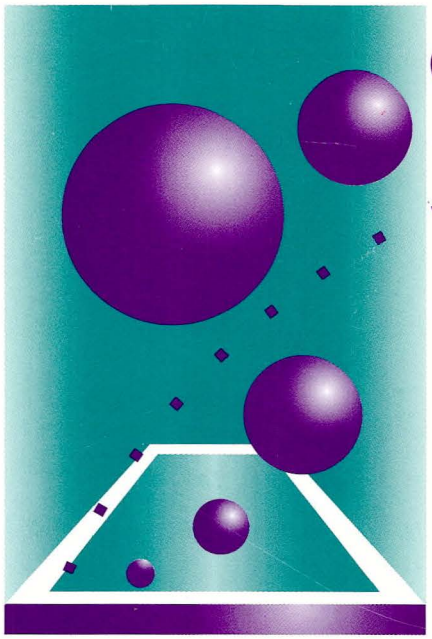


CONVEX

ConvexNQS+ System Manager's Guide for Exemplar Systems

First Edition

CONVEX Computer Corporation
3000 Waterview Parkway
P.O. Box 833851
Richardson, TX 75083-3851
United States of America
(214)497-4000

ConvexNQS+ System Manager's Guide for Exemplar Systems

Order No. DSW-860

First Edition
August 1994

CONVEX Press
Richardson, Texas
United States of America

ConvexNQS+
System Manager's Guide
for Exemplar systems

Order No. DSW-860

Printed in the United States of America

Revision information for

ConvexNQS+
System Manager's Guide for Exemplar Systems

Edition	Document No.	Description
First	770-007430-000	Released for ConvexNQS+ software V2.0 for use with CONVEX Exemplar systems, August 1994.

Contents

Figures

Tables

Using this book

Purpose and audience

The *ConvexNQS+ System Manager's Guide for Exemplar Systems* addresses the needs of ConvexNQS+ managers and operators on Exemplar systems, and covers the following topics:

- Basic ConvexNQS+ concepts
- How to configure your ConvexNQS+ network
- How to maintain queues and queue requests on a regular basis

Organization

Specifically, this guide is organized into the following chapters:

- Chapter 1 introduces general concepts necessary for using NQS+ software
- Chapter 2 describes how to configure ConvexNQS+ for the first time, and how to change your existing configuration
- Chapter 3 and Chapter 4 describe how to maintain queues and queue requests
- Chapter 5 describes how to generate accounting reports
- Chapter 6 and Chapter 7 contain alphabetical listings of the commands available with the `qmgr` and `qmapmgr` utilities respectively
- Appendix A describes the ConvexNQS+ run-time directory hierarchy
- Appendix B lists ConvexNQS+ daemons

Conventions

This section discusses notational conventions used in this book.

Command syntax

Consider this example:

```
COMMAND input_file [...] {a | b} [output_file]
```

`COMMAND`	must be typed as it appears.
input_file	indicates a file name that must be supplied by the user.
`[...]`	The horizontal ellipsis in brackets indicates that additional input file names may be supplied.
`{a	b}`	Either `a` or `b` must be supplied.
[output_file]	enclosed in brackets indicates an optional file name supplied by the user.

General conventions

In general, the following conventions are used in this guide:

- *Italics*:
  - Designate user-supplied variables in a command-line example
  - Introduce new and important terms
  - Indicate document titles
- `Constant-width font` designates:
  - System output in screens and examples
  - Command names and options
  - Directives, program statements, display examples, printout examples, and error messages returned.
- **`Bold, constant-width font`** designates user input in screens and examples, and must be typed exactly as it appears.
- Vertical ellipsis shows that lines of code have been left out of an example.
- Words and abbreviations that indicate keyboard keys you press are identified in a distinctive bold type. For example, **RETURN** refers to the carriage return key. Words separated by a hyphen indicate two keys that you must press simultaneously. For example, **CTRL-X** indicates that you must press and hold down the **CTRL** key and then press the **X** key.

- The word "enter" in a phrase such as "enter `ls`" means that you type the command and then press **RETURN**.
- References to man pages appear in the form adb(1), where the name of the man page is followed by its section number enclosed in parentheses.

Note

A note highlights supplemental information.

Associated documentation

Using this software may require information not specific to the tasks described in this document.

For more information on Convex products, you can order these books from CONVEX Computer Corporation:

- *ConvexNQS+ User's Guide for Exemplar Systems* (DSW-861). Describes how to use ConvexNQS+ to submit, track, and control jobs for batch execution in queues.
- *CONVEX SPP-UX System Administration Guide* (DSW-853). Describes how to configure, control access to, and manage the file system of the SPP-UX operating system. Also describes the subcomplex manager, system accounting, and printer management.

Ordering documentation

To order the current edition of this or any other CONVEX document, send requests to:

CONVEX Computer Corporation
Customer Service
P.O. Box 833851
Richardson TX 75083-3851 USA

Include the order number or the exact title, as listed on the front cover.

Technical assistance

If you have questions that are not answered in this book, contact the CONVEX Technical Assistance Center (TAC).

- Within the continental U.S., call 1(800)952-0379.
- From Canada, call 1(800)345-2384.
- Outside continental U.S., contact local CONVEX office.

ConvexNQS+ overview

1

This chapter introduces basic ConvexNQS+ concepts and features, including:

- ConvexNQS+ basic processing
- Utilities available for configuring and controlling ConvexNQS+
- Levels of authorization required to take advantage of these utilities
- Differences between ConvexNQS+ and other NQS systems

Queue processing

ConvexNQS+ lets users submit jobs to a queue for batch execution.

A queue is a list of requests that are ready and waiting to run.

A request is one or more commands submitted by a user or a user program to a queue. These commands are usually run after a certain time or event has passed, and do not require further interaction with the user. A typical request is a program containing commands that perform large-scale, detailed computations on a static data file.

Users can submit requests to queues that reside on either a local or remote machine configured with ConvexNQS+ or another version of Network Queueing System (NQS).

ConvexNQS+ queues

There are two kinds of ConvexNQS+ queues:

- **Batch**—Batch queues run requests. They hold requests for scheduled, perhaps delayed, processing by subsystems within ConvexNQS+. Demand queues are special batch queues that accept requests only if they can place the requests into immediate execution.
- **Pipe**—Pipe queues are routing queues that do not directly run requests, but instead transmit requests to other queues. Each pipe queue has a pipeclient that does the actual routing and set of destination queues that are possible recipient queues. A destination queue can be either a batch queue, a demand queue, or another pipe queue on either the local or a remote machine.

ConvexNQS+ pipe clients

There are three types of ConvexNQS+ pipe clients:

- **pipeclient**—Routes a request to the first queue in its destination set that is able to accept the job.

  Destinations may reject the request due to queue limit violations or lack of account authorization, to name a few possibilities.
- **pipedemand**—Routes a request to the first queue in its destination set that can immediately run the request.

  A destination cannot immediately run a request if the number of jobs currently running at the destination matches its run limit.

If a request cannot be routed to a destination, the request stays queued in the pipe queue until a destination becomes available.

pipedemand minimizes the time a request spends in a queued state, and maximizes job throughput in a cluster environment.

- **pipeldav**—Sorts its destination set by load factor and routes a request to the destination with the lowest load factor that is able to accept the job.

  pipeldav places requests into queues quickly. It does not wait until a queue is empty or spend unnecessary time polling queues. Neither does pipeldav give all the jobs to a processor with a light load, because each job placed in a queue increases the load factor.

To route requests based on load factor, pipeldav:

- Scans the set of destination queues to determine associated host machines.
- Determines the host load average for each host machine using rstatd software loaded on the host machine. rstatd is an optional NFS product; contact your CONVEX sales representative for additional information.
- Determines the queue length for each destination queue by querying the netdaemon on the host machine.
- Calculates the load factor for each destination queue using the following formula:

$$\text{load factor} = \frac{\text{host load average} + (\text{queue length} \times \text{weight})}{\text{processor speed}}$$

  A ConvexNQS+ manager sets the weighting factor as an option of the pipeldav program; the weighting factor defaults to 1.0.

- Routes the request to the destination queue with the lowest load factor that is able to accept the job.

ConvexNQS+ configuration and control utilities

The following two ConvexNQS+ utilities configure and operate ConvexNQS+:

- `qmapmgr`
- `qmgr`

These utilities are discussed in the following sections.

qmgr utility

`qmgr` is the ConvexNQS+ utility that controls queues and requests on the local machine.

You can use the `qmgr` utility at any time to

- Abort queues
- Create queues
- Configure queues
- Delete queues
- Enable and disable queues
- Move requests from one queue to another
- Reorder requests inside a queue
- Set queue attributes
- Show information about attributes, managers, and queues
- Start and stop queues
- Start and stop ConvexNQS+

See Chapter 6, "`qmgr` commands," in this guide for detailed information on `qmgr`.

qmapmgr utility

`qmapmgr` is the ConvexNQS+ utility that builds and maintains a network database used to establish a mapping between ConvexNQS+ and each machine in the ConvexNQS+ configuration.

Without this mapping, ConvexNQS+ cannot recognize local and remote destinations and queues.

You can use `qmapmgr` to make changes to the network database on a local machine at any time; however, make sure that all machines have the same network database.

See Chapter 7, "`qmapmgr` commands," in this manual for detailed information on `qmapmgr`.

Levels of authorization

The `qmgr` commands available to each user depends on user type. ConvexNQS+ distinguishes three user types:

- **General users**—General users can track and control their own requests.
- **Operators**—Operators can track and control *any* user's requests, manage queues, and set run limits. Managers assign operator privileges to users.
- **Managers**—Managers have access to all `qmgr` commands. See Chapter 6, "`qmgr` commands," for a complete list. By default, root is a ConvexNQS+ manager and can assign this privilege to other users.

Differences between ConvexNQS+ and NQS

There are five major differences between ConvexNQS+ and other NQS systems. Understanding the following differences is important if you combine several different systems at one site:

- The ConvexNQS+ `move my_request` command does not exist in other NQS systems and can only be used within ConvexNQS+.
- ConvexNQS+ can mount remote files using NFS, other systems cannot.
- The ConvexNQS+ `maximum_request_priority` command, if used when submitting a request between ConvexNQS+ and another NQS system, decreases the priority of previously submitted requests. It does not delete previously submitted requests.
- Direct submission (for example, `qsub -q long@host`) between machines running ConvexNQS+ and other machines is not possible.
- Several ConvexNQS+ packet numbers are not recognized by other NQS systems.

Configuring ConvexNQS+ 2

ConvexNQS+ is suitable for most single-machine system configurations as it is shipped and installed; however, every site is different. You may want to:

- Create an optimal, efficient configuration that matches your work loads to available resources
- Configure ConvexNQS+ to handle site-specific situations
- Install ConvexNQS+ on multiple machines

Summary of steps

This chapter describes tasks you must perform to install and configure the ConvexNQS+ system. These tasks, in recommended order of performance, are:

Step 1 Install the base ConvexNQS+ system

Step 2 Taking a `qmapmgr` snapshot

Step 3 Assigning managers and operators

Step 4 Using the `add manager` command

Step 5 Using the `op` utility

Step 6 Determining queue structure

Step 7 Creating and configuring batch queues

Step 8 Creating pipe queues

Step 9 Deleting queues

Step 10 Configuring and activating ConvexNQS+ accounting

Step 11 Establishing a shell strategy

Step 12 Taking a `qmgr` snapshot

Step 13 Configuring error logging

Step 14 Automating queue operations

Step 15 Notifying users of changes

Getting started

Several procedures in this chapter require viewing the attributes of queues. Use the `show long queue` command in the `qmgr` utility to accomplish this.

You must start the `qmgr` utility before you can use this command. To start `qmgr`, enter:

```
qmgr
```

The command format is

`show long queue` *queuename*

where *queuename* is the name of the queue.

Figure 1 illustrates the output for this command when viewing the attributes for a batch queue. The attributes for a pipe queue are a subset of the batch queue attributes. That is, a pipe queue has fewer attributes.

Figure 1 Displaying queue attributes

```
# qmgr
Mgr: show long queue s
s@hostC; type=BATCH; [ENABLED, INACTIVE]; pri=48
0 exit; 0 run; 0 stage; 0 queued; 0 wait; 0 hold; 0 arrive;
Run_limit = 2; Per-user run-limit = NONE
Accounting: Off
Activity ID offset: 1
Maximum request priority: 63
Cumulative system space time = 0.000000 seconds
Cumulative user space time = 0.000000 seconds
Unrestricted access
Import directory: Yes
Shell execution as login: No
Per-process permanent file size limit = UNLIMITED
Per-process execution nice value = 0 <DEFAULT>
```

The first line of the output describes information about the queue.

```
s@hostC; type=BATCH; [ENABLED, INACTIVE]; pri=48
```

`s@hostC` is the name of this queue and the machine on which it is configured. In this example, the attributes shown apply to the *s* queue on *hostC*.

`type=BATCH` is the type of queue. Where:

`BATCH`	Runs ConvexNQS+ requests.
`DEMAND`	<help>
`PIPE`	Routes ConvexNQS+ requests to queues that can run them.

[ENABLED, indicates if this queue can accept requests. Where:

- ENABLED — ConvexNQS+ is running on the local machine, and the queue is accepting requests.
- DISABLED — ConvexNQS+ is running on the local machine, but the queue is not accepting requests.
- CLOSED — ConvexNQS+ is not running on the local machine.

INACTIVE] indicates if this queue can run requests. Where:

- INACTIVE — Requests in the queue can run; none are running.
- RUNNING — Requests in the queue can run; some are running.
- STOPPING — New requests sent to the queue cannot run, but requests that are currently running can complete.
- STOPPED — Requests in the queue cannot run; none are running.
- SHUTDOWN — ConvexNQS+ is not running on the local machine.

pri=48 indicates interqueue priority. This priority affects queue selection for running the next job. Priority can be any number from 0 to 63; 0 is the lowest priority and 63 the highest.

The second line of the output tallies the number of requests in specific states:

```
0 exit; 0 run; 0 stage; 0 queued; 0 wait; 0 hold; 0 arrive;
```

- 0 exit — Number of requests in this queue in a state of exiting.
- 0 run — Number of requests in this queue in a state of running.
- 0 stage — Number of requests in this queue in a staged state, which means the request has completed running and ConvexNQS+ is moving the stdout and stderr files to the appropriate destination directory.
- 0 queued — Number of requests in this queue in a state of being queued.

`0 wait` Number of requests in this queue in a state of waiting.

`0 hold` Number of requests in this queue in a state of holding.

`0 arrive` Number of requests in this queue in a state of arriving.

The next 11 lines show miscellaneous information about the queue.

```
Run_limit = 2; Per-user run-limit = NONE
Accounting: Off
Maximum request priority: 63
Cumulative system space time = 0.000000 seconds
Cumulative user space time = 0.000000 seconds
Unrestricted access
Import directory: Yes
```

`Run_limit` The maximum number of requests that can run in this queue at any one time. When the limit is exceeded, ConvexNQS+ queues requests until the number of jobs running is less than the limit (applies to batch queues only).

`Per-user run-limit`
The number of requests a user may run in this queue at any one time. Per-user run limits apply after per-queue run limits (applies to batch queues only).

`Accounting` Indicates if batch accounting is activated (applies to batch queues only).

`Maximum request priority`
The maximum priority at which a request can be submitted to the queue.

`Cumulative system space time`
Batch queues: the total amount of system time used to process requests in this batch queue since the queue was created.
Pipe queues: the total amount of system time used to route requests in this pipe queue since the queue was created.

`Cumulative user space time`
The total amount of user time used to process requests in this queue since the queue was created.

`Unrestricted access`

Indicates the access restrictions placed on this queue. These restrictions do not apply to a request submitted by the superuser; superuser requests are always queued. Access restrictions can be:

`Unrestricted`
: This queue can accept any request from any submitter.

`Restricted`
: This queue can accept only those requests submitted by a specified group(s) or user(s).

`Pipeonly`
: This queue can accept requests only from a pipe queue.

