The
Connection Machine
System

NQS for the CM-5

Version 2.0
January 1992

Thinking Machines Corporation
Cambridge, Massachusetts

First printing, January 1992

Thinking Machines Corporation
245 First Street
Cambridge, Massachusetts 02142–1264
(617) 234–1000/876–1111

Contents

About This Manual

Objectives of This Manual

This manual is a guide to using, configuring, and managing the NQS batch system on the Connection Machine system Model CM-5.

Intended Audience

CM-5 users should read Chapter 1 to learn how to use the NQS batch system. We assume that users are familiar with the UNIX operating system and the basics of executing programs on the CM-5.

System administrators should read Chapter 2 for an explanation of how to configure and manage NQS.

Revision Information

This manual is specific to the CM-5. It is a new manual.

Related Documents

- *CM-5 Software Summary,* CMOST Version 7.1
- *CM-5 System Administrator's Guide,* Version 7.1

Notation Conventions

The table below displays the notation conventions observed in this manual.

Convention	Meaning
bold typewriter	UNIX and CM System Software commands, command options, and filenames, when they appear embedded in text. Also C/Paris and C language elements, such as keywords, operators, and function names, when they appear embedded in text.
italics	Argument names and placeholders in function and command formats.
`typewriter`	Code examples and code fragments.
`% bold typewriter` `typewriter`	In interactive examples, user input is shown in **bold typewriter** and system output is shown in `regular typewriter` font.

Customer Support

Thinking Machines Customer Support encourages customers to report errors in Connection Machine operation and to suggest improvements in our products.

When reporting an error, please provide as much information as possible to help us identify and correct the problem. A code example that failed to execute, a session transcript, the record of a backtrace, or other such information can greatly reduce the time it takes Thinking Machines to respond to the report.

If your site has an applications engineer or a local site coordinator, please contact that person directly for support. Otherwise, please contact Thinking Machines' home office customer support staff:

U.S. Mail:	Thinking Machines Corporation Customer Support 245 First Street Cambridge, Massachusetts 02142–1264
Internet Electronic Mail:	customer–support@think.com
uucp Electronic Mail:	ames!think!customer–support
Telephone:	(617) 234–4000 (617) 876–1111

Chapter 1

Using NQS on a CM-5

NQS stands for *Network Queueing System*, which is a standard batch system. In a batch system, you submit one or more programs as a request to a queue, rather than executing them directly. The batch system in turn submits the requests in the queue for execution. Your request is generally executed when it reaches the head of its queue.

Read this chapter to learn how to use NQS on the CM-5. Read the next chapter if you are a system administrator and want to set up NQS on your CM.

To help you learn how to use NQS, we provide a trivial sample program in the first section of this chapter, followed by instructions on how to compile it. You can submit the compiled program via NQS to be executed on the CM.

1.1 A Simple Program

The program shown below is written in CM Fortran.

The program sets up three arrays of five elements each. The elements of array A are assigned the values 1, 2, 3, 4, 5; the elements of array B are each assigned the value 2. The program then squares each of these values, adds each element of A to the corresponding element of B, and puts the results in array C. It then prints the results. (This, of course, is not a typical data parallel program.)

```
PROGRAM SIMPLE

INTEGER A, B, C, N
PARAMETER (N=5)
DIMENSION A(N) B(N), C(N)

DATA A /1,2,3,4,5/
B=2

C=A**2 + B**2

PRINT */ 'Array C contains:'
PRINT *, C

END
```

Type this program in a file as you normally would; call the file **simple.fcm**. (Remember that in Fortran each program statement must begin in column 7.)

To compile the program, issue the following command at your UNIX prompt (which is represented as a percent sign in this manual):

```
% cmf simple.fcm -o simple
```

You now have a CM Fortran program called **simple** that is ready for execution on the CM.

1.2 Overview of NQS

As mentioned above, NQS is a standard batch system. NQS can also be used for batch submissions to computers other than the CM. In this manual, however, we focus only on using NQS to submit requests for execution on the Connection Machine system.

The CM system administrator is in charge of configuring NQS queues to meet the needs of your site. A particular partition manager on the CM may not have any queues, or it may have several. You may have only *batch queues*, which submit requests directly to the CM for execution, or you may in addition have *pipe queues*, which pass requests along to batch queues. Pipe queues are useful be-

cause they can be associated with several different batch queues; if one is unavailable, the pipe queue can try the next, until it finds one that will accept the request. You don't have to worry about finding the available queue yourself.

Here are some of the characteristics of batch queues that a system administrator can configure:

- *Who can use a queue.* The system administrator can restrict access to a queue to specific users or groups.
- *Limits associated with requests in the queue.* For example, there can be limits on the amount of CPU time or stack size that a request can use.
- *When the queue submits its requests for execution.* A batch queue can operate continuously, or it can operate only at specified times—for example, from midnight to six in the morning.

To submit a request for execution via either a batch queue or a pipe queue, you either:

- Issue the **qsub** command, using as an argument the name of a script file that contains the name of the program or programs to be run; or
- Submit the program or programs to **qsub** from the standard input

See Section 1.3, below.

Processes in a batch request run under timesharing, just like any other processes.

To obtain information about a queue, or about the status of a request in a queue, issue the **qstat** command. See Section 1.5.

Table 1 lists the NQS user commands described in the rest of this chapter. The appendix contains UNIX man pages for all NQS commands.

Table 1. User commands for the NQS batch system.

Command	Meaning
qdel	Delete or signal one or more batch requests.
qlimit	Display the supported limits on batch queues.
qstat	Display the status of queues and batch requests.
qsub	Submit a batch request.

1.3 Submitting a Batch Request: The qsub Command

Use the **qsub** command to submit a program for execution via either a batch queue or a pipe queue.

NOTE: The **qsub** command has many options associated with it; in this chapter, we discuss only some of the more important. See the man page for **qsub** in the appendix for a complete discussion of all its options. Table 2 summarizes the **qsub** options supported on the CM-5.

Table 2. Options for the qsub command.

Option	Meaning
-a *time*	Do not run the request before the specified time and/or date.
-e *filename*	Direct the standard error output to the specified file.
-eo	Direct the standard error output to the batch request output file.
-ke	Keep the standard error output on the execution machine.
-ko	Keep the standard output on the execution machine.
-lc *size*	Set the per-process corefile size limit.
-ld *size*	Set the per-process data-segment size limits.
-lf *size*	Set the per-process permanent-file size limit.
-ln *value*	Set the per-process nice execution value limit.
-ls *size*	Set the per-process stack-segment size limit.
-lt *time*	Set the per-process CPU time limit.
-lw *size*	Set the per-process working set limit.
-mb	Send mail when the request begins execution.
-me	Send mail when the request ends execution.
-mr	Send mail when the request restarts.
-mt	Send mail when the request is transported between queues.
-mu *username*	Send mail about the request to the specified user.
-nc	Declare that the request is not recoverable.
-nr	Declare that the request is not restartable.
-o *filename*	Send the output of the request to the specified file.
-p *priority*	Set the priority for the request in the batch queue.
-q *queue*	Send the request to the specified batch queue.
-r *name*	Assign the specified request name to the request.
-re	Remotely access the standard error output file.
-ro	Remotely access the standard output file.
-s *shell*	Use the specified shell to interpret the request.
-x	Export all environment variables with the request.
-z	Submit the request silently.

1.3.1 The Basics

To execute the program **simple** via the queue **cmq1**, put the program's name in a *script file*. A script file is simply a UNIX file that contains commands to be executed. For example, you could create a file that contains just the word **simple**; you could call the file **/yourname/simple_script**. You could then submit this file to **cmq1** as follows:

```
% qsub -q cmq1 /yourname/simple_script
```

The **-q** flag specifies the name of the queue to which you are submitting the request.

The system displays a response like this:

```
Request 276.barney.acme.com submitted to queue: cmq1
```

The number 276 is a *sequence number* assigned to this request by NQS, and **barney.acme.com** is the *hostname* of the computer from which the request was submitted; **276.barney.acme.com** is the *request-id* for this request.

When **simple** is finally executed, its output is placed in a file; error messages are placed in another file.

