procedural guide _____as a reference _____for troubleshooting _____other

Please rate this manual on the following criteria:

	Excellent			Poor
Accuracy	4	3	2	1
Appropriateness (correct technical level)	4	3	2	1
Accessibility (ease of finding information)	4	3	2	1
Physical qualities (binding, printing, illustrations)	4	3	2	1
Terminology (correct, consistent, and clear)	4	3	2	1
Number of examples	4	3	2	1
Quality of examples	4	3	2	1
Index	4	3	2	1

Please use the space below for your comments about this manual. Please include general comments about the usefulness of this manual. If you have discovered inaccuracies or omissions, please specify the number of the page on which the problem occurred.

Name ______________________
Title ______________________
Company ______________________
Telephone ______________________
Today's date ______________________

Address ______________________
City ______________________
State/Country ______________________
Zip code ______________________
Electronic mail address ______________________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO 6184 ST. PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

ATTN: Software Publications Group
655 LONE OAK DR BLDG F
EAGAN MN 55121-9957

Fold

Cut along this line

Reader's Comment Form

NQE Administration SG–2150 3.0

Your reactions to this manual will help us provide you with better documentation. Please take a moment to complete the following items, and use the blank space for additional comments.

List the operating systems and programming languages you have used and the years of experience with each.

Your experience with Cray Research computer systems: _____0-1 year _____1-5 year _____5+years

How did you use this manual: _____in a class _____as a tutorial or introduction _____as a procedural guide _____as a reference _____for troubleshooting _____other

Please rate this manual on the following criteria:

	Excellent			Poor
Accuracy	4	3	2	1
Appropriateness (correct technical level)	4	3	2	1
Accessibility (ease of finding information)	4	3	2	1
Physical qualities (binding, printing, illustrations)	4	3	2	1
Terminology (correct, consistent, and clear)	4	3	2	1
Number of examples	4	3	2	1
Quality of examples	4	3	2	1
Index	4	3	2	1

Please use the space below for your comments about this manual. Please include general comments about the usefulness of this manual. If you have discovered inaccuracies or omissions, please specify the number of the page on which the problem occurred.

Name ____________________
Title ____________________
Company ____________________
Telephone ____________________
Today's date ____________________

Address ____________________
City ____________________
State/Country ____________________
Zip code ____________________
Electronic mail address ____________________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO 6184 ST. PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

ATTN: Software Publications Group
655 LONE OAK DR BLDG F
EAGAN MN 55121-9957

Fold

Cut along this line