`Demand—<help>`

`Import directory`

Indicates if this queue imports the current working directory for a request (mounts the directory on the machine processing the request) before running the request (applies to batch queues only). This can be:

`Yes`
: This queue automatically imports the current working directory for any request running in the queue. A user can override this setting for individual requests using the `-ni` option of `qsub`.

`Available`
: Lets a user specify importation of the current working directory for any request submitted to this queue using the -i option of `qsub`.

`No`
: This queue does not import the current working directory for requests submitted to the queue. Furthermore, this queue rejects requests that require imported directories.

The rest of the output displays the per-process resource limits (if any) configured for this queue. SPP-UX can enforce only the per-process permanent file size limit and the per-process execution nice value.

If a user submits a request with limit specifications to a queue, ConvexNQS+ checks the request limits to ensure they do not exceed the enforceable limits set for the queue. If the request limits exceed the queue limits, ConvexNQS+ rejects the request.

If a user submits a request without limit specifications to a queue, ConvexNQS+ uses the enforceable limits set for the queue as default limits.

ConvexNQS+ assigns limits to a request when the request is queued. If a queue limit is changed after a request is queued, the queued request is not affected; however, if the previously-queued request exceeds the new queue limit, ConvexNQS+ displays a warning message.

`Per-process permanent file size limit`

The maximum permanent file size for any process in a running request. If any process tries to create a permanent file larger than the limit set, ConvexNQS+ sends a SIGXFSZ signal to the offending process. If the process does not catch the signal or ignores the signal, the process exits.

`Per-process execution nice value`

The minimum nice value for any process in a running request. The nice value determines the proportion of CPU time allocated to a process relative to all other processes in the system. The lower the nice value assigned to a process, the higher the proportion of CPU time allocated to that process. A practical range is from -20 to 20.

Installing the base ConvexNQS+ system

Perform the following steps to install your ConvexNQS+ base system.

Step 1 Determine which machines will run ConvexNQS+.

Step 2 Install ConvexNQS+ on the desired machines. To install the base ConvexNQS++ system on each machine, follow the steps listed in the *ConvexNQS+ Installation Procedure.*

Step 3 Log in as the superuser on a machine you are configuring.

Step 4 Check to see if nameserver is running.
To determine if the system is using `named`, check for the file /etc/resolved.conf and/or enter `% ps -eadf` and grep for `named`. If the nameserver is running, you must add the fully qualified domain name to the `qmapmgr` database when you first start ConvexNQS+.

Step 5 Enter the following command at the system prompt:

```
qmapmgr
```

The `qmapmgr` prompt appears:

```
Mapmgr:
```

Step 6 Create the nmap database on the local machine by entering:

```
create
```

Step 7 Add the MID for each host to the nmap database using the `add mid` command. Enter a separate command for each machine that will run ConvexNQS+. The command format is

`add mid` *n machine_name*

where

n	is a unique number identifying the machine that will run ConvexNQS+.
machine_name	is the name of the machine that will run ConvexNQS+. This name should correspond to an entry in the /etc/hosts file; it should not be an alias.

Figure 2 illustrates the use of the `create` and `add` commands to create the nmap database and assign MIDs to *hostA*, *hostB*, and *hostC*.

Figure 2 Creating nmap database

```
# qmapmgr
Mapmgr: create
nmap_success: successful completion.
Mapmgr: add mid 1 hostA
nmap_success: successful completion.
Mapmgr: add mid 2 hostB
nmap_success: successful completion.
Mapmgr: add mid 3 hostC
nmap_success: successful completion.
```

Step 8 If nameserver is not running, skip this step.

If nameserver is running, supply the fully qualified host name as an alias for each machine you are adding using the `add name` command. For example, if the fully qualified name for *hostA* is hostA.berkley.edu, enter:

```
add name hostA.berkley.edu 1
```

where 1 is the MID assigned to *hostA*.

Step 9 Exit `qmapmgr` by entering

```
exit
```

Step 10 Repeat Step 3 through Step 9 on each machine that will run ConvexNQS+.

Make sure you enter the same MID/machine name combinations at each machine. That is, *hostA* in the sample configuration must be MID 1 on all machines.

Taking a qmapmgr snapshot

After installing the base ConvexNQS+ system on each machine, save the current ConvexNQS+ configuration on each machine as a series of `qmapmgr` commands using the `qsnapshot -m` command.

This command saves the configuration to screen by default; however, you can save the configuration to a file and later use the contents of that file to restore the ConvexNQS+ `qmapmgr` configuration. For example, to save the configuration to a file named batch_nconfig, enter

```
qsnapshot -m > batch_nconfig
```

Figure 3 illustrates output from the `qsnapshot -m` command. For more information, refer to the qsnapshot(8), qmgr(8), and qmapmgr(8) man pages.

Figure 3 `qmapmgr` system snapshot

```
# qsnapshot -m
CREATE
ADD MID 1 hostA
ADD MID 2 hostB
ADD MID 3 hostC
```

Assigning managers and operators

After taking a snapshot of the ConvexNQS+ `qmapmgr` configuration on each machine, set up ConvexNQS+ managers and operators for each machine.

A ConvexNQS+ manager has access to all ConvexNQS+ commands. A ConvexNQS+ operator has access to ConvexNQS+ commands for managing queue requests, managing queues, and managing ConvexNQS+ in general. See the Chapter 1 section entitled "Levels of authorization" for a complete list of operator commands.

You can set up ConvexNQS+ managers and operators the following two ways:

- Use the `add manager` command in `qmgr` to grant users access to all commands defined by ConvexNQS+ as operator or manager commands.
- Use the op utility in ConvexOS to grant access to specific commands.

Using the add manager command

Perform the following steps to grant ConvexNQS+ manager and operator privileges using the `add manager` command:

Step 1 Log in as the superuser on a machine you are configuring.

Step 2 Start the `qmgr` utility by entering:

```
qmgr
```

Step 3 The `qmgr` prompt appears:

```
Mgr:
```

Step 4 Add ConvexNQS+ managers and operators using the `add manager` command. The command format is

`add manager` *user_name:x*

where

user_name	is the name of the user who is to have manager or operator access.
x	indicates manager or operator access. Enter `m` to grant manager access; enter `o` to grant operator access.

Figure 4 illustrates use of this command to grant manager privileges to the users batchman.

Figure 4 Granting ConvexNQS+ manager privileges

```
# qmgr
Mgr: show managers
 root:m
Mgr: add manager batchman:m
ConvexNQS+
manager[TCML_COMPLETE]:transaction complete
at local host
```

Step 5 Repeat these steps on each machine running ConvexNQS+.

Step 6 Exit `qmapmgr` by entering

```
exit
```

Using the op utility

To grant access to specific commands using the op utility, specify in the /etc/op.access file the command(s) accessible to each user. For example, enter the following lines in the /etc/op.access file to grant the privileges illustrated in Table 1.

```
enableq /usr/convex/qmgr enable queue *1;users=batchman

disableq /usr/convex/qmgr disable queue *1;users=batchman

deletereq /usr/convex/qmgr delete request *1;users=batchman, batchop
```

Table 1 Example granting access privileges using the op utility

Command privilege	batchman	batchop
Enable queue	✔	
Disable queue	✔	
Delete request	✔	✔

Make sure that programs listed in the op.access file are secure. As a general protective measure, make sure the op.access file does not contain interactive programs or shell scripts that run as root.

If the /etc/op.access file does not already exist, you must create it. Refer to *Managing ConvexOS: Configuration Guide* or the op.access(5) man page for detailed information on using the op utility and setting up the op.access file.

Determining queue structure

After assigning ConvexNQS+ manager and operator privileges, determine the queue structure needed to satisfy the requirements of your users. Do this for each machine configured with ConvexNQS+.

There are two types of ConvexNQS+ queues

- **Batch**
  Batch queues run requests. They hold requests for scheduled, perhaps delayed, processing by subsystems within ConvexNQS+. Demand queues are special batch queues that accept requests only if they can place the requests into immediate execution.
- **Pipe**
  Pipe queues are routing queues that do not directly run requests, but instead transmit requests to other queues. Each pipe queue has a pipeclient that does the actual routing and set of destination queues that are possible recipient queues. A destination queue can be either a batch queue, a demand queue, or another pipe queue on either the local or a remote machine.

Assume a sample configuration with four machines: *hostA*, *hostB*, *hostC*, and *hostD*. Your intent:

- Use *hostA* primarily for electronic mail, word processing, spreadsheet and other applications that require reasonable interactive response time.
- Use *hostB*, *hostC* and *hostD* primarily for number-crunching, resource-intensive applications.

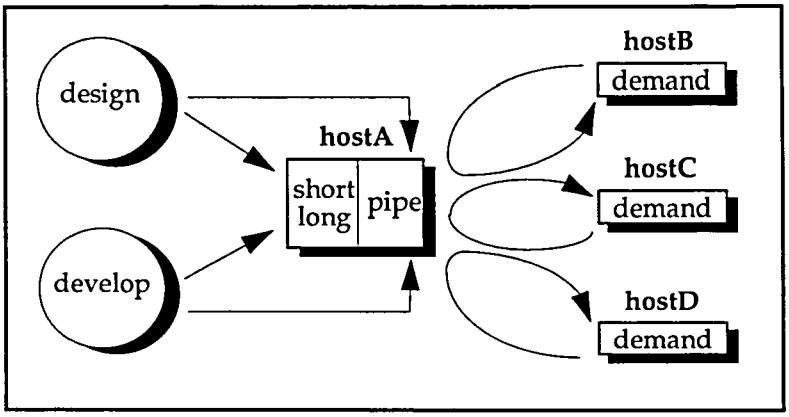
Also assume two user groups in this sample configuration: a development group and a design group. Some users may belong to both groups. Each user in each group has an account on each machine, and the login names and UIDs are the same across machines (that is, no account mapping is required).

Recommendation:

- Create three demand queues—one on *hostB*, one on *hostC* and one on *hostD*.
- Create a pipe queue on *hostA*, from which the demand queues can *pull* resource-intensive jobs.

Figure 5 illustrates this configuration.

Figure 5 Sample queue configuration



All users can submit jobs that require little or moderate resource usage to the default batch queues on *hostA*.

Design users can submit long-running jobs to the pipe queue on *hostA*. Whenever a demand queue on *hostB*, *hostC* or *hostD* becomes idle, it pulls a queued job from the pipe queue and runs it.

Development users can also submit long-running jobs to the pipe queue on *hostA*. Whenever a demand queue on *hostB*, *hostC* or *hostD* becomes idle, it pulls a queued job from the pipe queue and runs it.

Not only does this sample configuration minimize the time a job spends in a queued state and maximize job throughput in a cluster environment, but it also prevents one group from monopolizing the other group's resources.

While this sample configuration is not designed to cover every situation you may encounter, it covers the main issues you must consider when establishing your system.

Creating batch queues

Once you determine the queue structure needed to satisfy user requirements, you may need to add batch queues to all or some of the machines configured with ConvexNQS+.

Perform the following steps to add batch queues:

Step 7 Log in as a ConvexNQS+ manager on a machine you are configuring.

Step 8 Start the `qmgr` utility by entering:

```
qmgr
```

The `qmgr` prompt appears:

```
Mgr:
```

Step 9 Add a batch queue using the `create batch_queue` command. The command format is

```
create batch_queue queuename priority=priority
[pipeonly] [import_dir=import-option]
[run_limit=run-limit] [demand] [sender=sender]
```

where

queuename is the name of the queue. The name can consist of any printable nonblank character except for the at sign (@), a comma (,), an equal sign (=), and a left or right parenthesis ( ). The name cannot start with a digit.

priority is the interqueue priority for the queue. This priority affects queue selection for running the next job. Can be any number from 0 to 63; 0 is the lowest priority and 63 the highest.

`pipeonly` indicates the queue can only accept requests submitted from a pipe queue. Otherwise, the queue can accept requests from any source (such as a program or script file).

import_option indicates if ConvexNQS+ imports the current working directory for a request (mounts the directory on the machine processing the request) before running the request. This can be:

`Yes` ConvexNQS+ automatically imports the current working directory for any request running in the queue. A user can override this setting for individual requests using the -ni option of `qsub`.

`Available`	Lets a user specify importation of the current working directory for requests submitted to the queue using the `-i` option of `qsub`.
`No`	ConvexNQS+ does not import the current working directory for requests. Furthermore, the queue rejects requests that require imported directories.
	If you specify Yes or Available, ConvexNQS+ imports directories with NFS by making temporary mount points in the /tmp directory of the originating machine. Be aware of local automatic clean-up facilities of /tmp that could change these NFS mount points.

run_limit is the maximum number of requests that can run in the queue at one time. For example, if you submit four requests to a queue whose run limit is set to two, two of the requests begin running and two are queued. If you do not supply a run limit, the value defaults to one.

`demand` indicates that requests may enter this queue only if they can execute immediately (which is determined by *run-limit*). `Demand` implies `pipeonly`.

sender is the name of the pipe queue(s) that can send requests to this batch queue, and applies only if `demand` is specified. If you supply more than one sender, separate items in the list with commas and surround the list with parentheses. For example, to designate three destinations, enter:

```
(sendq1, sendq2, sendq3)
```

The syntax of the sender queue name must be in one of the following forms. Square brackets [ ] are required where shown.

local_queue_name

local_queue_name@local_machine_name

remote_queue_name@remote_machine_name

remote_queue_name@[remote_machine_mid]

Caution

Make sure all senders are defined with a server=(/usr/lib/nqs/pipedemand) attribute.

For example, to add a new batch queue named *b* with an interqueue priority of 48, enter:

```
create batch_queue b pr=48
```

Step 10 Repeat Step 9 for each added batch queue on this machine.

Step 11 Exit `qmgr` by entering:

```
exit
```

Step 12 Repeat Step 1 through Step 11 on each machine running ConvexNQS+.

Configuring batch queues

Once you add any needed batch queues, you must configure them. You must also configure all default batch queues on all machines configured with ConvexNQS+.

Perform the following steps to configure batch queues:

Step 1 Log in as a ConvexNQS+ manager on the machine you are configuring.

Step 2 Start the `qmgr` utility by entering:

```
qmgr
```

The `qmgr` prompt appears:

```
Mgr:
```

Step 3 Set the per-user run limit for the queue using the `set per_user run_limit` command. The command format is

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

where

run-limit is the number of requests a user can run in a queue at one time. To turn off per-user run limit, set this value to 0. ConvexNQS+ applies per-user run limits after applying per-queue run limits.

queuename is the name of the queue for which you are setting the limit.

Step 4 Set the global per-user run limit for all queues using the `set global per_user run_limit` command. The command format is

`set global per_user run_limit` = *run-limit*

where *run-limit* is the number of requests a user can run in all ConvexNQS+ queues at one time.

Table 2 shows how per-user, per-queue, and global run limits work together. Requests with an asterisk begin running; requests without asterisk are queued.

Table 2 Per-user, per-queue, and global run limits

Limits and requests	S queue	L queue	All queues
Per-user run limit	2	2	
Per-queue run limit	4	4	
Global per-user run limit			3
Requests	johndoe* johndoe* johndoe root* root* janedoe janedoe	root* janedoe* janedoe* janedoe root johndoe*	

You must set the maximum permanent file size for any batch queue residing on an Exemplar system machine. Recommended setting: (4194303 b)

Step 5 The maximum permanent file size controls the file size for any process in a running request. When a request is submitted to a queue, ConvexNQS+ checks the request limits to ensure the maximum permanent file size limit for the request does not exceed the maximum permanent file size limit for the queue. If it does exceed the maximum limit, ConvexNQS+ rejects the request.

Set the maximum permanent file size limit using the `set per_process cpu_limit` command. The command format is

`set per_process permfile_limit =` (*limit*) *queuename*

where

limit[*units*] is the maximum size a permanent file can be for any process in the running request in *limit* [*units*] format. Recommended setting: (4194303 b). *limit* can be any integer up to 8 digits. You can specify that no limit be applied by entering `unlimited`.

units can be any one of the following:

`b` bytes

`w` words

`kb` kilobytes (2^{10} bytes)

`kw`	kilowords (2^{10} words)
`mb`	megabytes (2^{20} bytes)
`mw`	megawords (2^{20} words)
`gb`	gigabytes (2^{30} bytes)
`gw`	gigawords (2^{30} words)

If you omit *units*, bytes is assumed.

queuename is the name of the queue for which you are setting the limit.