Submitting a batch request has these basic elements:

- Specifying the queue to which the request is being submitted
- Specifying the request to be run
- Specifying options that affect the way the request is to be run

You can embed **qsub** options at the beginning of the script file, along with the name of the executable program and other commands. See Section 1.3.2, below, for an example. NQS looks at options in the script file only if they are not specified on the **qsub** command line; this lets you override a script file option by specifying a different setting for the option on the **qsub** command line.

1.3.2 Specifying the Queue

There are several different methods of specifying the queue to which you want to submit your request. You can find out the names and characteristics of available batch queues or pipe queues by issuing the **qstat** command with the **-b** or **-p** option, respectively; see Section 1.5.2.

You can use the following methods to specify a batch queue:

- Use the **-q** option on the **qsub** command line, as shown in the example above. NQS submits the request to the queue you specify.
- Embed the **-q** option in a script file that you name on the **qsub** command line. All **qsub** options must appear at the beginning of the script file, and must begin with a pound sign (#) followed by an "at" sign (@) and a dollar sign ($). The option must begin immediately after the dollar sign—no white space is allowed. Comments must begin with a pound sign. For example, the following script file sends the program **simple** to queue **cmq1** for execution:

```
#
# Example of a batch script file
#
# @$-q cmq1 # Send request to cmq1 unless
#           # overridden on command line.
#
simple
```

If you named this script file **simple_script**, you could execute the program by issuing the following command:

```
% qsub simple_script
```

- Set the environment variable **QSUB_QUEUE** to the name of the queue to which you want the request submitted. You would typically do this to set up a default queue for all requests, which you could override for a specific request by using the **-q** option. If you use the C shell, you could put the following command in your **.cshrc** file to set the default queue to **cmq2**:

```
setenv QSUB_QUEUE cmq2
```

If you don't use any of these methods for specifying a queue, the request is submitted to the default batch queue for the system, if your system administrator has defined one.

1.3.3 Specifying a Request from a Script File

As we have already shown, you can execute a program by including its name in a script file. You can include UNIX commands in the file as well. Typically, NQS interprets the commands in a script file exactly as if you had typed them at your UNIX prompt. It may, however, use a different shell to interpret the commands, depending on how your system administrator has configured NQS. See Section 1.3.7, below.

1.3.4 Specifying a Request from Standard Input

Instead of using a script file, you can simply enter the request from standard input—that is, directly after the **qsub** command line. Put each command or program name on a separate line, and type the **Ctrl-d** key combination at the end to signal that there is no more input. For example:

```
% qsub -q cmq1
  simple
  cmsetsafety on
  simple
  Ctrl-d
```

If you are executing a shell under Emacs or Gmacs, type **Ctrl-c Ctrl-d**.

1.3.5 The Output from a Request

NQS places the output from a batch request in a file, which is by default placed in the directory from which you issued **qsub**. You can control the name and location of this file. The default filename consists of the first seven characters of the script name, followed by **.o**, followed by the sequence number of the request. Thus, in our example in Section 1.3.1, NQS would put the output in the file **sim-**

ple_.o276 in your current working directory. Messages to standard error go into a file with **.e** in the name instead of **.o** (**simple_.e276** in this example).

If you submit the request from standard input, the default output and error files would begin with **STDIN.o** and **STDIN.e**, followed by the sequence number.

To specify a different output filename, use the **-o** option, followed by a pathname, on the **qsub** command line or in a script file. NQS writes output to the pathname you specify. For example,

```
% qsub -o /requests/simple.out simple
```

causes the standard output of the program **simple** to be written to **/requests/simple.out**.

Similarly, use the **-e** option to specify a different pathname for standard error output.

Another way to change the name of the output file is to use the **-r** option, followed by a request-name of up to 15 characters. This request-name identifies the request when you issue the **qstat** command to check the status of requests; if you don't specify a request-name, NQS uses the name of the script file (or **STDIN**) instead. If you do specify a request-name, NQS substitutes it for the name of the script file (or **STDIN**) in the name of the output file.

Here is sample output for our **simple_script** batch request:

```
Cold boot...
Array C contains:
          5      8      13     20     29
FORTRAN STOP
logout
```

NOTE: In processing the batch request, NQS runs a script as if it were logged in as you; start-up files like **.cshrc** and **.login** are executed. This means that you may see various messages along with the output. In particular, you will probably see the following message:

```
Warning: no access to tty; thus no job control in this
shell...
```

This comes from the shell, warning you that there is no terminal associated with this job. You can ignore this message.

NOTE: While the request is running, NQS considers your home directory to be your current working directory (because it runs the request as a newly logged-in process). Thus, if the process dumps core, the corefile is placed in your home directory, rather than the directory from which you submitted the request.

1.3.6 Setting Limits on a Request

As shown in Table 2, the **qsub** command has many options you can specify to set limits on the amount of resources a batch request can use. The batch queue has its own set of limits. You can find them out by issuing the **qstat** command with the **-l** option; see Section 1.5.2. You may want to set lower limits to obtain more favorable scheduling for your request, or to avoid running up accounting charges if, for example, your program goes into an infinite loop. If the limit you set *exceeds* the limit for a queue, the queue will not accept the request.

For example, use the **-lt** option to set a limit on the amount of partition-manager CPU time an individual program within a batch request can use. The following command sets a limit of 120 seconds of CPU time for the program **simple**:

```
% qsub -lt 120 simple
```

CMOST does not support all of the limit options that **qsub** lets you specify. If you specify an unsupported option, NQS ignores it. To find out which options your site supports, issue the **qlimit** command. For example, if your partition manager is named Barney, issue **qlimit** as follows:

```
% qlimit barney
```

The response might look like this:

```
Core file size limit (-lc)
Data segment size limit (-ld)
Per-process permanent file size limit (-lf)
Nice value (-ln)
Stack segment size limit (-ls)
Per-process cpu time limit (-lt)
Working set limit (-lw)

Shell strategy = FREE
```

These are the limits you can set for this partition manager. The "shell strategy" in this response refers to the default way in which NQS chooses a shell to interpret commands in a script file. See Section 1.3.7, below.

NOTE: Other limits may be available if you are sending your request to another computer.

1.3.7 Choosing a Shell

As we mentioned above, your system administrator can specify how NQS is to interpret commands in batch script files. This is called the *shell strategy*; you can find out the default shell strategy via the **`qlimit`** command. The possible shell strategies are:

- *Free.* Your login shell determines the appropriate shell to be used to execute the commands in your script file, and executes that shell. This typically means that NQS uses the shell that would have been used if you had issued the commands in the script file interactively. For example, if your script file begins with the line

  ```
  #! /bin/csh
  ```

  your login shell would execute a C shell for the script file.

- *Login.* NQS uses your login shell to execute the commands in your script file, regardless of the contents of your file.

- *Fixed.* NQS uses a specified shell to execute the commands, regardless of the contents of your script file. Use **qlimit** to find out the name of this shell.

You can override the shell strategy by using the **-s** option with the **qsub** command. For example,

```
% qsub -s /bin/csh simple_script
```

specifies that the C shell is to be used to interpret the commands in the script file **simple_script**.

1.3.8 Setting a Priority for a Batch Request

To set a priority for your batch request in its queue, use the **qsub** option **-p**, followed by an integer from 0 to 63, inclusive; 63 is the highest priority, and 0 is the lowest priority. This priority determines the request's position in the queue. The request is placed in front of all requests with lower priority, and behind all requests with higher or the same priority.

If you don't specify a priority, the request is assigned a default priority, as set by the system administrator. Use the **qstat** command with the **-fb** or **-fp** option to determine the default priority for a queue; see Section 1.5.2.

NOTE: NQS does not necessarily run requests in the order in which they appear in a batch queue. It can take requests out of order to use resources efficiently. Generally, however, requests at the beginning of the queue are run before requests that appear later in the queue.

1.3.9 Receiving Mail about a Batch Request

Use the **qsub** options **-mb** and **-me** to specify that NQS is to send you mail about your batch request. Specify **-mb** to get mail when the request begins execution; specify **-me** to get mail when the request ends execution. Use the **-mu** option with a username to send the mail to another user.

To obtain more information about the status of a batch request, use the **qstat** command with the **-a** option; see Section 1.5.1.