ConvexNQS+ generates an error if the command in Step 6 is applied to a batch queue residing on an Exemplar system.

Step 6 If you want to restrict queue access, use the `set no_access` command. The command format is

`set no_access` *queuename*

where *queuename* is the name of the queue for which you are restricting access.

Step 7 If you set the access restriction parameter to no access in Step 6, supply a list of users who may access the queue. Use the `add user` command. The command format is

`add users` = *user queuename*

where

user is one or more users who may access the queue. If you supply more than one user, separate items in the list with commas and surround the list with parentheses.

user can be either the user name or UID. Enclose a UID in square brackets [ ].

queuename is the name of the queue for which you are restricting access.

Step 8 If you set the access restriction parameter to no access in Step 6, supply a list of groups who may access the queue. Use the `add group` command. The command format is

`add group` = *group queuename*

where

group is one or more groups who may access the queue. If you supply more than one group, separate items in the list with commas and surround the list with parentheses

group can be either the group name or GID. Enclose a UID in square brackets [ ].

queuename is the name of the queue for which you are restricting access.

Step 9 Make sure all attributes for all queues have been correctly set. Use the `show long queue` command. The command format is

`show long queue` *queuename*

For example, to show the attributes for the queue named *b*, enter:

```
show long queue b
```

Figure 6 illustrates the output for this command.

Figure 6 Displaying batch queue attributes

```
Mgr: show long queue b
b@hostA; type=BATCH; [ENABLED, INACTIVE]; pri=48
 0 exit;   0 run;   0 stage;   0 queued;   0 wait; 0 hold;  0 arrive;
Run_limit = 1; Per-user run-limit = NONE
Accounting: Off
Activity ID offset: 0
Maximum request priority: 63
Cumulative system space time = 0.000000 seconds
Cumulative user space time = 0.000000 seconds
Unrestricted access
Import directory: Not available
Checkpoint: Not available
Copy open files on checkpoint: No
Share policy fixed = short
Per-process core file size limit = UNLIMITED
Per-process data size limit = UNLIMITED
Per-process permanent file size limit = UNLIMITED
Per-process execution nice value = 0 <DEFAULT>
```

Step 10 Fix any incorrect attributes using the set commands. See Chapter 6, "qmgr commands" for a complete list of these commands and their formats.

Step 11 Exit `qmgr` by entering:

```
exit
```

Step 12 If you configured any queues to import the current working directory for a request, edit the /etc/exports file on each machine to include entries for all file systems eligible for remote mounting.

For example, to specify exportation of the /usr and /mnt file systems on *hostA* to *hostB* and *hostC*, include the following lines in the /etc/exports file on *hostA*:

```
/usr -access=hostB:hostC
/mnt -access=hostB:hostC
```

If you are using a nameserver, you must supply the fully qualified machine name. Refer to the exports(5) man page for more information on the format of this file.

Step 13 After editing the exports file, initialize the list of exportable file systems using the exportfs command. From the shell prompt, enter:

```
exportfs -a
```

Step 14 Edit the /etc/hosts file on the local machine to include any remote machine names you are exporting from.

If you are using TCP/IP protocol, Figure 7 illustrates an example /etc/hosts file entry on *hostB*.

Figure 7 Example /etc/hosts file entry with TCP/IP protocol

```
130.168.71.160      hostA       # any comment
130.168.71.162      hostB       # any comment
```

Each line in this file represents one entry; each entry represents one machine. The format of the /etc/hosts file is:

internet_address official_name [*aliases* ...] [*#comment*]

where

internet_address
: is the official internet address for this machine.

official_name
: is the official name for this machine, as specified with the `hostname` program.

aliases
: is an unofficial name or list of unofficial names for this machine.

#comment
: is any comment you want about this machine.

If you are not using TCP/IP protocol, you must include an entry in this file for the local machine. Figure 8 illustrates an example /etc/hosts file entry on *hostA*.

Figure 8 Example /etc/hosts file entry without TCP/IP protocol

```
130.168.71.160hostAlocalhost
130.168.71.162hostB#any comment
```

Step 15 If you want to restrict access to file systems on a remote machine, skip to Step 16.

Edit the /etc/hosts.equiv file on the remote machine to include the names of the machines to which file systems will be imported. This makes it possible for all users to log into the machine without further password validation as long as the user has an account on that machine.

Figure 9 illustrates an example /etc/hosts.equiv file on *hostC*. Each line in this file represents one entry. Each entry represents one remote machine.

Figure 9 Example /etc/hosts.equiv file

```
hostA
hostB
```

Step 16 If you want to restrict access to file systems on a remote machine, tell those users to edit their .rhosts file on the remote machine to include the host names of the hosts to which file systems will be imported.

The .rhosts file has the same format as the /etc/hosts.equiv file described in Step 15. Refer to the rhost(5) man page for more details.

Step 17 During processing, ConvexNQS+ holds job output and transaction scripts in several directories in usr/spool/nqs before sending job output to the submitter's output file. If /usr/spool/nqs becomes full, jobs will fail.

Consider creating a separate disk partition for /usr/spool/nqs. Refer to the *SPP-UX System Administration Guide* (DSW-853) for details on setting up partitions.

Step 18 Start the `qmgr` utility by entering:

```
qmgr
```

Step 19 Enable all batch queues using the `enable queue` command. For example, to enable the batch queue named *b*, enter:

```
enable queue b
```

Step 20 Start all batch queues using the `start queue` command. For example, to start the batch queue named *b*, enter:

```
start queue b
```

Step 21 Exit `qmgr` by entering:

```
exit
```

Step 22 Repeat Step 1 through Step 21 on each machine running ConvexNQS+.

Creating pipe queues

Once you determine the queue structure needed to satisfy user requirements, you must add pipe queues to machines configured with ConvexNQS+.

Perform the following steps to add pipe queues:

Step 1 Log in as a ConvexNQS+ manager on the machine you are configuring.

Step 2 Decide if the new pipe queue will route jobs to the first destination that will accept the request, or to the first destination that can run the job immediately, or to the destination with the lowest load factor that is able to accept the request. Refer to Chapter 1, "ConvexNQS+ pipe clients," for more details.

Step 3 Start the `qmgr` utility by entering:

```
qmgr
```

The `qmgr` prompt appears:

```
Mgr:
```

Step 4 Create a pipe queue using the `create pipe_queue` command in `qmgr`. The command format is

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

where

queuename is the name of the queue. The name can consist of any printable nonblank character except for the at sign (@), a comma (,), an equal sign (=), and a left or right parenthesis ( ). The name cannot start with a digit.

priority is the interqueue priority for the queue. This priority affects queue selection for running the next job. Can be any number from 0 to 63; 0 is the lowest priority and 63 the highest.

server is the name of the pipe client that transports submitted requests to one of the destination queues. This can be:

`pipeclient` Routes the request to the first destination that will accept it. Destinations may reject a request due to queue limit violations or lack of account authorization. The full path name for this pipe client is /usr/lib/nqs/pipeclient.

`pipedemand` Routes the request to the first destination that can run the job immediately. The full path name for this pipe client is /usr/lib/nqs/pipedemand.

`pipeldav` Sorts the destination list by load factor and tries destinations with low load factors first. The full path name for this pipe client is /usr/lib/nqs/pipeldav.

Refer to the pipeclient(8) man page for more details.

destination is the name of destination queue(s) to which this pipe queue can route requests. If you supply more than one destination, separate items in the list with commas and surround the list with parentheses. For example, to designate three destinations, enter:

```
(destq1, destq2, destq3)
```

The syntax of the destination queue name must be in one of the following forms. Square brackets [ ] are required where shown

Figure 10 Destination queue name syntax examples.

```
local_queue_name
local_queue_name@local_machine_name
remote_queue_name@remote_machine_name
remote_queue_name@[remote_machine_mid]
```

Caution

If you specify a pipedemand pipe client, make sure all destination queues are defined with a demand attribute.

`pipeonly` indicates the queue can only accept requests from a pipe queue. Otherwise, the queue can accept requests from any source.

run_limit is the maximum number of pipe clients that may run simultaneously to deliver requests to their destination.

You must set the maximum permanent file size for any batch queue residing on an Exemplar system. Recommended setting: (4194303 b)

Step 5 The maximum permanent file size controls the file size for any process in a running request. When a request is submitted to a queue, ConvexNQS+ checks the request limits to ensure the maximum permanent file size limit for the request does not exceed the maximum permanent file size limit for the queue. If it does exceed the maximum limit, ConvexNQS+ rejects the request.

Set the maximum permanent file size limit using the set `per_process cpu_limit` command. The command format is

`set per_process permfile_limit` = (*limit*) *queuename*

where

limit is the maximum size a permanent file can be for any process in the running request in *limit* [*units*] format. Recommended setting: (4194303 b). *limit* can be any integer up to 8 digits. You can specify that no limit be applied by entering `unlimited`.

units can be any one of the following:

`b`	bytes
`w`	words
`kb`	kilobytes (2^{10} bytes)
`kw`	kilowords (2^{10} words)
`mb`	megabytes (2^{20} bytes)
`mw`	megawords (2^{20} words)
`gb`	gigabytes (2^{30} bytes)
`gw`	gigawords (2^{30} words)

If you omit *units*, bytes is assumed.

queuename is the name of the queue for which you are setting the limit.

Step 6 If you want to restrict queue access, use the `set no_access` command. The command format is

`set no_access` *queuename*

where *queuename* is the name of the queue for which you are restricting access.

Step 7 If you set the access restriction parameter to no access in Step 6, supply a list of users who may access the queue. Use the `add user` command. The command format is

`add users` = *user queuename*

where

user is one or more users who may access the queue. If you supply more than one user, separate items in the list with commas and surround the list with parentheses. *user* can be either the user name or UID. Enclose a UID in square brackets [ ].

queuename is the name of the queue for which you are restricting access.

Step 8 If you set the access restriction parameter to no access in Step 6, supply a list of groups who may access the queue. Use the `add group` command. The command format is

`add group` = *group queuename*

where

group is one or more groups who may access the queue. If you supply more than one group, separate items in the list with commas and surround the list with parentheses.

group can be either the group name or GID. Enclose a GID in square brackets [ ].

queuename is the name of the queue for which you are restricting access.

Step 9 Make sure all attributes for the queue have been correctly set. Use the `show long queue` command. The command format is

`show long queue` *queuename*

For example, to show the attributes for the queue named *best*, enter:

```
show long queue best
```

Figure 11 illustrates the output for this command.

Figure 11 Displaying pipe queue attributes

```
Mgr: show long queue best
best@hostA; type=PIPE; [ENABLED, INACTIVE]; pri=48
 0 depart;   0 route;   0 queued;   0 wait;   0 hold;   0 arrive;
Run_limit = 1;   Per-user run limit = NONE
Cumulative system space time = 0.000000 seconds
Cumulative user space time = 0.000000 seconds
Unrestricted access
Queue server: /usr/lib/nqs/pipeldav -w 1.0 hostC 2.0
Destset = [v@hostA,v@hostB,v@hostC]
```

Step 10 Fix any incorrect attributes using the `set` commands. Refer to Chapter 6, "`qmgr` commands," for a complete list of these commands and their formats.

Step 11 Enable the queue using the `enable queue` command. For example, to enable the pipe queue named *best*, enter:

```
enable queue best
```

Step 12 Start the queue using the `start queue` command. For example, to start the pipe queue named *best*, enter:

```
start queue best
```

Step 13 Repeat Step 4 through Step 12 for each pipe queue on this machine.

Step 14 Exit `qmgr` by entering:

```
exit
```

Step 15 Repeat Step 1 through Step 15 on each machine running ConvexNQS+.

Deleting queues

If you create a queue you no longer need, perform the following steps to delete it. The queue must be empty before you can delete it.

Step 1 Log in as a ConvexNQS+ manager on the machine you are configuring.

Step 2 Start the `qmgr` utility by entering:

```
qmgr
```

The `qmgr` prompt appears:

```
Mgr:
```

Step 3 You must disable the queue before deleting it. The command format is

`disable q` *queuename*

where *queuename* is the name of the queue you want to disable.

Step 4 You must stop the queue before deleting it. The command format is

`stop q` *queuename*

where *queuename* is the name of the queue you want to stop.

Step 5 Delete the queue using the `delete queue` command. The command format is

`delete q` *queuename*

where *queuename* is the name of the queue you want to delete.

Step 6 Exit `qmgr` by entering:

```
exit
```

Step 7 Repeat Step 1 through Step 6 on each machine running ConvexNQS+.

Configuring and activating ConvexNQS+ accounting

The ConvexNQS+ batch accounting system tracks the system resources used by an individual user or group by *request*.

It collects information according to the structure defined in the /usr/include/batch-acct.h file. Figure 12 illustrates the ConvexNQS+ accounting structure.

Figure 12 Example batch accounting structure from /usr/include/batch-acct.h file

```
struct batch_acct {
char quename[MAX_QUEUENAME+1]; /*the name of the batch queue*/
char host[MAXHOSTNAMELEN+1];  /*the host where the job ran*/
time_t submit_time; /*when the job was submitted */
time_t complete_time; /*when the job completed*/
uid_t uid; /*user's uid (see getuid(2))*/
gid_t gid; /*user's gid (see getgid(2))*/
long aid; /*activity id (see getaid(2))
long seqno; /*job sequence number */
char rhost[MAXHOSTNAMELEN+1];  /*originating host name */
short rpriority; /*request (intra-queue) priority
short qpriority; /*queue (inter-queue) priority*/
short nice; /*nice value */
struct rusage rusage; /*resource usage */
time_t start_time /*when the job was started*/
int reserved[7]; /*reserved for future use */
}
```

Once the accounting information is collected, the system manager can use the `qsa` utility to interpret it. See Chapter 5, "Generating accounting reports" for details on using `qsa`.

Perform the following steps to configure and activate the ConvexNQS+ accounting system:

Step 1 Log in as a ConvexNQS+ manager on a machine you are configuring.

Step 2 Start the `qmgr` utility by entering:

```
qmgr
```

The `qmgr` prompt appears:

```
Mgr:
```

Step 3 Identify the log file where accounting data is collected on this machine using the `set acc_logfile` command. For example, to identify a log file named /usr/adm/batchacct, enter:

```
set acc_logfile /usr/adm/batchacct
```

Step 4 Using the `show parameters` command, display queue parameters to verify the change. Figure 13 illustrates use of this command to check the accounting log file.

Figure 13 Checking the accounting log file

```
Mgr: show parameters


Accounting log file = /usr/adm/batchacct
```

Step 5 Activate ConvexNQS+ accounting for this queue using the `set accounting` command.

When accounting is activated for a queue, ConvexNQS+ saves batch job information in an accounting log file (if one is specified) and, if requested with the `qsub -me` command, sends mail to the user detailing what resources were used. For example, to activate accounting for the *s* queue enter:

```
set accounting=on s
```

Step 6 Determine the per-queue shell execution resource using the set `exec_shell_login` command.

If your jobs require customized resources, you may want to set this attribute to `Yes`. This causes any request submitted to this queue to be executed under your login shell, by sourcing your .cshrc and .login files. If you do not want the jobs to use your login shell, submit the job with `qsub -nl`.

If your jobs normally do not require customized resources but you may need them, you set this attribute to `Available`. This causes any request sent to this queue to *not* be executed under your login shell unless submitted with `qsub -l`.

If your jobs normally do not requre customized resources and you do not wish to allow this, set this attribute to `No`. None of the requests will be executed under a login shell.

The command format is

`set exec_shell_login=`*`Answer queuename`*

where

Answer can be one of the following:
Yes
No
Available

queuename is the name of the queue you are configuring.

For example, to set the attribute to allow per-queue shell execution of the user's .login and .cshrc files for the *s* queue, enter:

```
set exec_shell_login=Yes s
```

Step 7 Using the `show long queue` command, display queue attributes to verify the change. Figure 14 illustrates the use of this command to check if accounting is activated.