1.4 Deleting a Batch Request: The qdel Command

Issue the **qdel** command to delete a request from a queue. As an argument, specify the request-id that was displayed when you submitted the request. (You can also obtain the request-id by issuing the **qstat** command with the **-a** option; see Section 1.5.1, below.) For example,

```
% qsub -q cmq1 simple_script
Request 276.barney.acme.com submitted to queue:
cmq1
% qdel 276
```

submits a request, then deletes it. (You don't need to specify the hostname if you are issuing the command from the local host.)

This form of the **qdel** command does not delete a request that is actually running. To do this, use the **-k** option. This option sends a SIGKILL signal to the specified request, causing it to exit and be deleted from the queue. If the request contains more than one process, all are signalled.

To send a signal other than SIGKILL to a running request, specify its number instead of **k** (see the discussion of **sigvec** in your UNIX documentation for signal numbers). For example, to send a SIGTERM signal to a running request with request-id 276, issue this command:

```
% qdel -15 276
```

1.5 Obtaining Information: The qstat Command

Use the **qstat** command to obtain information about queues and batch requests. This section describes some of the more important options; for complete information on **qstat**, see its man page in the appendix.

1.5.1 Finding Out about Batch Requests

If you issue **qstat** with no options, the status of all your requests on the local host are displayed. To find out the status of all requests on any queue, issue **qstat** with the **-a** option. To find out the status of the requests of a particular user, use the **-u** option, followed by the user's name. For example:

```
% qstat -u sharon
```

The response might look like this:

```
=============================================================
NQS Version: 2        REQUESTS on diamond.think.com
=============================================================
REQUEST     NAME   OWNER   QUEUE    PRI  NICE   CPU    MEM       STATE
0@diamond.  cm5sub sharon  pn_128   32    20  36000  UNLIMITED  RUNNING
```

This shows that user Sharon has one request, with request-id **0@diamond.think.com** and name **cm5sub**, running on the queue **pn_128**. The other fields are:

- **PRI** — This is the intraqueue priority for the request; see Section 1.3.8.
- **NICE** — This is the nice value for the request. If you don't specify this value via the **-n** option to the **qsub** command, NQS uses the default value for the queue. (See your UNIX documentation for more information on nice values.)
- **CPU** — This is the maximum partition-manager CPU time for the request; once again, you can set this via a **qsub** option (**-lt**), or you can accept the default queue limit.
- **MEM** — This is the maximum memory for the request; for requests running on a CM-5, this will always be UNLIMITED.
- **STATE** — This is the state of the request. A request can be in one of these states:
  - **Arriving** — A request is *arriving* if it is being placed on the queue from a remote host.
  - **Holding** — A request is *holding* if it is currently prevented from entering any other state because a hold has been placed on it.

- **Waiting** — A request is *waiting* if it was submitted with the constraint that it not be run before a certain date or time, and that date or time hasn't arrived yet. You submit a request in this way by using the **qsub** option **-a**.
- **Queued** — When a request is *queued*, it is eligible to run as soon as the resource is available or the queue's restriction window opens.
- **Routing, Departing** — If the queue is a pipe queue, a request can be *routing* or *departing* as it passes through the queue.
- **Staging** — A request is *staging* when its input files are being brought on to the partition manager on which it is to execute.
- **Running** — A request is *running* when it is actually executing.
- **Exiting** — A request is *exiting* when it has completed execution and the required output files are being returned.

1.5.2 Finding Out about Queues

Use **qstat** with the **-b** option to find out the status of NQS batch queues. The response might look like this:

```
=======================================================
NQS Version: 2   BATCH QUEUES on diamond.think.com
=======================================================
QUEUE NAME     STATUS   TOTAL   RUNNING  QUEUED  HELD TRANSITION
-------------------------------------------------------------------
q1             AVAILBL    1       1/1       0      0       0
q2             AVAILBL    2       2/10      0      0       0
```

There are two batch queues, **q1** and **q2**, on **diamond.think.com**. The other fields provide the following status information about the queues:

- **STATUS** — A queue can be in one of these states:
  - **AVAILABL** — The queue is *enabled* (that is, you can submit requests to it) and *started* (that is, submitted requests can run).
  - **STOPPED** — The queue is enabled but stopped (that is, the queue is enabled, but queued requests are blocked from running).

- **DISABLED** — You cannot submit requests to the queue.
- **UNAVAIL** — The queue is disabled and stopped.
- **NQS DOWN** — The NQS daemon for the partition manager is not running.

You can submit requests to a queue only if the queue is enabled and the local NQS daemon is present.

- **TOTAL** — This is the number of requests currently in the queue.
- **RUNNING** — The first value in this field is the number of requests that are running in the queue. The second value is the *run limit* for the queue (that is, the maximum number of requests that can be running in the queue at the same time). Thus, in the example, **q1** has one request running, and its run limit is 1; **q2** has two requests running out of a possible 10.
- **QUEUED** — This is the number of requests in the queue waiting to run.
- **HELD** — This is the number of requests that are ineligible for processing.
- **TRANSITION** — This is the number of requests that are in between queues (for example, between a pipe queue and a batch queue).

Use the **-p** option to find out this information about pipe queues.

Use the **-f** option along with **-b** or **-p** to display information about a queue in a fuller format. For example,

```
% qstat -fb bq0
```

might produce this response about the batch queue **bq0**:

```
=========================================================
NQS version:2  BATCH      QUEUE:  bq0.eno.think.com    status:  AVAILBL
=========================================================
                                      Priority: 32
ENTRIES:
        Total:   0      Running: 0
        Queued:  0      Held:    0      Transiting: 0
COMPLEX MEMBERSHIP:

RESOURCES:
  Per-proc core file size limit= 1 megabytes <DEFAULT>
  Per-process data size limit  = 1 megabytes <DEFAULT>
  Per-proc perm file size limit= 1 megabytes <DEFAULT>
  Per-proc execution nice value= 0 <DEFAULT>
  Per-process stack size limit = 1 megabytes <DEFAULT>
  Per-process CPU time limit   = 36000.0 <DEFAULT>
  Per-process working set limit= 1 megabytes <DEFAULT>
  Restriction Window start time = <NONE>
  Restriction Window stop time = <NONE>
  Restriction Window mode = <NOT SET>
  Restriction Window send terminate signal = <NOT SET>

ACCESS
  Unrestricted access
```

Comments about some of these fields:

- The **COMPLEX MEMBERSHIP** field indicates whether or not this queue belongs to a *queue complex*. Multiple queues can belong to a queue complex that shares a single run limit.
- The **Restriction Window** fields indicate the time during which the queue is available to run requests.
- The **ACCESS** field indicates what access restrictions, if any, have been placed on the queue. The system administrator can restrict access to specified users and groups.

Chapter 2

Configuring and Managing NQS

This chapter describes how to configure and manage the NQS batch system.

You can configure the batch system to meet the needs of your site. For example, you can:

- Configure a batch queue to run at certain times or on certain days, so that low-priority jobs can run late at night or on weekends. (See Section 2.5.3.)
- Restrict a queue to specified users or groups of users, thereby limiting access to the CM to high-priority jobs. (See Section 2.7.)
- Set limits on the kinds of processes that can run in the queue. (See Section 2.8.)
- Assign batch queues to a *queue complex*, which has an overall limit on the number of requests that can run at the same time. (See Section 2.9.)
- Create *pipe queues*, which feed requests to batch queues. This allows users to submit requests to a single pipe queue, which can automatically find an available CM–5 partition. (See Section 2.10.)

NQS has more capabilities than those discussed in this chapter, which focuses on using a CM via the batch system. For complete information on NQS, see the man pages for NQS commands, especially `qmgr`, in the appendix.

2.1 Starting Up NQS

The NQS software is typically installed on a partition manager as part of the installation of CMOST. To start up NQS, run the script **startnqs**. This script starts up the NQS daemon, **nqsdaemon**, and directs its output to **/usr/spool/nqs/log.daemon**.

The installation puts NQS user commands in **/usr/bin**; it puts daemons (such as **nqsdaemon**) in **/usr/etc**.

If you want to set up pipe queues on computers that are not partition managers, you must install NQS on those machines. After installing NQS, you must map the machine into the NQS network, as described in Section 2.12.6.

NQS must be installed before you can use the **qmgr** utility, which is described in the following section.

2.2 Configuring NQS — The qmgr Utility

Use the **qmgr** utility to set up NQS for your system. When you issue **qmgr**, you are put into an environment from which you can issue commands to configure all aspects of NQS. Subsequent sections of this chapter describe some of these commands; for complete information, see the man page for **qmgr** or issue the **help** command in the **qmgr** environment.

When using **qmgr** for the first time, you must log in as superuser before issuing the command. Then, at the **Mgr:** prompt, issue a **qmgr** command with the following format:

```
Mgr: add manager your_login_name:m
```

This makes you an NQS manager; you need not subsequently log in as the superuser to issue **qmgr** commands.

You can abbreviate **qmgr** commands, as long as the abbreviation for each keyword in the command is unique. In the sections that follow, we show the minimum abbreviations after the first use of the command.

We recommend putting all configuration commands in a file, so that you can run them automatically if you have to reinitialize NQS or set up NQS on another partition manager. For example, if the commands are in the file **setup.qmgr**, you could issue the following command:

```
% qmgr < setup.qmgr
```

2.3 Who Can Issue qmgr Commands

There are three privilege levels for issuing **qmgr** commands:

- NQS managers, as well as the superuser, can issue any **qmgr** command.
- NQS operators can issue a subset of **qmgr** commands. In general, they cannot issue commands to create and configure queues. They can, however, start and stop queues and shut down NQS.
- Users with neither manager nor operator privileges can issue informational NQS commands; see Section 2.13.

We recommend that you designate a sufficient number of NQS operators so that an operator is always available when queues are running, in case a problem arises.

As noted above, you must initially run the **qmgr** command as superuser; you can then specify yourself as a manager. Subsequently, you can issue any **qmgr** commands without being the superuser.

Use the **set managers** command (minimum abbreviation: **se man**) to specify a list of NQS managers and operators. Specify an account name or a user ID for each manager or operator. If the user is on a remote machine, use the format *userid@machine_name*. To make a user an NQS manager, add **:m** at the end; to make the user an operator, add **:o**. For example,

```
Mgr: set managers rich:m [1469]@gemstone:o josh:o
```

makes **rich** a manager and **josh** an operator; user **1469** on the remote computer **gemstone** is also made an operator.

Use the **add managers** command (minimum abbreviation: **ad m**) to add one or more managers or operators, using the same format. Use the **delete managers** command (minimum abbreviation: **de m**) to take away manager or operator privileges from specified users.

2.4 Configuring Batch Queues — An Overview

Batch queues are the basic mechanism for providing batch access to a CM. Below is a **qmgr** session that sets up a batch queue called **cmq1**. You would set up the queue on the partition manager where you want the requests to run.

```
% qmgr
Mgr: create batch_queue cmq1 priority=32
Mgr: set restriction_window cmq1 start_time=(18:00)\
     stop_time=(6:00) mode=(timeonly)
Mgr: enable queue cmq1
Mgr: exit
```

After each command, the system responds:

```
NQS manager[TCML_COMPLETE]: Transaction complete at local host.
```

Let's look in more detail at the commands.

```
create batch_queue cmq1 priority=32
```

creates the queue **cmq1**. The **priority** argument sets the queue's priority among other batch queues that might be running at the same time.

```
set restriction_window cmq1 start_time=(18:00)\
stop_time=(6:00) mode=(timeonly)
```

specifies when the queue is active. In this case, it comes up at six o'clock every night and runs till six o'clock in the morning.

```
enable queue cmq1
```

makes the queue available to accept requests from users. The requests are not executed until the queue starts.

```
exit
```

causes you to leave the **qmgr** environment. Your UNIX prompt is displayed once again.

2.5 Creating, Enabling, and Starting a Batch Queue

There are three steps to making a batch queue operational:

1. Create the queue, using the **create batch_queue** command.
2. Enable the queue, using the **enable queue** command. This allows the queue to accept batch requests.
3. Start the queue, using the **start queue** or the **set restriction_window** command.

2.5.1 The create batch_queue Command

Use the **create batch_queue** command (minimum abbreviation: **c b**) to create a batch queue.

create batch_queue takes a **priority** as an argument (minimum abbreviation: **pr**); the priority must be an integer between 0 and 63, inclusive. This number specifies the *interqueue priority* of this queue — that is, its priority in relation to other batch queues running on this computer at the same time.

For example,

```
Mgr: create batch_queue cmq1 priority=63
```

creates a queue called **cmq1** with a priority of 63 — the highest interqueue priority.

NOTE: If the batch queue will be used by a pipe queue (described in Section 2.11), be careful in naming the queue so that the pipe queue will operate in the manner you desire; see Section 2.11.2.

There are two optional arguments to **create batch_queue**:

- **run_limit** (minimum abbreviation: **r**). The run limit specifies the maximum number of requests allowed to run in this queue at any given time; the default is 1.
- **pipeonly** (minimum abbreviation: **pi**). Use this argument if you want the batch queue to accept requests only if they come from a pipe queue. If you are using pipe queues, you may want to use this option for your batch queues, since it ensures that users will not bypass the pipe queue mechanism and submit jobs directly to batch queues.

Manager privileges are required to issue the **create batch_queue** command.

2.5.2 The enable queue Command

Use the **enable queue** command (minimum abbreviation: **en q**) to enable a queue; this permits the queue to accept batch requests. The queue must also be "started" for it to run queued requests, as described below.

Operator privileges are required for the **enable queue** command.

2.5.3 The start queue and set restriction_window Commands

There are two commands that start a queue:

- Issue the **start queue** command (minimum abbreviation: **sta q**) to start the queue immediately and make requests eligible to run; use this command when the queue is to operate continuously.
- Use the **set restriction_window** command (minimum abbreviation: **se rest**) to set the times or dates when the queue is to start and stop. The queue starts automatically when the specified starting time arrives, and stops when the specified stopping time arrives. Its arguments are described below.

NOTE: You can use either of these commands to start a batch queue; you can use only the **start queue** command to start a pipe queue.

Operator privileges are required to issue the **start queue** and **set restriction_window** commands.

The **set restriction_window** command takes the following arguments (all except **term** are required):

- **start_time** and **stop_time** (minimum abbreviations: **sta** and **sto**). Use these arguments to specify when the queue is to start and when it is to stop. You can specify either a time (in the form *hh:mm:ss*, and assuming the current day) or a date with or without a time. Specify the date by indicating *either* the day of the week — *Fri*, for example — or the date — *17 Feb*, for example. (Specifying both the day of the week and the date will generate an error message.) If you do not specify a time with the date, 0:00:00 is assumed. You can also specify the date as *today*, *tomorrow*, or *yesterday*. For information about time, day, and date syntax, please see the **qmgr** manual page.

  To have the queue start immediately, specify a start time that is prior to the current time.

- **mode** (minimum abbreviation: **m**). Specify **mode=(timeonly)** if **start_time** and **stop_time** specify only times. Specify **mode=(timedate)** if the arguments include a date as well as a time.

For example,

```
Mgr: set restriction_window cmq1 \
     start_time=(00:00:00) stop_time=(08:00:00)\
     mode=(timeonly)
```

specifies that **cmq1** is to start at midnight and stop at eight o'clock in the morning every day.

```
Mgr: set restriction_window cmq1 \
     start_time = (feb 17 18:00:00) \
     stop_time = (feb 20 08:00:00) mode=(timedate)
```

specifies that the queue is to start at six o'clock on the evening of February 17th, and stop at eight o'clock on the morning of February 20th.

- **term** (minimum abbreviation: **t**). If you specify **term=(true)**, NQS sends a SIGTERM signal to any processes that are executing when the queue is stopping.

  If **term=(false)**, NQS does not send a SIGTERM signal to the processes; this is the default. The processes are allowed to finish executing before the queue stops.

  In both cases, the queue ends up stopped but still enabled. Requests in the queue remain in the queue, but do not run. New requests can be added to the queue.

Deleting a Restriction Window

Use the **delete restriction_window** command (minimum abbreviation **del rest**) to remove a restriction window from a batch queue. For example,

```
Mgr: delete restriction_window cmq1
```

deletes the restriction window currently in effect for queue **cmq1**. You don't have to issue this command before creating a new restriction window; a new window

automatically supersedes the window previously in effect. Operator privileges are required to issue this command.