Figure 14 Viewing queue attributes to check accounting activation

```
Mgr: show long queue s
s@hostC; type=BATCH; [ENABLED, INACTIVE]; pri=48
0 exit; 0 run; 0 stage; 0 queued; 0 wait; 0 hold; 0 arrive;
                .
Accounting: On  :
                .
```

Step 8 Repeat Step 6 through Step 7 for each queue that requires ConvexNQS+ accounting on this machine.

Step 9 Exit `qmgr` by entering:

```
exit
```

Step 10 Repeat Step 1 through Step 9 on each machine running ConvexNQS+.

Establishing a shell strategy

A shell strategy determines the command interpreter used to interpret request script commands if a request is submitted to a queue without an identified shell interpreter. Perform the following steps to set the shell strategy.

Step 1 Log in as a ConvexNQS+ manager on the machine you are configuring.

Step 2 Start the `qmgr` utility by entering:

```
qmgr
```

The `qmgr` prompt appears:

```
Mgr:
```

Step 3 Set the shell strategy using the `set shell_strategy` command. The format for this command is

```
set shell_strategy option
```

where *option* can be one of the following:

`fixed=`*shell*	indicates the shell that interprets script file commands, where *shell* can be csh, ksh, or sh. Supply the absolute path name. The shell must exist and be executable or this command fails.
`free`	indicates the user's login shell (as defined in the /etc/passwd file) is used initially to interpret commands. ConvexNQS+ then supplies the name of the script file to the login shell as standard input. The user's login shell reads the first line of the script file. If the first line specifies a shell, ConvexNQS+ uses that shell to interpret the script file commands. Otherwise ConvexNQS+ uses sh.
`login`	indicates the user's default login shell (as defined in the /etc/password file) is used to interpret the script file commands.

For example, to set the shell strategy to free, enter:

```
set shell_strategy free
```

Step 4 Exit `qmgr` by entering:

```
exit
```

Step 5 Repeat these steps on each machine running ConvexNQS+.

Taking a qmgr snapshot

Save the current ConvexNQS+ configuration on each machine as a series of `qmgr` commands using the `qsnapshot` command.

This command saves the configuration to screen by default; however, you can save the configuration to a file and later use the contents of that file to restore the ConvexNQS+ `qmgr` configuration. For example, to save the new configuration to a file named batch_qconfig, enter:

```
qsnapshot > batch_qconfig
```

Figure 15 illustrates the output from `qsnapshot` command. For more information, refer to the qsnapshot(8), qmgr(8), and qmapmgr(8) man pages.

Figure 15 `qmgr` system snapshot

```
# qsnapshot
SET ACC_LOGFILE /dev/null
SET AID_MASK = 1
SET DEFAULT BATCH_REQUEST PRIORITY 31
SET DEFAULT DESTINATION_RETRY TIME 72
SET DEFAULT DESTINATION_RETRY WAIT 5
SET MAIL 0
SET GLOBAL PER_USER RUN_LIMIT = 4
SET MANAGERS root:m test:m janedoe:m johndoe:m
SET SHELL_STRATEGY LOGIN
CREATE PIPE_QUEUE best PRIORITY =48 SERVER = (usr/lib/nqs/pipeldav) RUN_LIMIT = 1
CREATE PIPE_QUEUE v PRIORITY = 48 SERVER = (usr/lib/nqs/pipeclient) RUN_LIMIT = 1
CREATE BATCH_QUEUE short PRIORITY = 48 RUN_LIMIT = 1 IMPORT_DIR = No SHARE_POLICY
FIXED = short
SET ACCOUNTING = Off short
SET COPY_OPEN_FILES = No short
SET NICE_VALUE_LIMIT = ( 0 ) short
SET PER_PROCESS PERMFILE_LIMIT = ( UNLIMITED ) short
SET MAXIMUM REQUEST_PRIORITY 63 short
SET PER_USER_RUN_LIMIT = 1 short
ADD ALIAS s short
```

Configuring error logging

When a ConvexNQS+ utility encounters an error, it writes a message directly to the user's terminal. A user can usually interpret these messages.

When daemons, such as nqsdaemon and netdaemon, encounter an error, they communicate with logdaemon. logdaemon handles error logging for daemons according to level of severity, and sends the error message to the syslog daemon at the syslog level shown in Table 3.

Table 3 Error severity levels

Error level	Logdaemon action	Syslog level
Fatal	Log error, write to stdout, mail operators	LOG_CRIT
Error	Log error, write to stdout	LOG_ERR
Warn	Log error, write to stdout	LOG_WARNING
Info	Log error, write to stdout	LOG_INFO
Log	Log error	LOG_INFO
Debug	Log error	LOG_DEBUG

`syslog` handles these messages as defined in the /etc/syslog.conf file. Perform the following steps to configure the /etc/syslog.conf file to your needs.

Step 1 Log in as the superuser on the system console for the machine you are configuring.

Step 2 Change the syslog.conf file. Each line in the syslog.conf file represents a message group. The format for this file is

```
facility.level send_message_here
```

where

facility is the part of the system that generates the message. Enter `local0` .

level describes the error level of the message. This can be any error level listed in Table 3. When you choose an entry, the system records errors for that entry level and all preceding entry levels in the chart. For example, if you choose `Info`, you receive error messages from `Info`, `Warn`, `Error`, and `Fatal`.

send_message_here can be one of the following:

- Absolute path name of a file—writes messages to the named file. The file named here must exist before messages can be logged to it. Be sure to complete Step 3.
- Hostname preceded with an at sign (@)—forwards messages to the named site.
- A list of users separated by commas. These users receive the messages if they are logged in.
- An asterisk—sends messages to all users logged in.

For example, to log all batch errors of levels preceding and including debug to /usr/adm/batchlog on a CONVEX C Series, enter the following line in /etc/syslog.config:

```
batch.debug /usr/adm/batchlog
```

To accomplish the same logging into an Exemplar system machine, enter the following line in /etc/syslog.config:

```
local0.debug /usr/adm/batchlog
```

Refer to the syslogd(8) man page for more details on how to set up syslog.conf.

Step 3 If you supply a path name in the *send_message_here* field of the syslog.conf file, create those files using the `touch` command. For example, create /usr/adm/batchlog by entering:

```
touch /usr/adm/batchlog
```

Step 4 Reinitialize the syslog daemon, syslogd, by entering:

```
kill -HUP 'cat /etc/syslog.pid'
```

Step 5 Check the /etc/rc.local file to make sure the following line, which automatically starts the syslogd daemon each time the system is booted, exists.

```
/usr/etc/syslogd & echo 'starting system logger'
```

If it does not exist, add it using an editor.

Step 6 If you included debug messages for logging, decide on the debug level. The system sends significant debug messages to logdaemon only if the debug level is greater than 0. Figure 16 illustrates the level of logging if the debug level is greater than 0.

Figure 16 Logging with debug level greater than zero

```
04/21/90 12:12 ConvexNQS+(DEBUG): main(): Configuration loaded.
04/21/90 12:12 ConvexNQS+(DEBUG): main(): Rebuild queue state.
04/21/90 12:12 ConvexNQS+(DEBUG): main(): Queue state rebuilt.
04/21/90 12:12 ConvexNQS+(DEBUG): main(): Enabling virtual timers.
04/21/90 12:12 ConvexNQS+(DEBUG): main():Looping to read request packets
```

Step 7 If you want to change the debug level, log in as a ConvexNQS+ manager.

Step 8 Start the `qmgr` utility by entering:

```
qmgr
```

The `qmgr` prompt appears:

```
Mgr:
```

Step 9 Set the debug level using the `set debug` command. The command format is

`set debug` *level*

where *level* can be one of the following:

`0`	No debugging information is displayed.
`1`	Minimum debugging information is displayed.
`2`	Maximum debugging information is displayed.

Step 10 Repeat Step 1 through Step 9 on each machine running ConvexNQS+.

Automating queue operations

You can automatically control starting, stopping, and aborting queues using the `cron` utility.

`cron` runs commands at specified dates and times according to the instructions in the /.crontab file.

Refer to the crontab(5) man page for detailed information on using the cron utility and setting up the crontab file.

Notifying users of changes

After you have configured ConvexNQS+, notify your users of changes and recommendations for the new ConvexNQS+ system.

Remote access capabilities

When ConvexNQS+ is configured on more than one machine, a user can submit jobs to a remote machine if he of she has access privileges on the remote machine.

There are two ways to provide access to a remote machine:

- Edit the /etc/hosts.equiv file on the user's machine to include the remote machine.
- Create a .rhosts file in the user's home directory that includes the remote machine.

Tell your users if they should create a .rhosts file in their home directory.

Miscellaneous information

Also, furnish users with the following information:

- Names and aliases of any ConvexNQS+ queue
- Default shell strategy established for the ConvexNQS+ system
- Import attribute setting for each ConvexNQS+ queue
- Manager and operator privileges assignments
- Default limits for each ConvexNQS+ queue
- Hours that each ConvexNQS+ queue accepts and runs requests

Controlling queue operations 3

Several commands are available for controlling queue operation, including commands for:

- Aborting running requests in a queue
- Purging non-running requests from a queue
- Moving requests to another queue
- Disabling a queue
- Enabling a queue
- Starting a queue
- Stopping a queue

This chapter describes how to perform each of these tasks.

Getting started

Start the `qmgr` utility by logging in as a ConvexNQS+ manager or operator. Then enter:

```
qmgr
```

The following prompt appears:

```
Mgr:
```

Removing requests

You can remove requests from a queue by:

- Aborting all running requests
- Purging all non-running requests
- Moving requests to another queue

Each of these actions is described in the following sections.

Aborting requests

Use the `abort queue` command to abort all running requests in a queue.

ConvexNQS+ sends a SIGTERM signal to each process running in the queue, then sends a SIGKILL signal to any process that continues to run after the SIGTERM signal.

ConvexNQS+ removes all aborted requests from the queue and returns all output files associated with the requests to the appropriate destination.

The command format is

`abort queue` *queuename* [*seconds*]

where

queuename	is the name of the queue to be aborted.
seconds	is the number of seconds to wait before executing the SIGKILL signal after the SIGTERM signal is sent. If you don't supply a *seconds* value, ConvexNQS+ delays 60 seconds.

Purging requests

Use the `purge queue` command to drop all non-running requests from a queue. ConvexNQS+ completes all running requests. The purged requests are irretrievable.

The command format is

`purge queue` *queuename*

where *queuename* is the name of the queue to be purged.

Moving requests

Use the `move queue` command to move all non-running requests to another queue.

ConvexNQS+ moves requests regardless of queue limit violations, access restrictions, or attribute violations.

The command format is

`move queue` *queuename des_queue*

where

queuename	is the name of the queue whose requests are to be moved.
des_queue	is the destination queue for the moved requests.

Disabling and enabling queues

You must enable a queue before the queue can accept requests for processing. You must disable it to prevent it from accepting requests.

This section describes how to enable and disable queues.

Enabling queues

Use the `enable queue` command to allow a queue to accept requests for processing.

The command format is

`enable queue` *queuename*

where *queuename* is the name of the queue to be enabled.

Disabling queues

Use the `disable queue` command to prevent a queue from accepting requests.

The command format is

`disable queue` *queuename*

where *queuename* is the name of the queue to be disabled.

Starting and stopping queues

You must start a queue before the queue can process accepted requests. You must stop it to prevent it from processing accepted requests.

Starting queues

Use the `start queue` command to allow a queue to process requests.

The command format is

`start queue` *queuename*

where *queuename* is the name of the queue to be started.

Stopping queues

Use the `stop queue` command to prevent it from processing requests.

ConvexNQS+ completes currently running requests but queues all non-running requests. Users may still submit new requests; however, they are also queued.

The command format is

`stop queue` *queuename*

where *queuename* is the name of the queue to be stopped.

Controlling requests

4

Several commands are available for controlling requests, including the following commands for:

- Deleting a specific request
- Delaying a queued request
- Delaying a running request
- Moving a specific non-running request
- Changing the priority of a non-running request
- Forcing a queued request to run

This chapter describes how to perform each of these tasks.

Getting started

To perform most of the commands described in this chapter, you must know

- A request's unique identifier
- How to start the `qmgr` utility

Determining a request ID

You can display a request's identifier and other request information using the `qstat` command. The command format is

`qstat` [*option* ...] [*queuename*[*@hostname*] ...]

where

option controls the type and amount of information displayed. If no options are specified, `qstat` shows only those requests belonging to the user issuing the command. *option* can be one or more of the following:

`-a` Displays status for all requests in the queue.

`-l`	Displays additional information about the queue and requests.
`-m`	Displays the date and time requests are scheduled to run.
`-u` *username*	Displays only those requests belonging to the specified *username*.
`-x`	Displays additional information about the queue.
queuename	is the name of the queue for which you want status information. If you do not supply a queue, ConvexNQS+ displays information for all queues on the requested machine.
hostname	is the name of the machine that receives the request. If *hostname* is omitted, ConvexNQS+ assumes the local machine.

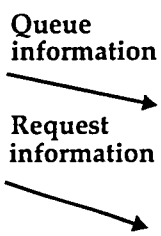
Refer to the `qstat`(1) man page or *ConvexNQS+ User's Guide for Exemplar systems* for more details.

For example, to display standard output for requests in the *v* queue on the local host, enter:

```
qstat v
```

Figure 17 illustrates the output for this command.

Figure 17 `qstat` standard output




```
% qstat v
verylong@hostC; type=BATCH; [ENABLED, RUNNING]; pri=16
aliases: v, verylong_queue, V, VERYLONG
 0 exit; 1 run; 0 stage; 0 queued; 0 wait; 0 hold; 0
arrive;

REQUEST NAME    REQUEST ID   USER     PRI   STATE       PGRP
1:myjob         47.mach2     test     31    RUNNING     12103
```

`qstat` displays two types of information:

- Information on the queue
- Information on each request in the queue

Information important to the actions described in this chapter is *request* information. This information is described below:

```
REQUEST NAME    REQUEST ID   USER     PRI   STATE       PGRP
1:myjob         47.mach2     test     31    RUNNING     12103
```

REQUEST NAME
: Name assigned to the request.

REQUEST ID
: Unique identifier assigned to request when it is submitted to the queue.

USER
: User submitting the request.

PRI
: Intraqueue priority assigned to request. This priority affects which request runs next in a queue. Priority can be any number from 0 to 63; 0 is the lowest priority and 63 the highest.

STATE
: State of the request. This can be:

  ARRIVING
  : Request is arriving at the queue.

  DEPARTING
  : Request is departing from the queue but has not yet been received by the destination queue.

  EXITING
  : Request has completed running and will exit from the system after ConvexNQS+ returns the required output files to their intended destinations.

  HOLDING
  : A hold has been placed on the request, preventing it from entering any other state.

  QUEUED
  : Request is queued and eligible for running or routing. This is the most common state.

  ROUTING
  : Request has reached the head of a pipe queue and is being routed to another queue.

  RUNNING
  : Request has reached its final destination batch queue and is running.

  WAITING
  : Request is waiting for a specified amount of time to pass before trying to run. This could be because it was submitted with a

future date and time specified for running, or because a pipe queue could not route the request but will try later.

`SUSPENDED`
Request is suspended from running. It will remain suspended until manually resumed.

`PGRP` Process group of the request, if available to the local ConvexNQS+ daemon. This information is displayed only for processes that are running.

For more information on queue or request properties, refer to the qstat(1) man page.

Starting qmgr

Start the `qmgr` utility by logging in as a ConvexNQS+ manager or operator. Then enter:

```
qmgr
```

The following prompt appears:

```
Mgr:
```

Deleting specific queue requests

The following two commands delete a specific request(s) from a queue:

- The `qmgr` utility `delete request` command
- The ConvexNQS+ `qdel` command

Each command is described in the following sections.

Using the delete request command

The `delete request` command is a `qmgr` utility command, which means you must start `qmgr` before you can use this command.

The command format is

`delete request` *request_id* [*request_id ...*]

where *request_id* is the number assigned to the request when it is submitted to ConvexNQS+. Use the `qstat` command to find the *request_id*.

You can specify more than one request.

You can specify a running or non-running request. If a request is running, ConvexNQS+ sends a SIGKILL signal to all processes in the request.