If you issue this command while the queue is running (that is, within a restriction window), the queue continues to run. Issue a **stop queue** command to stop the queue in this case.

If you issue this command while the queue is not running (that is, outside a restriction window), you must issue the command **start queue** or **set restriction_window** to start the queue.

2.6 Stopping, Disabling, and Deleting a Queue

For taking a queue out of operation, there are three commands analogous to the ones described in Section 2.5:

- To stop a queue, use the **stop queue** command (minimum abbreviation: **sto q**), specifying the name of the queue. Requests currently running are allowed to complete. Users can still add requests to the queue, but no more requests will run until the queue is started again.
- Use the **disable queue** command (minimum abbreviation: **di q**), along with a queue name, to prevent any more requests from being placed in the specified queue.
- Use the **delete queue** command (minimum abbreviation: **de q**), along with a queue name, to delete the specified queue. The queue must have no requests in it, and it must be disabled.

stop queue and **disable queue** are operator commands; **delete queue** requires full manager privileges.

In addition, an NQS operator can do the following:

- Issue the **purge queue** command (minimum abbreviation: **pu q**) to purge all requests from a specified batch queue. Requests that are currently running are allowed to complete; all other requests are lost and can't be recovered.

- Issue the **abort queue** command (minimum abbreviation: **ab q**) to abort all requests that are currently running in the specified queue. The command sends a SIGTERM signal to each running process. After a specified number of seconds (the default is 60), the command sends a SIGKILL signal to any remaining processes that are still running. For example,

```
Mgr: abort queue cmq1 120
```

aborts the processes running in **cmq1**, giving them 120 seconds between the time they are sent the SIGTERM signal and the time they are sent the SIGKILL signal. The aborted requests are deleted from the queue. Other requests remain in the queue and are eligible to run.

2.7 Who Can Use a Queue?

When a queue is first created, anyone can submit batch requests to it. The **qmgr** utility provides commands that let you restrict access to a queue.

The superuser can always use the queue, unless you issue the **disable queue** command to prevent any more requests from being placed in the queue.

To restrict access to the superuser, issue the **set no_access** command (minimum abbreviation: **se no_a**), specifying the name of the queue.

To subsequently add users to the access list for the queue, issue the **add users** command (minimum abbreviation: **ad u**) or **add groups** command (minimum abbreviation: **ad g**) instead. Use the **add users** command to specify a user, or a list of users, to the access list for a queue. You can specify them either by their user names or their user IDs. Put user IDs in square brackets; separate names or IDs with commas and put parentheses around the list. For example,

```
Mgr: add users=(lauranne, carl, janet, [1001]) cmq1
```

adds four users to the access list for **cmq1**.

Similarly, issue the **add groups** command to add specified groups to the access list for a batch queue. For example,

```
Mgr: add groups=(systems, research, software) cmq1
```

adds the users who belong to the specified groups to the batch queue **cmq1**.

NOTE: If anyone can currently use the queue, you must first issue the **set no_access** command before issuing the **add users** or **add groups** command to specify users who are to have access to the queue.

You can delete individual users and groups by issuing the **delete users** and **delete groups** commands (minimum abbreviations: **de u** and **de g**).

If you have restricted access to the queue and you now want to let anyone use it, issue the **set unrestricted_access** command (minimum abbreviation: **se u**), specifying the name of the queue. For example:

```
Mgr: set unrestricted_access cmq1
```

2.8 Setting Limits

NQS allows you to specify a variety of limits on the resources that a batch request (or an individual process in a request) can use in a specified batch queue. Users can specify lower limits for individual processes within a batch request; they cannot, however, specify higher limits.

The limits listed below are meaningful for batch queues running on a partition manager. See the **qmgr** man page for complete information on the limit commands and on how to specify limit values, or issue the **help** command within the **qmgr** environment.

- **Core file size** — By default, the maximum per-process corefile size is unlimited. Use the **set corefile_limit** command (minimum abbreviation: **se cor**) to specify a limit.
- **CPU time limit** — By default, the maximum amount of front-end CPU time a process can use is 36000 seconds. Use the **set per_process cpu_limit** command (minimum abbreviation: **se per_p c**) to specify a different limit.

- **Data size** — By default, there is no maximum per-process data segment size. Use the **set data_limit** command (minimum abbreviation: **se da**) to specify a limit.

- **File size** — By default, the maximum per-process permanent file size on the partition manager is 1 megabyte. Use the **set per_process permfile_limit** command (minimum abbreviation: **se per_p p**) to specify a different limit for processes running in a specified batch queue.

- **Nice value** — By default, the nice value for a process is 0. Use the **set nice_value_limit** command (minimum abbreviation: **se ni**) to specify a different default. Nice values are integers in the range –20 to 20; a lower number specifies a higher execution priority.

- **Stack size** — By default, the maximum per-process stack segment size is 6 megabytes. Use the **set stack_limit** command (minimum abbreviation **se st**) to specify a different limit.

- **Working set limit** — By default, the working set limit (that is, the physical memory quota) for a process is 25 megabytes. Use the **set working_set_limit** command (minimum abbreviation: **se w**) to specify a different limit.

In all cases where it is applicable, the setting **unlimited** specifies that there is to be no limit on the resource.

NQS manager privileges are required to issue these commands.

2.9 Creating Queue Complexes

You can assign one or more batch queues to a *queue complex*, and you can assign an overall run limit to the queue complex. This can be useful if you have several queues, each with an individual run limit, but you want the overall run limit to be less than the total of the run limits for all the queues (for example, to keep the CM from being tied up).

For example, if **q1**, **q2**, and **q3** each have a run limit of 4, you could assign them to a queue complex with a run limit of 8; this guarantees that no more than eight requests will run at the same time.

Use the **create complex** command (minimum abbreviation: **c c**) to create a queue complex. Put the names of the batch queues in parentheses; separate their names with commas.

For example,

```
Mgr: create complex = (q1, q2, q3) qc1
```

creates a queue complex called **qc1**; by default, it has a run limit of 1.

To set or change the run limit associated with a queue complex, use the **set complex run_limit** command (minimum abbreviation: **se com r**). For example,

```
Mgr: set complex run_limit=8 qc1
```

changes **qc1**'s run limit from the default (1) to 8.

Use the **delete complex** command (minimum abbreviation: **de c**) to delete a complex; give the name of the complex as the only argument.

To add a queue to an existing queue complex, use the **add queues** command (minimum abbreviation: **ad q**). For example,

```
Mgr: add queues = (q4, q5) qc1
```

adds two queues to the queue complex **qc1**.

Use the **remove queue** command (minimum abbreviation: **r q**) to remove one or more queues from a queue complex. For example:

```
Mgr: remove queue = (q1, q2) qc1
```

2.10 Pipe Queues — An Overview

Pipe queues provide a convenient and flexible way of giving users access to multiple CM resources. A pipe queue can be associated with one or more destination batch queues. When the user submits a batch request to a pipe queue, the pipe queue tries each of its destination queues in turn, until it finds one that will run the request. A batch queue would reject the request if, for example, it is stopped, or it has reached its run limit. If at first no batch queue accepts the request, the pipe queue keeps cycling through its destination list until it finds a queue that will accept it.

Once a batch queue accepts the request, it is treated like any request submitted directly to the queue. After the request has run, the results are returned to the user in the normal way, as described in Chapter 1.

Using pipe queues in combination with batch queues offers considerable advantages over using batch queues alone:

- **From the user's perspective.** Users need not worry about searching for the batch queue on which their requests will run the soonest; the pipe queue can take care of this automatically. For example, a user might submit a request to a pipe queue that is associated with four batch queues, each of which is in turn associated with one partition of a CM, and each of which has a run limit of 1. The pipe queue will cycle through each of its associated batch queues until it finds one that is available; this is the queue to which the user's request will be submitted.

  In addition, users need not be logged in to a partition manager in order to use a CM. They can have a pipe queue set up on their local computer, and submit their requests to it. The pipe queue will communicate with the batch queue on the remote computer.

- **From the system administrator's perspective.** The use of pipe queues ensures that the load on the CM resources is evenly balanced. Requests on one batch queue will not be backed up while another batch queue lies idle. A pipe queue, unlike a batch queue, can be created on a computer that is not a partition manager; as long as it can reach a partition manager via a network, it can use the CM.

To create and configure a pipe queue, follow these steps:

1. If NQS has not been installed on the computer on which you want to create the pipe queue, you must first install it.

2. As part of installing NQS, you must run the **nmapmgr** utility on the computer on which the pipe queue is to run, and on all computers with which the pipe queue is to communicate. This utility creates a database of mappings between computers. See Section 2.12.6.

3. You then use the **qmgr** utility as you do for batch queues, to create, enable, and start the pipe queue. See Section 2.11.

2.11 Creating, Configuring, and Managing Pipe Queues

This section describes how to use **qmgr** commands to make an NQS pipe queue operational. See Section 2.2 for information on **qmgr**.

The only new command introduced in this section is **create pipe_queue**, which we discuss in Section 2.11.2.

2.11.1 An Example

The following **qmgr** commands create, enable, and start a pipe queue called **pipe1** that is associated with batch queues **express_q**, **short_q**, **medium_q**, and **long_q**:

```
Mgr: create pipe_queue pipe1 priority = 32 server = \
     (/usr/etc/pipeclient) destination = \
     (express_q, short_q,medium_q, long_q) run_limit = 5
Mgr: enable queue pipe1
Mgr: start queue pipe1
```

As mentioned in Section 2.4, we recommend putting all **qmgr** configuration commands in a file so that you can run them automatically.

2.11.2 Creating a Pipe Queue: The create pipe_queue Command

Use the **create pipe_queue** command (minimum abbreviation: **c p**) to create a pipe queue. Specify the name of the queue, followed by required and optional arguments. You must be an NQS manager to issue this command.

The following arguments are required:

- **priority** (minimum abbreviation: **pr**). The **priority** argument specifies the interqueue priority for this queue. It must be an integer between 0 and 63, inclusive; 63 is the highest priority. The interqueue priority for pipe queues specifies the order in which NQS handles requests in the pipe queues; all requests in the higher-priority queue are serviced before requests in the lower-priority queue.

- **server** (minimum abbreviation: **s**). The server is the program binary for pipe queues. Specify this server as **/usr/etc/pipeclient**.

- One or more **destinations** (minimum abbreviation: **d**). These destinations are the queues to which requests sent to this queue are in turn sent. Each destination can be either a batch queue or another pipe queue. The queues can be on a remote computer or on the computer on which this pipe queue is to exist. Separate the destination queues with commas. If there is more than one destination (there is no limit to the number of destination queues), enclose the list in parentheses. For example:

```
destination = (express_q, short_q, medium_q, \
long_q@remote.think.com)
```

When **create pipe_queue** is executed, the destination queues are sorted into two groups by whether they are local or remote. Each group is then sorted into alphabetical order.

When a pipe queue receives a request, it attempts to submit it to the first queue in the group of local queues. If that queue does not accept the request (for example, because it has reached its run limit), the pipe queue then tries the second local queue, and so on, until it finds a queue that accepts the requests. If no local queue accepts the request, the pipe queue then begins trying the remote queues: first the one that is alphabetically first, then the one that is alphabetically second, and so on, until it finds a queue that accepts the request. If no queue on the destination list accepts the request, the pipe queue waits five minutes, then tries again, starting with the first local queue. It keeps on trying for up to three days.

The local queues in the above destination list, for example, would be tried in the order **express_q**, **medium_q**, **short_q**.

Because the queues are tried in alphabetical order, you may have to finagle the names of the queues appropriately when you create them. For example, to get NQS to try first **express_q**, then **short_q**, then **medium_q**, and then **long_q**, you could create the queues as follows:

```
Mgr: create batch_queue AA_express_q, BB_short_q, \
CC_medium_q, DD_long_q priority=63
```

The following arguments are optional:

- **pipeonly** (minimum abbreviation: **pi**). This argument specifies that this pipe queue is to accept only requests coming from other pipe queues; users will not be allowed to submit requests directly to this queue.
- **run_limit** (minimum abbreviation: **r**). The run limit sets a ceiling on the number of requests allowed to run in the pipe queue at any given time. If you omit this argument, the default run limit is 1.

Here are two sample **create pipe_queue** commands:

```
Mgr: create pipe_queue pipeq pr=16 \
     s=(/usr/local/lib/nqs/pipeclient) d = batchq r = 5
Mgr: c p pipeq2 pr=32 server=(/usr/lib/nqs/pipeclient) \
     destination = (bq1, bq2, bq3)
```

2.11.3 Configuring and Managing Pipe Queues

To configure and start a pipe queue, you can use **qmgr** commands we have already discussed in connection with batch queues. For example:

- To specify who can use a queue, you can issue the **set no_access**, **set unrestricted_access**, **add users**, and **add groups** commands, as discussed in Section 2.7.
- To enable the queue, use the **enable queue** command discussed in Section 2.5.

- To start the queue, use the **start queue** command, as discussed in Section 2.5. Note that you can't use the **set restriction_window** command to start a pipe queue.
- Use the various commands listed in Section 2.6 for stopping, disabling, and deleting a pipe queue, and for purging and aborting requests in a pipe queue.

There are no resource limits, like those discussed in Section 2.8, associated with pipe queues. The limits in effect for a batch request submitted to a pipe queue are those of the batch queue that ultimately accepts the request.

2.12 Managing NQS

The **qmgr** utility contains several commands you can use to manage NQS as a whole, in addition to setting up and managing individual batch or pipe queues. These commands are described in Sections 2.12.1–2.12.5.

In addition, Section 2.12.6 describes the **nmapmgr** utility, which creates and updates a data base of mappings between the NQS computers.

2.12.1 Shutting Down NQS

To shut down NQS on a computer, run the script **stopnqs**.

This script executes the **shutdown** command (minimum abbreviation: **shu**), which sends a SIGTERM signal to each process of each request currently running. After a specified number of seconds (the default is 60), **shutdown** sends a SIGKILL signal to all the remaining processes. All the requests are put back in their queues, unless they terminated when they received the SIGTERM signal. For example,

```
shutdown 120
```

gives processes 120 seconds to respond to the shutdown before being killed.

An NQS operator can issue this command.

To start NQS back up again, run the **startnqs** command, as discussed in Section 2.1. You can't restart NQS until the specified shutdown time has expired.

2.12.2 Creating a Default Batch Queue

Use the **set default batch_request queue** command (minimum abbreviation: **se def b q**) to specify a default batch queue for batch requests. If a user submits a batch request without specifying the name of a queue, the request is sent to this default queue. For example:

```
Mgr: set default batch_request queue cmq1
```

makes **cmq1** the default batch queue.

Use the **set no_default batch_request queue** command (minimum abbreviation: **se no_d b q**) to subsequently specify that there is to be no default batch queue. You need not issue this command if you haven't already set a default queue.

2.12.3 Determining What Shell Is to Be Used

Use the **set shell_strategy** command (minimum abbreviation: **se sh**) to specify how to determine what shell is to be used to execute a batch request, in cases where the user does not specify a shell. There are three strategies: *free*, *fixed*, and *login*.

- Use the **free** argument (minimum abbreviation: **fr**) to specify that the user's login shell is to determine the shell to be used in executing a batch request; this is the default strategy. The login shell is the shell named within the user's entry in the password file. This shell is executed, and it in turn is instructed to examine the shell script file and spawn another shell of the appropriate type for interpreting the script. Thus, if the script specifies that the Bourne shell is to be executed, the login shell will execute that shell.

  This strategy therefore runs the script exactly as though the user had issued the commands in the script file interactively.

- Use the **fixed** argument (minimum abbreviation: **fi**) to specify a particular shell that is to be used to execute all batch requests. Specify the shell as an absolute pathname; enclose the pathname in parentheses. For example,

```
Mgr: set shell_strategy fixed=(/bin/csh)
```

  specifies that the C shell is to be used to execute all batch requests.

- Use the **login** argument (minimum abbreviation: **l**) to specify that the user's login shell is to be used to execute a batch request.

The fixed and login strategies exist for computers that are short on available free processes. In these two strategies, a single shell is executed, and that same shell executes all the commands in the batch request script. In the free strategy, two shells are executed: the login shell and the shell spawned by the login shell.