ConvexNQS+ removes deleted requests and discards them.

Using the qdel command

The `qdel` command is a ConvexNQS+ command that runs from a shell command line. The command format is

`qdel` [*option*] *request_id* [@*hostname*] [*request_id* [@*hostname*] ...]

where

option can be one of the following:

`-u` *username*

Lets you delete requests other than your own, where *username* is the name of the user who owns the request. By default, only the user who submitted the request can delete it from a queue. This option lets the superuser, or ConvexNQS+ manager or operator delete someone else's request from a queue.

`-k`

Sends a SIGKILL (-9) signal to a running request. When the request exits, ConvexNQS+ deletes it.

sig

Sends a specified signal to a running request. *sig* can either be the signal number or signal name in the /usr/include/signal.h file.

request_id The number assigned to the request when it is submitted to ConvexNQS+. Use the `qstat` command to find the *request_id*.

You can specify more than one request.

You can specify a running or non-running request.

If you are using the -u option, the *request_id* must belong to the user defined in *username*.

hostname

The name of the machine where the queue containing the request resides. If you omit *hostname*, ConvexNQS+ assumes the local host.

For example, a user with batch operator privileges enters the following command to delete the request identified as *291* on the local machine submitted by user *smith*.

```
qdel -u smith 291
```

Delaying queued requests

Use the `hold request` and `release request` commands to delay the running of a queued request.

If a ConvexNQS+ manager or operator places the request on hold, only a ConvexNQS+ manager or operator can release the hold.

The `hold request` and `release request` commands are `qmgr` utility commands, which means you must start `qmgr` before you can use these commands.

Each command is described in the following sections.

Placing a queued request on hold

Use the hold request command to place a request on hold, thereby preventing it from running. The command format is

`hold request` *request_id* [*request_id ...*]

where *request_id* is the number assigned to the request when it is submitted to ConvexNQS+. Use the `qstat` command to find the *request_id*.

You can specify more than one request.

You must specify a request in the queued state.

Releasing the hold on a queued request

Use the `release request` command to release the hold on a queued request, making it eligible for execution. The command format is

`release request` *request_id* [*request_id ...*]

where *request_id* is the number assigned to the request when it is submitted to ConvexNQS+. Use the `qstat` command to find the *request_id*.

You can specify more than one request.

You must specify a request in the holding state.

Delaying running requests

Use the `suspend request` and `resume request` commands to delay the running of a running request.

The `suspend request` and `resume request` commands are `qmgr` utility commands, which means you must start `qmgr` before you can use these commands.

Each command is described in the following sections.

Moving specific non-running requests to another queue

Use the `move request` command to move a specific non-running request from one queue to another.

The `move request` command is a `qmgr` utility command, which means you must start `qmgr` before you can use this command.

The command format is

`move request` *request_id* [*request_id ...*] *queue*

where

request_id — is the number assigned to the request when it is submitted to ConvexNQS+. Use the `qstat` command to find the *request_id*.
You can specify more than one request.
You must specify a non-running request.

queue — is the name of the queue to which you want to move the request.

If the request violates any queue limits, access restrictions, or attributes in the receiving queue, ConvexNQS+ does not move the request.

To move a running request, first suspend the request, move it, then resume the suspended request. Refer to the "Delaying running requests" section in this chapter for details.

Changing the priority of non-running requests

Use the `modify request` command to change the priority of a non-running request. The `modify request` is a `qmgr` utility command, which means you must start `qmgr` before you can use this command.

The command format is

`modify request priority =` *value request_id* [*request_id ...*]

where

value — is the new priority of the queue request. This is a number between 0 and 63; 0 is the lowest priority and 63 the highest. A user can only decrease the priority of a request. A ConvexNQS+ manager or operator can raise a request's priority.

request_id — is the number assigned to the request when it is submitted to ConvexNQS+. Use the `qstat` command to find the *request_id*.
You can specify more than one request.
You must specify a non-running request.

Forcing requests to run

There are two commands to force a queued request to begin running immediately:

- The ConvexNQS+ `qrun` command
- The `qmgr` utility `run request` command

Each command is described in the following sections.

Using the qrun command

The `qrun` command is a ConvexNQS+ command that runs from a shell command line. The command format is

`qrun request` *request_id* [*request_id ...*]

where *request_id* is the number assigned to the request when it is submitted to ConvexNQS+. Use the `qstat` command to find the *request_id*.

You can specify more than one request.

If running the request exceeds the current run limit of the queue, ConvexNQS+ increases the queue's run limit by one until the request finishes running.

Using the run request command

The `run request` command is a `qmgr` utility command, which means you must start `qmgr` before you can use this command.

The command format is

`run request` *request_id* [*request_id ...*]

where *request_id* is the number assigned to the request when it is submitted to ConvexNQS+. Use the `qstat` command to find the *request_id*.

You can specify more than one request.

If running the request exceeds the current run limit of the queue, ConvexNQS+ increases the queue's run limit by one until the request finishes running.

Request flow

The remaining sections in this chapter describes the flow of a request through each of the following queue types:

- Local batch queue
- Remote batch queue
- Local pipe queue
- Remote pipe queue

Local batch queue processing

You submit a request to a local batch queue using the `qsub` command.

`qsub` sends the request to the nqsdaemon. The nqsdaemon:

- Checks any qualifiers included with the `qsub` command against defined queue limits and attributes
- Checks the ability of the queue to import files
- Determines whether the recipient queue is a pipe-only queue
- Accepts or rejects the request based on these checks

When the nqsdaemon determines it is time to run an accepted request, it spawns a shepherd process that:

- Sets up the environment for the request
- Runs the job
- Sends mail to the user if it cannot run the shell script submitted
- Waits for the job to complete
- Logs accounting information
- Undoes anything it set up to run the job (such as unmounting remote file systems mounted to import files)
- Returns output files to the user's directory

Remote batch queue processing

You submit a request to a remote batch queue using the `qsub` command.

`qsub` sends the request to the netdaemon on the remote machine.

The netdaemon spawns a netserver that tries to queue the request with the nqsdaemon.

The nqsdaemon:

- Checks to make sure the user has an account on the remote machine
- Checks to make sure the user name on the local machine matches the user name on the remote machine

- Checks any qualifiers included with the `qsub` command line against the defined queue limits and attributes
- Checks the ability of the queue to import files
- Determines if the recipient queue is a pipe-only queue
- Accepts or rejects the request based on these checks

When the nqsdaemon determines it is time to run an accepted request, it spawns a shepherd process that:

- Sets up the environment for the request
- Runs the job
- Sends mail to the user if it cannot run the shell script submitted
- Waits for the job to complete
- Logs accounting information
- Undoes anything it set up to run the job (such as unmounting remote file systems mounted to import files)
- Returns output files to the user's directory

Local pipe queue processing

You submit a request to a local pipe queue using the `qsub` command.

`qsub` sends the request to the nqsdaemon.

The nqsdaemon:

- Checks the `qsub` qualifiers against defined queue limits and attributes
- Determines whether the recipient queue is a pipe-only queue
- Accepts or rejects the request based on these checks
- Routes an accepted request when it reaches the top of the queue by spawning the pipe client

The pipe client selects a destination queue for the request based on characteristics of:

- The request
- Each queue in the destination set defined for the pipe queue

If the pipe client finds a suitable destination on a remote machine, it contacts the netdaemon on the destination machine and sends the request to it. The netdaemon spawns the netserver, which in turn tries to queue the request with the local nqsdaemon.

If the pipe client does not find a suitable destination, it returns an appropriate transaction code to nqsdaemon.

Remote pipe queue processing

You submit a request to a remote pipe queue using the `qsub` command.

`qsub` sends the request to the netdaemon on the remote machine.

The netdaemon spawns a netserver that tries to queue the request with the nqsdaemon.

The nqsdaemon:

- Checks to make sure the user name on the local machine matches the user name on the remote machine
- Accepts or rejects the request based on this check
- Routes an accepted request when it reaches the top of the queue by spawning the pipe client

The pipe client selects a destination queue for the request based on characteristics of:

- The request
- Each queue in the destination set defined for the pipe queue

If the pipe client finds a suitable destination on a remote machine, it contacts the netdaemon on the destination machine and sends the request to it. The netdaemon spawns the netserver, which in turn tries to queue the request with the local nqsdaemon.

If the pipe client does not find a suitable destination, it returns an appropriate transaction code to nqsdaemon.

Generating accounting reports 5

The ConvexNQS+ batch accounting systems tracks the system resources used by an individual user or group by *request*.

Implementing ConvexNQS+ batch accounting involves identifying a log file to collect ConvexNQS+-specific accounting data, and then enabling batch accounting on a per-batch-queue basis. See Chapter 2, the section titled "Configuring and activating ConvexNQS+ accounting" for more information.

Once you implement ConvexNQS+ batch accounting, you can generate accounting reports for:

- An entire batch system
- Specific users
- Specific queues
- Specific users in specific queues

This chapter describes how to generate ConvexNQS+ accounting reports. Perform the following steps to generate ConvexNQS+ reports:

Step 1 Log in. You may need to log in as root, depending on the permissions set on the accounting log file.

Step 2 Generate a report using the `qsa` command. The command format is

`qsa` *mode* [*constraints*][*acct-file*]

where

mode	is the desired output. This can be one of the following:
`-r`	Raw mode Formats each record that matches the specified constraints and writes it to the user's stdout. Figure 18 illustrates one record from raw mode output.

Figure 18 qsa raw mode sample output

```
% qsa -r -q l
queue long host mach1
sub time 643564104 com time 643564107
uid 16 gid 49 aid 0
seqno 224 rhost mach1 rprio -1 qprio 4 nice 0
usertime 0.076545 systime 0.151084
io 15
```

`-x` Extended mode
Formats each record that matches the specified constraints and writes it to user's stdout. It differs from raw mode in that it adds time spent waiting in queues, time spent executing, and time between submission and completion; and it converts all time values to ASCII. Figure 19 illustrates one record from extended mode output.

Figure 19 qsa extended mode sample output

```
% qsa -x -q l
queue long host mach1
sub time Thu May 24 10:28:24 1990
com time Thu May 24 10:28:27 1990
sta time Thu May 24 10:28:24 1990
user test group dev aid 0
seqno 224 rhost mach1 rprio -1 qprio 4 nice 0
turnaround 3 secs
waited
ran 3 secs
user time 0.076545 system time 0.151084
io 15
```

`-s` Summing mode
Processes each record that matches the specified constraints and appends to stdout totals for CPU time consumed, user CPU time consumed, system CPU time consumed, and I/O operations performed. Figure 20 illustrates one record from summing mode output.

Figure 20 qsa summing mode sample output

```
% qsa -s
in 17 records from file /usr/adm/batchacct
total turnaround 2242 secs
total execution 2221 secs
total user time 1076.766425 secs
total system time 386.580745 secs
total io 19877 operations
```

`-a` Averaging mode
Processes each record that matches the specified constraints and appends to stdout averages for CPU time consumed, user CPU time consumed, system CPU time consumed, I/O operations performed, time spent waiting in queue, time spent executing, and time between submission and completion. Without the -q option, gives averages for every queue on your system in which accounting is activated. Figure 21 illustrates one record from averaging mode output.

Figure 21 qsa averaging mode sample output

```
% qsa -a
in 17 records from file /usr/adm/batchacct
average turnaround 131.882355 secs
average execution 130.647064
average user time 63.306437
average system time 22.757830
average io 1169.235352 operations
```

constraints controls the records selected for processing. You can specify records by queue, by user, or both. If you omit this option, qsa processes all records. Constraints can be one or more of the following:

`-Q` Processes records in all queues. ConvexNQS+ groups records by each queue that appears in the accounting file. You cannot use this option with the `-q` flag.

`-q` *queuename*

Processes records in a specified queue where *queuename* can be one or more queues. ConvexNQS+ groups records by queues specified in one or more occurrences of this option. You cannot use this option with a `-Q` flag.

If you do not supply a queue name, `qsa` displays accounting information for all queues listed in the accounting file.

`-u` Processes records for all users where *username* can be one or more names of users. ConvexNQS+ groups records by each user that appears in the accounting file. You cannot use this option with the `-u` flag.

`-u` *username*

Processes records for specific users. ConvexNQS+ groups records by users specified in one or more occurrences of this option. You cannot use this option with the `-U` flag.

If you do not supply a *username*, `qsa` displays accounting information for queues enabled for ConvexNQS+ batch accounting.

acct_file is the name of the log file where the ConvexNQS+ accounting data is stored. If you do not supply a file name, ConvexNQS+ uses `/usr/adm/batchacct`.

Refer to the qsa(8) man page for more information.

qmgr commands 6

qmgr reference pages

This chapter contains a description of each `qmgr` command. Use these commands to configure and operate ConvexNQS+ on the local machine.

You can enter commands in any combination of uppercase and lowercase characters.

The first two or three characters of each word in each command are unique, and you need to enter only those characters for ConvexNQS+ to recognize the command. These accepted abbreviations are indicated in the "Synopsis" section of each command description as uppercase letters.

ConvexNQS+ man pages

In addition, the following online man pages are available for ConvexNQS+:

- batch-acct(5)
- nqsdaemon(8)
- pipeclient(8)
- qdel(1)
- qjlist(1)
- qlimit(1)
- qmapmgr(8)
- qmgr(8)
- qps(1)
- qrun(8)

- qsa(8)
- qsnapshot(8)
- qstat(1)
- qsub(1)

abort queue

Authorization

ConvexNQS+ manager or operator privileges are required to use this command.

Description

Removes all running requests in a queue.

ConvexNQS+ sends a SIGTERM signal to each process of each request running in the queue. After the time indicated in *seconds* has passed, ConvexNQS+ also sends a SIGKILL signal to all remaining processes.

All running requests are lost when the queue is aborted; therefore, you must suspend and then resume any running requests you want to save before you abort the queue.

You can automatically abort queues using the cron utility. Refer to the crontab(5) man page for detailed information on using the cron utility and setting up the crontab file.

Syntax

`ABort Queue` *queuename* [*seconds*]

Parameter	Meaning
queuename	Name of the queue to abort.
seconds	Amount of real time ConvexNQS+ waits after sending a SIGTERM signal to send a SIGKILL signal. If you omit *seconds*, ConvexNQS+ assumes a default value of 60 seconds.

add alias

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Adds an alternate name to a queue.

You can use the queue name or any alias assigned to the queue to reference the queue.

Syntax

`ADd Alias` *alias queuename*

Parameter	Meaning
alias	Unique, alternate name to add.
queuename	Name of the queue to which the alias is assigned.

add destination

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Adds one or more destination queues to a pipe queue destination set.

If you supply more than one destination, separate items in the list with commas and surround the list with parentheses. For example, to designate three destinations, enter `(`*destq1, destq2, destq3*`)`.

Syntax

`ADd DEStination =` *destination queuename*

Parameter	Meaning
destination	Name of the destination queue to add. The syntax of the destination queue name must be in one of the following forms. Where shown, square brackets [ ] are required. *local_queue_name* *local_queue_name@local_machine_name* *remote_queue_name@remote_machine_name* *remote_queue_name@[remote_machine_mid]* You must define all remote machine names and machine IDs in the local system database. See Chapter 2, the section "Installing the base ConvexNQS+ system," for details on how to do this.
queuename	Name of the queue to which you are adding destinations.

add groups

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Adds one or more groups to the list of groups allowed access to a queue.

If you supply more than one group, separate items in the list with commas and surround the list with parentheses. For example, to designate three groups, enter `(`*group1*`,` *group2*`,` *group3*`)`.

You must set the queue for no access before adding groups. See the `set no_access` command for details on how to do this.

Syntax

`ADd Groups =` *group queuename*

Parameter	Meaning
group	Name or GID of the group to add. Enclose a GID in square brackets [ ].
queuename	Name of the queue to which access is granted.

add managers

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Grants one or more users access to `qmgr` commands.

Syntax

`ADd Managers` *username:option* [*username:option* ...]