2.12.4 Setting the Default Intraqueue Priority

A user can specify a priority when issuing a batch request. This priority determines where the request is placed in the batch queue relative to other requests. You can issue the **set default batch_request priority** command (minimum abbreviation: **se def b pr**) to specify a default priority for all requests that do not include a priority. As an argument, specify an integer between 0 and 63, inclusive; 63 is the highest priority. For example,

```
Mgr: set default batch_request priority 58
```

specifies a default priority of 58. If you don't specify a default priority for the queue, its priority is 31.

Note the difference between *interqueue* and *intraqueue* priorities:

Interqueue priority determines the order with which requests from different queues are run. The NQS manager sets the interqueue priority for a queue with the **priority** argument of the **create batch_queue** and **create pipe_queue** commands.

Intraqueue priority determines the order with which requests in the same batch queue are run. The NQS manager sets a default for this priority with the **set default batch_request priority** command; the user can override this default when submitting the request.

2.12.5 Logging Error Output

To move NQS error output to a file other than **/usr/spool/nqs/log.daemon**, you can either:

- Edit the **startnqs** script, or
- Use the **qmgr** command **set log_file** *filename*. Full NQS manager privileges are required to use this command.

Problems are easier to diagnose if this file is left in its expected location, however.

To specify the amount of NQS debugging output that will be logged, use the **qmgr set debug** *level* command, where *level* is 0, 1, or 2:

0	No debug
1	Minimum debug
2	Maximum debug

By default, the debugging level is set to 0.

2.12.6 Running the nmapmgr Utility

Use the **nmapmgr** utility to provide the mapping between NQS computers. For NQS to be installed on a computer, its **nmapmgr** database must at least have an entry for that computer. If a computer has a pipe queue, its **nmapmgr** database must contain entries for itself and for every computer that contains a queue with which it wants to communicate. Similarly, a computer with a batch queue must have entries for itself and for any pipe-queue computers that are to communicate with it.

The Machine ID

Each entry in the **nmapmgr** database contains the full hostname of a computer and an associated machine ID, which is a number you assign to the computer via the **nmapmgr** utility. If you do not know the machine ID of a particular computer, log on to a computer that has an entry for that computer in its database and issue the following command:

```
% nmapmgr
```

At the **nmapmgr** prompt, issue the command **get mid**, followed by the full hostname of the computer. For example:

```
NMAPMGR>: get mid tuntu.think.com
```

nmapmgr responds with the machine ID of the computer.

If the computer has not yet been assigned a machine ID, you can assign it one, as described below; any integer will do, as long as it is not the machine ID of another computer.

NOTE: A computer must be identified by the same machine ID in every **nmapmgr** database that references it.

Adding an Entry to the nmapmgr Database

To add an entry to a computer's **nmapmgr** database, follow these steps:

1. Log in as root and issue the **nmapmgr** command:

```
# nmapmgr
```

2. *If this is the first time you are adding an entry to the database,* issue the **create** command at the **NMAPMGR>:** prompt to create the database:

```
NMAPMGR>: create
```

If the database has already been created, skip this step.

3. At the **NMAPMGR>:** prompt, issue the **add mid** command. Its arguments are the machine ID of the computer you are adding and the computer's hostname. For example,

```
NMAPMGR>: add mid 57 argus.cross.com
```

adds the computer **argus.cross.com**, with a machine ID of 57, to the **nmapmgr** database of the computer from which you issued the command.

4. When you are finished adding entries, issue the command **exit** to return to your UNIX prompt:

```
NMAPMGR>: exit
#
```

As mentioned above, the database for a computer must contain:

- An entry for itself
- An entry for each computer with which it is to communicate via NQS

For example, if you create a pipe queue on **argus.cross.com**, and its destinations are batch queues on **jason.cross.com**, **scylla.cross.com**, and **hercules.cross.com**, the **nmapmgr** data base on **argus** must contain entries for **argus** itself and for **jason**, **scylla**, and **hercules**. In addition, the databases on **jason**, **scylla**, and **hercules** must all contain entries for **argus**. All databases must use the same machine ID when referring to a computer.

To delete an entry from the database, use the **nmapmgr** command **delete mid**. For example,

```
NMAPMGR>: delete mid 57
```

removes the entry for the computer with machine ID 57 from this computer's **nmapmgr** database.

2.13 Obtaining Information

You can use the **shownqs** script to find all running NQS daemon processes.

In addition, the **Show All** command is one form of the **show** command (minimum abbreviation: **sho**), which is used in the **qmgr** environment to obtain information about various aspects of NQS. No privileges are required to issue this command. Table 3 shows the various forms of the **show** command. In addition to **show**, there is a **help** command that displays information about **qmgr** commands.

Table 3. qmgr show commands.

Command	Use
SHOw All	Display information about limits, managers, parameters, and queues.
SHOw LImits_supported	Display resource limits supported on this partition manager
SHOw LOng Queue [*queue* [*user*]]	Display in long format the status of all queues, a specified queue, or requests from a specified user.
SHOw Managers	Display the list of NQS managers.
SHOw Parameters	Display the general NQS parameters.
SHOw Queue [*queue* [*user*]]	Display the status of all queues, a specified queue, or requests from a specified user.

The **show** command cannot provide information about a queue running on a remote host. To obtain information about a queue running on a remote host, execute the **qstat** *qname@hostname* command from a shell (not from within the **qmgr** environment). If *@hostname* is not specified, the local host is assumed.

If *hostname* specifies a remote host, the remote host's name must appear *explicitly* in the local host's **/etc/hosts.equiv** file. If it does not appear explicitly, this error will be generated:

```
No account authorization at transaction peer.
```

Also, the local machine and the remote host must have a **nmapmgr** database that includes the machine IDs of both the local machine and the remote host. See Section 2.12.6 for more information about **nmapmgr** databases and machine IDs.

As a system administrator, most likely you will use either the **-a** option to **qstat**, which displays the status of all requests on the specified queue, or the **-u** *user-name* option, which displays the status only of all requests belonging to *user-name* on the specified queue.

The **-f** option is also useful; it displays extensive information about a queue; see Chapter 1 for sample output.

2.13.1 Obtaining Accounting Information

NQS provides accounting information for requests that execute in batch queues. Binary records for the start and end of each batch request are written in the file **/usr/adm/nqs**. You don't have to do anything to turn accounting on; the information is provided by default. You must, however, create your own postprocessor to analyze the information in the file.

Here is the header file that specifies the format for these records:

```
#define NQSACCT_FILE    "/usr/adm/nqs" /* file for NQS account records */
#define NQSACCT_INIT    1              /* job initiation record */
#define NQSACCT_FIN     2              /* job termination record */

/*
 *      NQS account file record header.
 */
struct  nqsacct {
        int     type;                  /* type of record */
        int     length;                /* length of record (bytes) */
        int     jobid;                 /* UNICOS job id */
};

/*
 *      NQS account file record for job initiation.
 */
struct  nqsacct_init {
        struct  nqsacct h;      /* NQS account record header */
        char    user[16];              /* user name */
        char    queue[16];             /* NQS queue name */
        int     priority;              /* priority of NQS queue */
        long    sub_time;              /* time submitted */
```

```
        long    start_time;             /* requested start time "-a" */
        long    init_time;              /* time initiated */
        int     orig_mid;               /* originating machine ID */
        char    scp_tid[8];             /* station TID */
        char    scp_mf[2];              /* station mainframe ID */
        long    fill;
};

/*
 *      NQS account file record for job termination.
 */
struct  nqsacct_fin {
        struct  nqsacct h;              /* NQS account record header */
        char    user[16];               /* user name */
        char    queue[16];              /* NQS queue name */
        int     priority;               /* priority of NQS queue */

        long    s_sec;                  /* System time in seconds */
        long    s_usec;                 /* System time in microseconds */
        long    u_sec;                  /* User time in seconds */
        long    u_usec;                 /* User time in microseconds */
        int     orig_mid;               /* Originating Machine ID */
};
```

Note these points in interpreting the records:

- UNICOS job ID, station TID, and mainframe ID are meaningless in the CM-5 environment.
- "Time initiated" is the time at which the request started executing.