Parameter	Meaning
username	Name of the user to receive access. The syntax of *username* must be in one of the following forms. Where shown, square brackets [ ] are required. *local_account_name* *[local_account_id]* *[remote_user_id]@ remote_machine_name* *[remote_user_id]@[remote_machine_mid]* You must define all remote machine names and IDs in the local system database. See Chapter 2, the section "Installing the base ConvexNQS+ system," for details on how to do this.
option	Indicates if the user receives manager or operator privileges. `m` grants manager access; `o` grants operator access.

add sender

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Adds one or more pipe queues to the list of queues that can send requests to a demand queue (batch queue created with the demand attribute).

If you supply more than one pipe queue, separate items in the list with commas and surround the list with parentheses. For example, to designate three pipe queues, enter (*sendq1,sendq2,sendq3*).

Syntax

`ADd Sender` = *sender queuename*

Parameter	Meaning
sender	Name of the pipe queue to add. The syntax of the pipe queue name must be in one of the following forms. Where shown, square brackets [ ] are required. *local_queue_name* *local_queue_name@local_machine_name* *remote_queue_name@remote_machine_name* *remote_queue_name@[remote_machine_mid]* You must define all remote machine names and machine IDs in the local system database. See Chapter 2, the section "Installing the base ConvexNQS+ system," for details on how to do this.
queuename	Name of the demand queue to which you are adding senders.

Restriction

When the demand batch queue notifies senders, it accepts jobs on a first-come, first-served basis, which is generally the first sender in the list.

add users

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Adds one or more users to the list of users allowed access to a queue.

If you supply more than one user, separate items in the list with commas and surround the list with parentheses. For example, to designate three users, enter `(`*user*`1, `*user*`2, `*user*`3)`.

You must set the queue for no access before adding users. See the `set no_access` command for details on how to do this.

Syntax

`ADd Users =` *user queuename*

Parameter	Meaning
user	Name or UID of the user to add. Enclose a UID in square brackets [ ].
queuename	Name of the queue to which access is granted.

create batch_queue

Authorization

Manager privileges are required to use this command.

Description

Creates a new ConvexNQS+ batch queue.

Syntax

`CReate Batch_queue` *queuename* `PRiority=`*priority* [`PIpeonly`][`Import_dir=`*import-option*][`Run_limit=`*run-limit*] [`Demand`][`SEnder=`*sender*][`SUbcomplex =`*subcomplex name*]

where

queuename — Name of the new batch queue. The name can consist of any printable nonblank character except for the at sign (@), a comma (,), an equal sign (=), and a left or right parenthesis ( ). The name cannot start with a digit.

priority — Interqueue priority for the queue. This priority affects queue selection for running the next job. Can be any number from 0 to 63; 0 is the lowest priority and 63 the highest.

`PIpeonly` — Indicates the queue can only accept requests submitted from a pipe queue. Otherwise, the queue can accept requests from any source (such as a program or script file).

import_option — Indicates if ConvexNQS+ imports the current working directory for a request (mounts the directory on the machine processing the request) before running the request. *import-option* can be:

`Yes`
: ConvexNQS+ automatically imports the current working directory for any request running in the queue. A user can override this setting for individual requests using the -ni option of `qsub`.

`Available`
: Lets a user specify importation of the current working directory for requests submitted to the queue using the -i option of `qsub`.

No ConvexNQS+ does not import the current working directory for requests submitted to the queue. Furthermore, the queue rejects requests that require imported directories.

run-limit Maximum number of requests that can run in the queue at any given time. If you do not supply a run limit, the value defaults to 1.

`Demand` indicates that requests may enter this queue only if they can execute immediately (which is determined by *run-limit*). `Demand` implies `PIpeonly`.

sender is the name of the pipe queue(s) that can send requests to this batch queue, and applies only if `Demand` is specified. If you supply more than one sender, separate items in the list with commas and surround the list with parentheses. For example, to designate three destinations, enter:

(sendq1, sendq2, sendq3)

The syntax of the sender queue name must be in one of the following forms. Square brackets [ ] are required where shown.

local_queue_name

local_queue_name@local_machine_name

remote_queue_name@remote_machine_name

remote_queue_name@[remote_machine_mid]

Note

Make sure all senders are defined with a server=(/usr/lib/nqs/pipedemand) attribute.

`SUbcomplex` = *subcomplex name*

Causes all requests run from this queue to execute on the named subcomplex. The subcomplex must be a valid and currently loaded subcomplex when the queue is created and started. A subcomplex can be configured using the `scm` utility or by using `qmgr` COnfig_subcomplex. `COnfig_subcomplex` executes the `scm` utility.

Note

The subcomplex permissions should be set to reflect the user and group set intended for use on a subcomplex batch queue.

create pipe_queue

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Creates a new ConvexNQS+ pipe queue.

Syntax

`CReate Pipe_queue` *queuename* `PRiority=`*priority* `Server=`*(server)* [`Destination=`*destination*] [`PIpeonly`] [`Run_limit=`*run-limit*]

Parameter	Meaning
queuename	Name of the new pipe queue. The name can consist of any printable nonblank character except for the at sign (@), a comma (,), an equal sign (=), and a left or right parenthesis ( ). The name cannot start with a digit.
priority	Interqueue priority for the queue. This priority affects queue selection for running the next job. This can be any number from 0 to 63; 0 is the lowest priority and 63 the highest.

server	Name of the pipe client that transports submitted requests to one of the destination queues. *(server)* can be:
	
	`pipeclient`
	Routes the request to the first destination that will accept it. Destinations may reject a request due to queue limit violations or lack of account authorization. The full path name for this pipe client is /usr/lib/nqs/pipeclient.
	
	`pipedemand`
	Routes the request to the first destination that can run the job immediately. The full path name for this pipe client is /usr/lib/nqs/pipedemand.
	
	`pipeldav`
	Sorts the destination list by load factor and tries destinations with low load factors first. The full path name for this pipe client is /usr/lib/nqs/pipeldav.

Refer to the pipeclient(8) man page for more details.

destination — Name of one or more destination queues to which this pipe queue can route requests. If you supply more than one destination, separate items in the list with commas and surround the list with parentheses. For example, to designate three destinations, enter (*destq1*, *destq2*, *destq3*).

The syntax for the destination queue name must match one of the following forms. Where shown, square brackets [ ] are required.

local_queue_name

local_queue_name@local_machine_name

remote_queuename@remote_machine_name

remote_queuename@[remote_machine_mid]

You must define all remote machine names and IDs in the local system database. See Chapter 2, the section "Installing the base ConvexNQS+ system," for details on how to do this.

Note

If you specify a pipedemand pipe client, make sure all destination queues are defined with a demand attribute.

`PIpeonly` — Indicates the queue can only accept requests submitted from a pipe queue. Otherwise, the queue can accept requests from any source.

run-limit — Maximum number of pipe clients that may run simultaneously to deliver requests to their destination. If you do not supply a run limit, ConvexNQS+ defaults to 1.

delete alias

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Deletes an alternate name for a queue.

Syntax

`DElete Alias` *alias*

Parameter	Meaning
alias	Alternate name to delete.

delete destination

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Deletes one or more destination queues from a pipe queue destination set.

If you supply more than one destination, separate items in the list with commas and surround the list with parentheses. For example, to designate three destinations, enter (*destq1, destq2, destq3*).

ConvexNQS+ successfully completes any request being transferred to the deleted destination before deleting the destination.

If you delete all destinations for the pipe queue, the pipe queue is effectively stopped, although its actual status remains unchanged. Adding a new destination immediately starts the queue running again.

Syntax

`DElete DEStination=`*destination queuename*

Parameter	Meaning
destination	Name of the destination queue to delete. The syntax for the name must be in one of the following forms. Where shown, square brackets [ ] are required. *local_queue_name* *local_queue_name@local_machine_name* *remote_queue_name@ remote_machine_name* *remote_queue_name@[remote_machine_mid]* You must define all remote machine names and IDs in the local system database. See Chapter 2, the section "Installing the base ConvexNQS+ system," for details on how to do this.
queuename	Name of the pipe queue from which you are deleting destinations.

delete groups

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Deletes one or more groups from the list of groups allowed access to a queue.

If you supply more than one group, separate items in the list with commas and surround the list with parentheses. For example, to designate three groups, enter `(`*group1*`,` *group2*`,` *group3*`)`.

You can only use this command to delete groups added with the add groups command.

You must set the queue for no access before deleting groups. See the `set no_access` command for details on how to do this.

Syntax

`DElete Groups =`*group queuename*

Parameter	Meaning
group	Name or GID of the group to delete. Enclose a GID in square brackets [ ].
queuename	Name of the queue to which access is denied.

delete managers

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Deletes one or more managers from the ConvexNQS+ manager access list.

You cannot delete the superuser account (root).

Syntax

`DElete Managers` *username:option* [*username:option* ...]

Parameter	Meaning
username	Name of the user to delete. The syntax of *username* must be in one of the following forms. Where shown, square brackets [] are required. *local_account_name* *[local_account_id]* *[remote_user_id]@ remote_machine_name* *[remote_user_id]@[remote_machine_mid]* You must define all remote machine names and IDs in the local system database. See Chapter 2, the section "Installing the base ConvexNQS+ system," for details on how to do this.
option	Indicates if the user was granted manager or operator privileges. `m` removes manager access; `o` removes operator access.

delete queue

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Deletes a queue.

You cannot delete the queue unless it is empty and disabled. See the `disable queue` command in this chapter for details on how to do this.

Syntax

`DElete Queue` *queuename*

Parameter	Meaning
queuename	Name of the queue to delete.

delete request

Authorization

ConvexNQS+ manager or operator privileges are required to delete a request you do not own.

Description

Deletes one or more requests from a machine.

You can delete any running or non-running request. If a request is running, ConvexNQS+ sends a SIGKILL signal to all processes in the request. ConvexNQS+ removes and discards deleted requests.

Syntax

`DElete Request` *request_id* [*request_id* ...]

Parameter	Meaning
request_id	Number assigned to the request when it is submitted to ConvexNQS+. Use the `qstat` command to find the *request_id*. Refer to the qstat(1) man page for details on using the `qstat` command.

delete sender

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Deletes one or more pipe queues from a demand queue sender set. A demand queue is a batch queue created with the demand attribute.

If you supply more than one pipe queue, separate items in the list with commas and surround the list with parentheses. For example, to designate three destinations, enter (*sendq1*, *sendq2*, *sendq3*).

ConvexNQS+ successfully completes any request being transferred from the deleted sender before deleting the pipe queue from the sender set.

If you delete all senders for a demand queue, the demand queue is effectively stopped, although its actual status remains unchanged. Adding a new sender immediately starts the queue running again.

Syntax

`DElete Sender` = *sender queuename*

Parameter	Meaning
sender	Name of the pipe queue to delete. The syntax of the pipe queue name must be in one of the following forms. Where shown, square brackets [ ] are required. *local_queue_name* *local_queue_name@local_machine_name* *remote_queue_name@remote_machine_name* *remote_queue_name@[remote_machine_mid]*

You must define all remote machine names and machine IDs in the local system database. See Chapter 2, the section "Installing the base ConvexNQS+ system," for details on how to do this.

queuename	Name of the demand queue from which you are deleting senders.

delete users

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Deletes one or more users from the list of users allowed access to a queue.

If you supply more than one user, separate items in the list with commas and surround the list with parentheses. For example, to designate three users, enter `(`*user1*`,` *user2*`,` *user3*`)`.

You can only use this command to delete users added with the add users command.

You must set the queue for no access before deleting users. See the `set no_access` command for details on how to do this.

Syntax

`DElete Users =`*user queuename*

Parameter	Meaning
user	Name or UID of the user to delete. Enclose a UID in square brackets [ ].
queuename	Name of the queue to which access is denied.

disable queue

Authorization

ConvexNQS+ manager or operator privileges are required to use this command.

Description

Prevents a queue from accepting requests for processing.

Syntax

`DIsable Queue` *queuename*

Parameter	Meaning
queuename	Name of the queue to disable.

enable queue

Authorization

ConvexNQS+ manager or operator privileges are required to use this command.

Description

Lets a queue accept requests for processing.

If the queue is already enabled, ConvexNQS+ ignores the command.

Syntax

`ENable Queue` *queuename*

Parameter	Meaning
queuename	Name of the queue to enable.

exit

Description

Exits from the ConvexNQS+ `qmgr` utility program.

You can also end the `qmgr` utility by sending an end-of-file character. The end-of-file character is typically **CTRL-d**.

Syntax

```
EXit
```

help

Description

Invokes the `qmgr` help facility and displays information about a `qmgr` command or topic.

Syntax

`HElp` [*command*]

Parameter	Meaning
command	Command for which you need information. If you do not supply a command, `qmgr` displays information describing available commands.

hold request

Authorization

ConvexNQS+ manager or operator privileges are required to hold a request you do not own.

Description

Places one or more queued requests on hold, preventing them from running.

A request remains on hold in the queue until you release it with the release request command.

You cannot place a running request on hold.

Syntax

`HOld Request` *request_id* [*request_id* ...]

Parameter	Meaning
request_id	Number assigned to the request when it is submitted to ConvexNQS+. Use the `qstat` command to find the *request_id*. Refer to the qstat(1) man page for details on using the `qstat` command.

modify request

Authorization

ConvexNQS+ manager or operator privileges are required to raise a request's priority. Users can lower the priority of jobs they own.

Description

Changes the intraqueue priority of one or more queued requests so that they run in a different order.

You cannot change the priority of a running request.

Syntax

`MODify Request priority=`*value request_id* [*request_id ...*]

Parameter	Meaning
value	New priority. This can be a number between 0 and 63; 0 is the lowest priority and 63 the highest.
request_id	Number assigned to the request when it is submitted to ConvexNQS+. Use the `qstat` command to find the *request_id*. Refer to the qstat(1) man page for details on using the `qstat` command.

move my_request

Authorization

ConvexNQS+ manager or operator privileges are required to move a request you do not own.

Description

Moves one or more of your queued requests to another queue.

ConvexNQS+ does not move the request if any queue limits, attributes, or access restrictions are violated.

To move a request that is running, first suspend the request, move it, then resume the suspended request.

Syntax

`MOVe My_request` *request_id* [*request_id ...*] *queuename*

Parameter	Meaning
request_id	Number assigned to the request when it is submitted to ConvexNQS+. Use the `qstat` command to find the *request_id*. Refer to the qstat(1) man page for details on using the `qstat` command.
queuename	Name of the destination queue.

move queue

Authorization

ConvexNQS+ manager or operator privileges are required to use this command.

Description

Moves all non-running requests in a queue to another queue.

ConvexNQS+ moves the requests regardless of any queue limit, access restrictions, or attribute violations.

You cannot move running requests.

Syntax

`MOVe Queue` *queuename dest_queue*

Parameter	Meaning
queuename	Name of the current queue.
dest_queue	Name of the destination queue.

move request

Authorization

ConvexNQS+ manager or operator privileges are required to use this command.

Description

Moves one or more non-running requests to another queue.

ConvexNQS+ does not check for queue limit violations, access restrictions, or attribute violations at the destination queue before moving the requests.

To move a request that is running, first suspend the request, move it, then resume the suspended request.

Syntax

`MOVe Request` *request_id* [*request_id ...*] *queuename*

Parameter	Meaning
request_id	Number assigned to the request when it is submitted to ConvexNQS+. Use the `qstat` command to find the *request_id*. Refer to the qstat(1) man page for details on using the `qstat` command.
queuename	Name of the destination queue.

purge queue

Authorization

ConvexNQS+ manager or operator privileges are required to use this command.

Description

Removes all non-running requests from a queue.

ConvexNQS+ completes all running requests. The purged requests are irretrievable.

Syntax

`Purge Queue` *queuename*

Parameter	Meaning
queuename	Name of the queue to purge.

release request

Authorization

ConvexNQS+ manager or operator privileges are required to release a request you do not own. If a ConvexNQS+ manager or operator puts a request on hold, only a ConvexNQS+ operator or manager can release it.

Description

Releases the hold on one or more queue requests, making them eligible to run.

A request must be in a holding state in order to be released.

Syntax

`RELease Request` *request_id* [*request_id* ...]

Parameter	Meaning
request_id	Number assigned to the request when it is submitted to ConvexNQS+. Use the `qstat` command to find the *request_id*. Refer to the qstat(1) man page for details on using the `qstat` command.

run request

Authorization

ConvexNQS+ manager or operator privileges are required to use this command.

Description

Forces one or more requests to begin running immediately.

If running the request exceeds the run limit of the queue, ConvexNQS+ increases the queue run limit until the request finishes running.

Syntax

`RUn Request` *request_id* [*request_id* ...]

Parameter	Meaning
request_id	Number assigned to the request when it is submitted to ConvexNQS+. Use the `qstat` command to find the *request_id*. Refer to the qstat(1) man page for details on using the `qstat` command.

set acc_logfile

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Defines a log file that collects ConvexNQS+ batch accounting information for a ConvexNQS+ machine.

Syntax

`SEt ACC_logfile` *logfile_name*

Parameter	Meaning
logfile_name	Name of the log file.

set accounting

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Enables ConvexNQS+ batch accounting for a batch queue.

Syntax

`SEt ACCOunting` = {ON | OFF} *queuename*

Parameter	Meaning
queuename	Name of the batch queue to set.

set debug

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Defines the level of debug output logged when a ConvexNQS+ utility encounters an error.

Syntax

`SEt DEBug` *level*

Parameter	Meaning
level	The amount of debug information logged: `0` Displays no debugging information. `1` Displays minimum debugging information. `2` Displays maximum debugging information.

set default batch_request priority

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Defines the default intraqueue priority for any batch request submitted without a priority assignment to a queue.

This priority determines the relative ordering of requests within the queue.

Syntax

`SEt DEFault Batch_request Priority` *priority*

Parameter	Meaning
priority	Default priority. This can be an integer ranging between 0 and 63; 0 is the lowest priority and 63 the highest.

set default batch_request queue

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Defines the default queue for any batch request submitted without a queue assignment to a ConvexNQS+ machine.

Syntax

`SEt DEFault Batch_request Queue` *queuename*

Parameter	Meaning
queuename	Name of the default queue.

set default destination_retry time

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Sets the default amount of time that can elapse while ConvexNQS+ tries to resubmit a request to a destination queue through a pipe queue.

If this time elapses, the request fails.

Syntax

`SEt DEFault DEStination_retry Time` *retry-time*

Parameter	Meaning
retry-time	Amount of time in hours.

set default destination_retry wait

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Defines the default time to wait before ConvexNQS+ resubmits a request to a destination queue through a pipe queue.

If a pipe queue destination fails to accept a request because of a network failure or remote server failure, ConvexNQS+ disables the destination for the number of minutes set in *wait-time*. At the end of this time, ConvexNQS+ enables the destination and resubmits the request.

Use the `set default destination_retry time` command to prevent an infinite number of retries.

Syntax

`SEt DEFault DEStination_retry Wait` *wait-time*

Parameter	Meaning
wait-time	Amount of time in minutes.

set description

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Defines the description associated with a queue.

Syntax

`SEt DESCription =`*(description) queuename*

Parameter	Meaning
description	A character string that describes the queue. This string can be up to 79 characters long. Enter `NULL` to delete a description.
queuename	Name of the queue to set.

set destination

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Creates a new destination set for a pipe queue.

If you supply more than one destination, separate items in the list with commas and surround the list with parentheses. For example, to designate three destinations, enter `(`*destq1*`,` *destq2*`,` *destq3*`)`.

ConvexNQS+ removes all previously-defined destinations.

Syntax

`SEt Destination =`*destination queuename*

Parameter	Meaning
`destination`	Name of the destination queue to add. The syntax of the destination queue name must be in one of the following forms. Where shown, square brackets [ ] are required. *local_queue_name* *local_queue_name@local_machine_name* *remote_queue_name@remote_machine_name* *remote_queue_name@[remote_machine_mid]* You must define all remote machine names and IDs in the local system database. See Chapter 2, the section "Installing the base ConvexNQS+ system," for details on how to do this.
queuename	Name of the pipe queue for which you are changing destinations.

set global per_user run_limit

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Controls the number of requests a single user can run in all ConvexNQS+ queues at one time.

Syntax

`SEt Global Per_user Run_limit =`*run-limit*

Parameter	Meaning
run-limit	Total number of requests.

set exec_shell_login

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Sets the status of the per-queue shell execution resource.

Syntax

`SEt_exec_shell_login` = *Answer queue*

Answer can be one of the following:

`Yes`
: Requests submitted to this queue will be executed under a login shell, unless `qsub -nl` is used to submit the request.

`No`
: Requests cannot be executed under a login shell.

`Available`
: Requests will be executed under a login shell only if specifically asked for using `qsub -l`.

queue Queue name

set import_dir

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Establishes whether a batch queue imports the current working directory for a request (mounts the directory on the machine processing the request) before running the request.

ConvexNQS+ imports directories with NFS by making temporary mount points in /tmp of the originating machine. Be aware of local automatic clean-up facilities of /tmp that could change these NFS mount points.

To use the import option, the system manager must first edit the /etc/exports file to add entries for all file systems eligible for remote mounting. Refer to the exports(5) and exportfs(8) man pages for details on how to do this.

Syntax

`SEt Import_dir` = *import-option queuename*

Parameter	Meaning

import-option	Can be one of the following:
	`Yes`
	Queue automatically imports the current working directory for requests submitted to the queue. A user can override this setting for individual requests using the -ni option of `qsub`.
	`Available`
	Lets the user specify importation of the current working directory for requests submitted to the queue using the -i option of `qsub`.
	`No`
	Queue does not import the current working directory. Furthermore, the queue rejects requests that require imported directories.

queuename	Name of the batch queue to set.

set mail

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Defines the user account name appearing as sender for ConvexNQS+ mail. The default account name is the superuser account.

This mail notifies users (when appropriate) of events concerning their ConvexNQS+ requests.

Syntax

`SEt MAIl` *accountname*

Parameter	Meaning
accountname	Name of the user account. The name can consist of any printable nonblank character except for the at sign (@), a comma (,), an equal sign (=), and a left or right parenthesis ( ).

set managers

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Defines a list of authorized `qmgr` managers and operators.

ConvexNQS+ removes all previously-defined authorizations.

ConvexNQS+ managers have full privileges for all `qmgr` commands; ConvexNQS+ operators have privileges for a subset of `qmgr` commands.

Syntax

`SEt MANagers` *username:option* [*username:option* ...]

Parameter	Meaning
username	Name of the user to receive access. The syntax of *username* must be in one of the following forms. Where shown, square brackets [ ] are required. *local_account_name* *[local_account_id]* *[remote_user_id]@ remote_machine_name* *[remote_user_id]@[remote_machine_mid]* You must define all remote machine names and IDs in the local system database. See Chapter 2, the section "Installing the base ConvexNQS+ system," for details on how to do this.
option	Indicates if the user receives manager or operator privileges. `m` grants manager access; `o` grants operator access.

set maximum request_priority

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Defines the maximum intraqueue priority a queue can accept for any request submitted to the queue.

ConvexNQS+ lowers the priority for any request submitted with a higher priority than the queue priority.

Syntax

`SEt MAXimum Request_priority=`*priority queuename*

Parameter	Meaning
priority	Maximum priority. This can be an integer between 0 and 63; 0 is the lowest priority and 63 the highest.
queuename	Name of the queue to set.

set nice_value_limit

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Defines the minimum nice value for any process in a running request in a queue.

The nice value determines the proportion of CPU time allocated to a process relative to all other processes in the system. The lower the nice value assigned to a process, the higher the proportion of CPU time allocated to that process. A practical range is from -20 to 20.

If a user submits a request with a nice value limit, ConvexNQS+ checks the request limit to ensure it does not exceed the queue limit. If the request limit exceeds the queue limit, ConvexNQS+ rejects the request.

If a user submits a request without a nice value limit, ConvexNQS+ uses the queue limit as the default limit.

ConvexNQS+ assigns limits to a request when the request is queued. If a queue limit is changed after a request is queued, the queued request is not affected; however, if the previously-queued request exceeds the new queue limit, ConvexNQS+ displays a warning message.

Syntax

`SEt NIce_value_limit =`*nice-value queuename*

Parameter	Meaning
nice-value	Minimum nice value. This can be an integer between -64 and 64.
queuename	Name of the queue to set.

set no_access

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Deletes all groups and users from the access list for a queue, rendering it inaccessible.

This command does not affect requests in the queue.

This command does not affect superuser privileges; the superuser always has access to all enabled queues, regardless of the contents of the access list.

To restrict access to a queue, use the `set no_access` command to remove all access, then use the add users and add groups commands to specify the users and groups permitted access.

Syntax

`SEt NO_Access` *queuename*

Parameter	Meaning
queuename	Name of the queue to restrict access.

set no_default batch_request queue

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Specifies there is no default batch queue for the ConvexNQS+ network, which means all jobs submitted to ConvexNQS+ must have a queue specification.

Syntax

```
SEt NO_Default Batch_request Queue
```

set per_process permfile_limit

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Defines the maximum permanent file size for any process in a running request in a queue.

If a user submits a request with a permanent file size limit, ConvexNQS+ checks the request limit to ensure it does not exceed the queue limit. If the request limit exceeds the queue limit, ConvexNQS+ rejects the request.

If a user submits a request without a permanent file size limit, ConvexNQS+ uses the queue limit as the default limit.

ConvexNQS+ assigns limits to a request when the request is queued. If a queue limit is changed after a request is queued, the queued request is not affected; however, if the previously-queued request exceeds the new queue limit, ConvexNQS+ displays a warning message.

If any process tries to create a permanent file larger than the limit set, ConvexNQS+ sends a SIGXFSZ signal to the offending process. If the process does not catch the signal or ignores the signal, the process exits.

Syntax

`SEt PER_Process Permfile_limit =`*(limit) queuename*

Parameter	Meaning
limit	Maximum size a permanent file can be for any process in the running request. This can be any integer representing less than 100,000,000 bytes with an optional fractional part. The syntax for *limit* is *integer* [*.fraction*] [*units*] where *integer* and *fraction* are strings of up to eight decimal digits and *units* is one of the following:

bbytes mb megabytes
wwords mw megawords
kbkilobytes gb gigabytes
kwkilowords gw gigawords

If you omit *units*, bytes is assumed.

You can also specify `unlimited` for *limit*.

queuename Name of the queue to set.

Notes

You must set the maximum permanent file size for all batch queues. Recommended setting: (4194303 b)

set per_user run_limit

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Controls the number of requests a user can run in a queue at any one time.

ConvexNQS+ applies per-user run limits after applying per-queue run limits.

Syntax

`SEt Per_user Run_limit=`*run-limit queuename*

Parameter	Meaning
run-limit	Maximum number of requests. A *run-limit* of 0 disables enforcement of this limit.
queuename	Name of the queue to set.

set pipe_client

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Defines the pipe client for a pipe queue.

Syntax

`SEt PIpe_client=`*(client) queuename*

Parameter	Meaning
client	Name of the pipe client. You must define the full path name of the pipe client, followed by any arguments required by the program.

ConvexNQS+ supplies three pipe clients:

- /usr/lib/nqs/pipeclient for standard pipe clients
- /usr/lib/nqs/pipedemand for demand queue pipe clients
- /usr/lib/nqs/pipeldav for load-balancing pipe clients

Refer to the pipeclient(8) man page for more information.

queuename	Name of the pipe queue to set.

set priority

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Defines the interqueue priority for a queue.

This priority affects queue selection for running the next job.

Syntax

`SEt PRiority =`*priority queuename*

Parameter	Meaning
priority	Interqueue priority. This can be any number between 0 and 63; 0 is the lowest priority and 63 the highest.
queuename	Name of the queue to set.

set run_limit

Authorization

ConvexNQS+ manager or operator privileges are required to use this command.

Description

Controls the maximum number of requests that can run in a queue at any one time.

ConvexNQS+ applies per-queue run limits before applying per-user run limits.

Syntax

`SEt Run_limit =`*run-limit queuename*

Parameter	Meaning
run-limit	Maximum number of requests. If you omit *run-limit*, the value defaults to one.
queuename	Name of the queue to set.

set sender

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Defines a list of pipe queues that can send requests to a demand queue (batch queue created with the demand attribute).

If you supply more than one pipe queue, separate items in the list with commas and surround the list with parentheses. For example, to designate three pipe queues, enter *(sendq1,sendq2,sendq3)*.

Syntax

`SEt SENder` = *sender queuename*

Parameter	Meaning
sender	Name of the pipe queue. The syntax of the pipe queue name must be in one of the following forms. Where shown, square brackets [ ] are required. *local_queue_name* *local_queue_name@local_machine_name* *remote_queue_name@remote_machine_name* *remote_queue_name@[remote_machine_mid]* You must define all remote machine names and machine IDs in the local system database. See Chapter 2, the section "Installing the base ConvexNQS+ system," for details on how to do this.
queuename	Name of the demand queue for which you are defining senders.

Restriction

When the demand batch queue notifies senders, it accepts jobs on a first-come, first-served basis, which is generally the first sender in the list.

set shell_strategy fixed

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Defines the shell used to interpret batch request script file commands submitted without a specified interpreter to ConvexNQS+.

Syntax

`SEt SHell_strategy FIxed =`*(shell)*

Parameter	Meaning
shell	This can be `csh`, `ksh`, or `sh`. Supply the absolute path names. The shell must exist and be executable or this command fails.

set shell_strategy free

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Indicates the user's login shell (as defined in the /etc/passwd file) as the initial shell used to interpret batch request script file commands submitted without a specified interpreter to ConvexNQS+.

ConvexNQS+ supplies the name of the script file to the login shell as standard input. The user's login shell reads the first line of the script file. If the first line specifies a shell, ConvexNQS+ uses that shell to interpret script file commands. Otherwise, ConvexNQS+ uses sh. In other words, the request is interpreted as though it were run interactively.

Syntax

```
SEt SHell_strategy FRee
```

set shell_strategy login

Authorization

ConvexNQS+ manager or operator privileges are required to use this command.

Decription

Indicates the user's login shell (as defined in the /etc/passwd file) as the shell used to interpret batch request script file commands submitted without a specified interpreter to ConvexNQS+.

Syntax

```
SEt SHell_strategy Login
```

set unrestricted_access

Authorization

ConvexNQS+ manager privileges are required to use this command.

Description

Adds all groups and all users to the access list for a queue, rendering it accessible to any user or group.

Syntax

`SEt Unrestricted_access` *queuename*

Parameter	Meaning
queuename	Name of the queue to access.

show

Description

Provides status information about ConvexNQS+ on the local machine.

Syntax

SHOW *option*

Parameter	Meaning
option	Can be one of the following:
ALL	Displays information about ConvexNQS+ queues, authorized managers, operating parameters and supported resource limits.
LIMITS_SUPPORTED	Displays supported resource limits.
LONG QUEUE	Displays ConvexNQS+ queue status in an expanded format.
MANAGERS	Displays the list of authorized ConvexNQS+ managers and operators.
PARAMETERS	Displays ConvexNQS+ operating parameters.
QUEUE	Displays ConvexNQS+ queue status in a short format.

The following pages describe each option in detail.

show all

Description

Displays information about ConvexNQS+ queues, authorized managers, operating parameters and supported resource limits.

Syntax

```
SHOw All
```

Output

Figure 22 shows sample output from this command:

Figure 22 `show all` sample output

```
Mgr: show all
Queues:

Queue for short jobs.
short@mach1; type=BATCH;  [ENABLED, INACTIVE];  pri=48
aliases: s, short_queue, S, SHORT
subcomplex: nqs
 0 exit; 0 run; 0 stage; 0 queued; 0 wait; 0 hold; 0 arrive;

Queue for long jobs.
long@mach1;  type=BATCH;  [ENABLED, INACTIVE];  pri=32
aliases: l, long_queue, L, LONG
subcomplex: nqs
 0 exit; 0 run; 0 stage; 0 queued; 0 wait; 0 hold; 0 arrive;

Managers:
 root:m
 batchop:m

ConvexNQS+ operating parameters:
 Debug level =  0
 Default batch_request priority = 31
 Default batch_request queue = long
 Default destination_retry time = 72 hours
 Default destination_retry wait = 5 minutes
 Accounting log file = /dev/null
 Activity ID mask = None
 Mail account = root
 Next available sequence number = 26
 Batch request shell choice strategy = FREE

Limits supported:
 Per-process permanent file size limit (-lf)
 Nice value (-ln)
```

show limits_supported

Description

Displays the resource limits the operating system on the local machine can enforce. If you submit a request with a limit that is not enforced, ConvexNQS+ ignores the limit.

Syntax

```
SHOw LImits_supported
```

Output

Figure 23 shows sample output from this command.

Figure 23 `show limits_supported` sample output

```
Mgr: show limits_supported
Per-process permanent file size limit (-lf)
Nice value (-ln)
```

show long queue

Description

Displays ConvexNQS+ queue status in an expanded format.

Syntax

`SHOw LOng Queue` [*queuename*] [*username*]

queuename — Name of the queue to display. If you omit *queuename*, ConvexNQS+ displays status information for all ConvexNQS+ queues.

username — Name of the user requests to display. If you omit *username*, ConvexNQS+ displays the status of each request in the queue.

Output

Figure 24 shows sample output from this command:

Figure 24 `show long queue` sample output

```
Mgr: show long queue v
Queue for very long jobs.
verylong@mach1;  type=BATCH;  [ENABLED, INACTIVE];  pri=16 aliases:
v, verylong_queue, V, VERYLONG
 0 exit; 0 run; 0 stage; 0 queued; 0 wait; 0 hold; 0 arrive;
 Run_limit = 1;
 Accounting: Off
 Activity ID offset: 0
 Add: maximum request priority: 63
 Cumulative system space time = 1703.580000 seconds
 Cumulative user space time = 2020.910000 seconds
 Unrestricted access
 Import directory: Yes
 Checkpoint: Not available
 Shell execution as login: No
 Per-process core file size limit = UNLIMITED
 Per-process data size limit = UNLIMITED
 Per-process permanent file size limit = UNLIMITED
 Per-process execution nice value = 4
```

show managers

Description

Displays the list of authorized ConvexNQS+ managers and operators.

Syntax

```
SHOw Managers
```

Output

Figure 25 shows sample output from this command:

Figure 25 `show managers` sample output

```
Mgr: show managers
root:m
batchop:o
leader:m
```

show parameters

Description

Displays the current values for the general ConvexNQS+ operating parameters.

Syntax

```
SHOw Parameters
```

Output

Figure 26 shows sample output from this command.

Figure 26 `show parameters` sample output

```
Mgr: show parameters
Debug level = 2
Default batch_request priority = 31
Default batch_request queue = long
Default destination_retry time = 72 hours
Default destination_retry wait = 5 minutes
Global per-user runlimit = NONE
Accounting log file = /dev/null
Activity ID mask = None
Mail account = root
Next available sequence number = 201
Batch request shell choice strategy = FREE
```

show queue

Description

Displays ConvexNQS+ queue status in a short format.

Syntax

`SHOw Queue` [*queuename*] [*username*]

Parameter	Meaning
queuename	Name of the queue to display. If you omit *queuename*, ConvexNQS+ displays status information for all ConvexNQS+ queues.
username	Name of the user requests to display. If you omit *username*, ConvexNQS+ displays the status of each request in the queue.

Output

Figure 27 shows sample output from this command:

Figure 27 `show queue` sample output

```
Mgr: show queue v
Queue for very long jobs.
verylong@mach1; type=BATCH; [ENABLED, INACTIVE]; pri=16
aliases: v, verylong_queue, V, VERYLONG
 0 exit; 0 run; 0 stage; 0 queued; 0 wait; 0 hold; 0 arrive;
```

shutdown

Authotization

ConvexNQS+ manager or operator privileges are required to use this command.

Description

Stops requests where appropriate, and then shuts down ConvexNQS+ on the local machine.

ConvexNQS+ sends a SIGTERM signal to each process of each running request. After the time indicated in *seconds* has passed, ConvexNQS+ also sends a SIGKILL signal to those processes that have not changed process groups.

All requests terminated as a result of the `shutdown` command are requeued for later execution.

Syntax

`SHUtdown` [*seconds*] [`force`]

Parameter	Meaning
seconds	Amount of real time that ConvexNQS+ waits after sending a SIGTERM signal to send a SIGKILL signal If you omit *seconds*, ConvexNQS+ assumes a default value of 60 seconds.
`force`	Forces ConvexNQS+ to shut down.

start nqs+

Authorization

ConvexNQS+ manager or operator privileges are required to use this command.

Description

Starts ConvexNQS+ on the local machine.

You cannot use this command to start ConvexNQS+ the first time you install ConvexNQS+.

Syntax

```
STArt Nqs+
```

start queue

Authorization

ConvexNQS+ manager or operator privileges are required to use this command.

Description

Allows a queue to process requests.

If the queue is already started, ConvexNQS+ sends a transaction completion message; no jobs are aborted.

You can automatically start queues using the cron utility. Refer to the crontab(5) man page for detailed information on using the cron utility and setting up the crontab file.

Syntax

`STArt Queue` *queuename*

Parameter	Meaning
queuename	Name of the queue to start.

stop queue

Authorization

ConvexNQS+ manager or operator privileges are required to use this command.

Description

Prevents a queue from processing requests.

ConvexNQS+ completes currently running requests but freezes all non-running requests. These frozen requests remain in the queue and run when you restart the queue using the `start queue` command.

You may still submit new requests; however, they are also frozen.

You can automatically stop queues using the cron utility. Refer to the crontab(5) man page for detailed information on using the cron utility and setting up the crontab file.

Syntax

`STOp Queue` *queuename*

Parameter	Meaning
queuename	Name of the queue to stop.

qmapmgr commands 7

qmapmgr reference pages

This chapter contains a description of each `qmapmgr` command. Use these commands to build and maintain the ConvexNQS+ database on the local machine.

You can enter commands in any combination of uppercase and lowercase characters.

The first two characters of each word in each command are unique, and you need to enter only those characters for ConvexNQS+ to recognize the command. These accepted abbreviations are indicated in the "Synopsis" section of each command description as uppercase characters.

ConvexNQS+ man pages

Online man pages available for ConvexNQS+ are listed on page 65.

add mid

Authorization

Superuser privileges are required to use this command.

Description

Adds a new machine identification to the ConvexNQS+ database.

Syntax

`Add Mid` *mid principal-name*

Parameter	Meaning
mid	Unique number identifying the machine to add.
principle-name	Name of the corresponding ConvexNQS+ machine. The name must be unique and correspond to an entry in the /etc/hosts file; it should not be an alias.

add name

Authorization

Superuser privileges are required to use this command.

Description

Adds an alternate name for a machine to the ConvexNQS+ database.

You can use the machine name or any alias assigned to the machine to reference a machine.

Syntax

`Add Name` *alias mid*

Parameter	Meaning
alias	Unique, alternate name. This alias is known only to the local ConvexNQS+ machine and is not recognized across machines.
mid	MID of the machine receiving the alias. The MID must already exist in the ConvexNQS+ database.

change name

Authorization

Superuser privileges are required to use this command.

Description

Changes the name of a machine in the ConvexNQS+ database.

Syntax

`CHange Name` *mid new-name*

Parameter	Meaning
mid	MID of the machine to change. The MID must already exist in the ConvexNQS+ database.
new-name	New machine name. The new name must be unique and correspond to an entry in the /etc/hosts file.

Authorization

Superuser privileges are required to use this command.

Description

Creates a new ConvexNQS+ database.

Syntax

```
CReate
```

delete mid

Authorization

Superuser privileges are required to use this command.

Description

Deletes a machine from the ConvexNQS+ database.

Syntax

`Delete Mid` *mid*

Parameter	Meaning
mid	MID of the machine to delete.

delete name

Authorization

Superuser privileges are required to use this command.

Description

Deletes an alternate name used to reference a machine.

Syntax

`Delete Name` *alias*

Parameter	Meaning
alias	Alternate name to delete.

exit

Description

Exits from the ConvexNQS+ qmapmgr utility.

You can also exit qmapmgr by entering the `quit` command or by pressing **CTRL-d**.

Syntax

```
Exit
```

get mid

Description

Displays the MID for a machine.

Syntax

`Get Mid` *machine-name*

Parameter	Meaning
machine-name	Name or alias of the machine to display.

get name

Description

Displays the machine name matching an MID.

Syntax

`Get Name` *mid*

Parameter	Meaning
mid	MID of the machine to display.

Description

Invokes the qmapmgr help facility and displays all available qmapmgr commands.

Syntax

```
Help
```

Output

Figure 28 shows sample output from this command:

Figure 28 qmapmgr help sample output

```
Mapmgr: help
Commands:
Add Name <name> <to_mid>
Add Mid <mid> <principal-name>
CHange Name <mid> <new-name>
CReate
Delete Name <name>
Delete Mid <mid>
Exit
Get Mid <name>
Get Name <mid>
Help
Quit
SHow
^D (to exit qmapmgr)
```

quit

Description

Exits from the ConvexNQS+ qmapmgr utility.

You can also exit qmapmgr by entering the exit command or by pressing **CTRL-d**.

Syntax

```
Quit
```

Description

Displays all entries in the ConvexNQS+ database in MID order.

Each entry includes the MID, machine name, and alias if assigned.

Syntax

```
Show
```

Output

Figure 29 shows sample output from this command:

Figure 29 qmapmgr show sample output

```
Mapmgr: show
Mid 1 = machine2, s
Mid 2 = machine1
Mid 3 = pluto, p
Mid 4 = test
Mid 5 = user1, u
```

ConvexNQS+ run-time directory hierarchy

A

This appendix describes the ConvexNQS+ runtime directory hierarchy. The chart on the following pages provides information on ConvexNQS+ directory/file content and usage.

Do not change access permissions on any of these directories. ConvexNQS+ sets access permissions automatically and uses them to limit access to files. If access permissions are changed erroneously, please call the CONVEX Technical Assistance Center (TAC) for instructions on restoring them.

While system managers and others logged in as root may use `cd` to move to these directories, many of the files are binary and cannot be read. You do not need to access them or remove anything from them, with the exception of the /failed directory, which contains output from failed jobs. Users whose output is sent to the /failed directory receive mail informing them of the failure and location of their output. The system manager may remove files from the /failed directory.

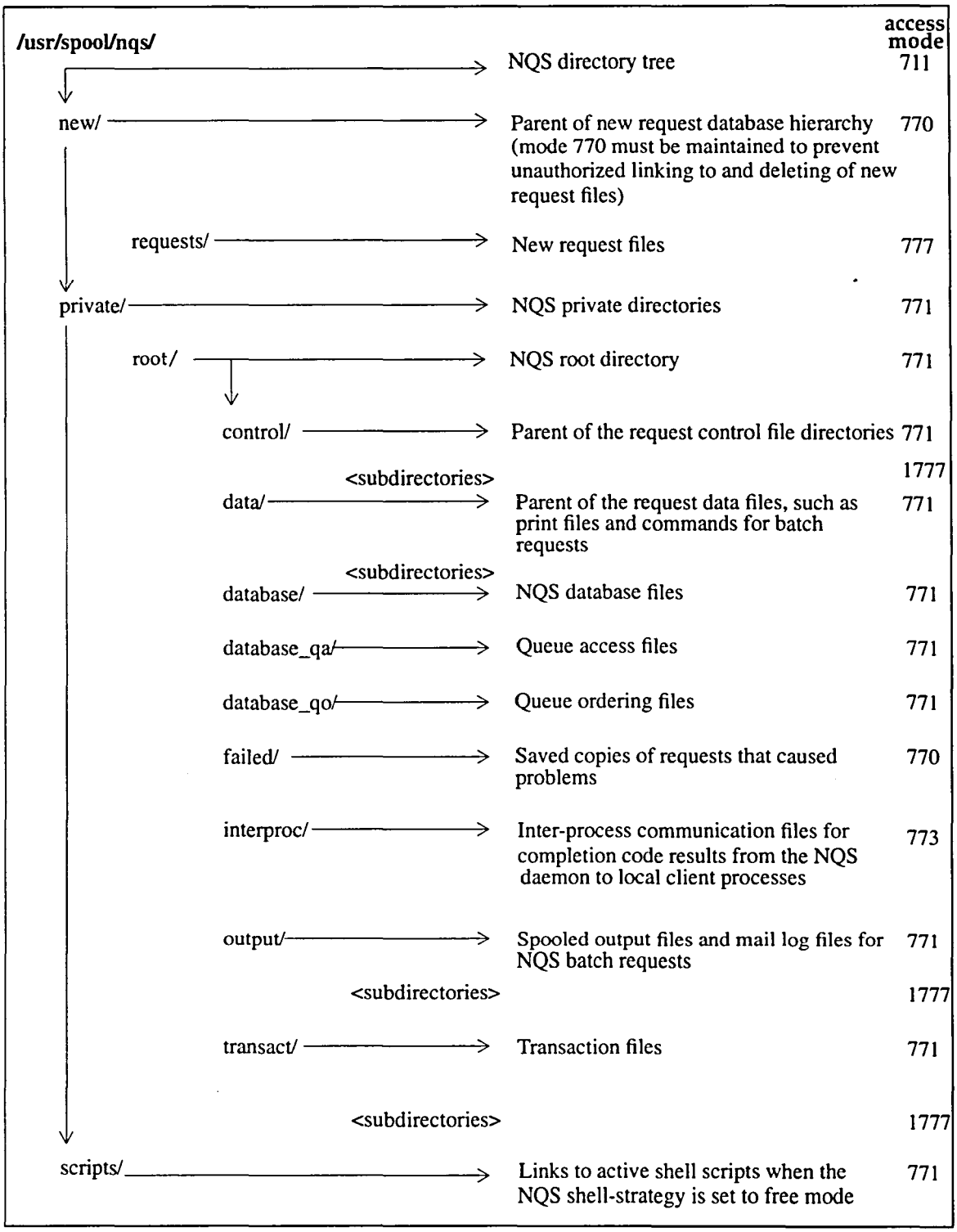
For more information on the contents or access mode of any of these files, please contact the CONVEX Technical Assistance Center (TAC).

Because it holds script files and job output until output is sent to user, the /usr/spool directory can become full. Consider creating a separate partition for /usr/spool/nqs.

The directory named database contains information for the `set` commands. If this or any other ConvexNQS+ database becomes corrupted, contact the CONVEX Technical Assistance Center (TAC).

Look in the directory named `failed` for output from jobs that fail or from jobs whose resource limits prevent correct output placement. You must be logged in as root to remove files from this directory.

Figure 30 The ConvexNQS+ runtime hierarchy



ConvexNQS+ daemons

B

This appendix describes the daemons used by ConvexNQS+.

A daemon is a process that provides a particular service when needed, but is not connected to a terminal or a particular user. ConvexNQS+ uses the following daemons:

logdaemon	Is contacted by nqsdaemon and netdaemon when they need to print an error message. logdaemon sends a message to syslogd (if defined) and notifies the ConvexNQS+ manager if the error is fatal.
netclient	Transfers jobs to remote machines and transfers stderr and stdout to appropriate destinations.
netdaemon	Handles all remote transactions involving queues, including submitting jobs and copying stdout and stderr files.
netserver	Is created by netdaemon to handle operations that require a substantial amount of time, such as queuing a request.
nqsdaemon	Handles all local transactions, including job submission and deletion, job scheduling, and system configuration.
pipeclient	Is created by nqsdaemon to route requests to the first queue in a pipe queue destination set that is able to accept the job.
pipedemand	Is created by nqsdaemon to route requests to the first queue in a pipe queue destination set that can immediately run the request
pipeldav	Is created by nqsdaemon to route requests based on load factor.

shepherd	A child of the nqsdaemon that watches over batch jobs, shepherd is responsible for setting up the job environment and returning output files.

Note

When you use qps to print information about ConvexNQS+ daemon processes, shepherd appears as nqsdaemon under COMMAND. The corresponding information under QUEUE and REQ distinguishes the shepherd process from the true nqsdaemon process.

Index

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

ORDER NUMBER
DSW-860

DOCUMENT NUMBER
770-007430-000