Appendix

This appendix contains the UNIX manual pages for NQS commands. These man pages are also available on-line.

NAME

qcmplx – display status of NQS complex(es)

SYNOPSIS

qcmplx [–**h**host-name] [–**n**] [–**Q**]
[complex-name] [complex-name@host-name]

DESCRIPTION

Qcmplx displays the Network Queueing System (NQS) complexes.

In the absence of a *-h host-name* specifier, the local host is assumed.

Each entry displays the complexes on a given host. The *-Q* option displays the queues within the complex. The *-n* option eliminates the qcmplx header display.

CAVEATS

NQS is not finished, and continues to undergo development. This command may or may not be supported on all of your machines in the network.

SEE ALSO

qdel(1), qdev(1), qlimit(1), qpr(1), qstat(1), qsub(1), qmgr(1M)

NPSN HISTORY

Origin: Sterling Software Incorporated

August 1985 - Brent Kingsbury, Sterling
Original release.

Feb. 1990 – Terrie Carver, Computer Science Corperation
Second release.

NAME

qdel – delete or signal NQS request(s).

SYNOPSIS

qdel [–k] [–h hostname] [–signo] [–u username] request-id ...

DESCRIPTION

Qdel deletes all queued NQS requests whose respective *request-id* is listed on the command line. Additionally, if the flag **-k** is specified, then the default signal of **SIGKILL** (-9) is sent to any running request whose *request-id* is listed on the command line. This will cause the receiving request to exit and be deleted. If the flag **–h** hostname is requested then the action will be taken on the given host. If the flag –signo is present, then the specified signal is sent instead of the **SIGKILL** signal to any running request whose *request-id* is listed on the command line. In the absence of the **-k** and –signo flags, *qdel* will **not** delete a *running* NQS request.

To delete or signal an NQS request, the invoking user *must* be the owner; namely the submitter of the request. The only exception to this rule occurs when the invoking user is the *superuser*, or has NQS operator privileges as defined in the NQS manager database. Under these conditions, the invoker may specify the **–u** username flag which allows the invoker to delete or signal requests owned by the user whose account name is *username*. When this form of the command is used, **all** *request-ids* listed on the command line are presumed to refer to requests owned by the specified user.

An NQS request is always uniquely identified by its *request-id*, no matter where it is in the network of the machines. A *request-id* is always of the form: *seqno* or *seqno.hostname* where *hostname* identifies the machine from whence the request was originally submitted, and *seqno* identifies the sequence number assigned to the request on the originating host. If the *hostname* portion of a *request-id* is omitted, then the local host is always assumed.

The *request-id* of any NQS request is displayed when the request is first submitted (unless the *silent* mode of operation for the given NQS command was specified). The user can also obtain the *request-id* of any request through the use of the *qstat*(1) command.

CAVEATS

When an NQS request is signalled by the methods discussed above, the proper signal is sent to *all* processes comprising the NQS *request* that are in the same *process group*. Whenever an NQS request is spawned, a new *process group* is established for all processes in the request. However, should one or more processes of the request successfully execute a *setpgrp*() system call, then such processes will **not** receive any signals sent by the *qdel*(1) command. This can lead to "rogue" request processes which must be killed by other means such as the *kill*(1) command. For the UNIX implementations that support the ability to "lock" a process, and all of its progeny into a *process-group*, NQS will exploit this capability to prevent processes from "escaping" in this manner.

SEE ALSO

qcmplx(1), qdev(1), qlimit(1), qpr(1), qstat(1), qsub(1), qmgr(1M), kill(2), setpgrp(2), signal(2)

NPSN HISTORY

Origin: Sterling Software Incorporated

August 1985 – Brent Kingsbury, Sterling Software
Original release.

May 1986
Second release.

Feb. 1990 – Terrie Carver, Computer Sciences Corporation
Third release.

NAME

qdev – display status of NQS devices

SYNOPSIS

qdev [device-name] [device-name@host-name ...]

DESCRIPTION

Qdev displays the status of devices known to the Network Queueing System (NQS).

If no devices are specified, then the current state of each NQS device on the local host is displayed. Otherwise, the response is limited to the devices specified. Devices may be specified either as *device-name* or *device-name@host-name*. In the absence of a *host-name* specifier, the local host is assumed.

A *device header* with several headings is displayed for each of the selected devices. The first heading in a device header appears as **Device:**, and is followed by the name of the device formatted as *device-name@host-name*. The second heading of **Fullname:** is followed by the full path name of the special file associated with the device. The third heading of **Server:** is followed by the command line which will be used to *execve(2)* the device server. The fourth heading of **Forms:** is followed by the forms configured for the device.

The final heading of **Status:** prefaces a display of the general device state. The general state of a device is defined by two principal properties of the device.

The first property concerns whether or not the device is willing to continue accepting queued requests. If it is, the device is said to be **ENABLED**. If the device is unwilling to continue accepting queued requests, and is idle, its state is **DISABLED**. A third state of **ENABLED/CLOSED** is used to describe a device that is unwilling to continue accepting queued requests, but is not yet idle.

The second principal property of a device concerns whether or not the device is busy. There are three cases. If the device is busy, it is said to be **ACTIVE**. If the device is idle and not known to be out of service, it is said to be **INACTIVE**. Finally, if the device is idle and known to be out of service, it is said to be **FAILED**. **FAILED** covers both hardware and software failures.

If a device is busy, information about the active request follows the device header. The *request-name*, *request-id*, and the name of the user who submitted the request are all displayed.

SEE ALSO

qdel(1), qlimit(1), qpr(1), qstat(1), qsub(1), qmgr(1M)

NPSN HISTORY

Origin: Sterling Software Incorporated

May 1986 – Robert Sandstrom, Sterling Software
Original release.

NAME

qlimit – show supported batch limits and shell strategy for the named host(s).

SYNOPSIS

qlimit [host-name ...]

DESCRIPTION

Qlimit displays the set of batch request resource limit types that *can* be directly enforced on the implied local host or named hosts, and also the *batch request shell strategy* defined for the implied local host or named hosts.

If no *host-names* are given, then the information displayed is only relevant to the local host. Otherwise, the supported batch request limits, and *batch request shell strategy* for each of the named hosts is displayed.

NQS supports many batch request resource limit types that can be applied to an NQS batch request. However, not all UNIX implementations are capable of supporting the rather extensive set of limit types that NQS provides.

The set of limits applied to a batch request, is always restricted to the set of limits that can be directly supported by the underlying UNIX implementation. If a batch request specifies a limit that cannot be enforced by the underlying UNIX implementation, then the limit is simply ignored, and the batch request will operate as though there were no limit (other than the obvious physical maximums), placed upon that resource type.

When an attempt is made to queue a batch request, each *limit-value* specified by the request (that can also be supported by the local UNIX implementation), is compared against the corresponding *limit-value* as configured for the destination batch queue. If the corresponding batch queue *limit-value* for all batch request *limit-values is defined as unlimited*, or is *greater than* or *equal to* the corresponding batch request *limit-value*, then the request can be successfully queued, provided that no other anomalous conditions occur. For request *infinity limit-values*, the corresponding queue *limit-value* must also be configured as infinity.

These resource limit checks are performed irrespective of the batch request arrival mechanism, either by a direct use of the *qsub*(1) command, or by the indirect placement of a batch request into a batch queue via a *pipe* queue. It is impossible for a batch request to be queued in an NQS batch queue if *any* of these resource limit checks fail.

Finally, if a request fails to specify a *limit-value* for a resource limit type that is supported on the execution machine, then the corresponding *limit-value* as configured for the destination queue, becomes the *limit-value* for the unspecified request limit.

Upon the successful queueing of a request in a batch queue, the set of limits under which the request will execute is frozen, and will not be modified by subsequent *qmgr*(1M) commands that alter the limits of the containing batch queue.

As mentioned above, this command also displays the *shell strategy* as configured for the implied local host, or named hosts. In the absence of a *shell specification* for a batch request, NQS must choose which shell should be used to execute that batch request. NQS supports three different algorithms, or *strategies* to solve this problem that can be configured for each system by a system administrator, depending on the needs of the user's involved, and upon system performance criterion.

The three possible shell strategies are called:

fixed,
free, and
login.

These shell strategies respectively cause the configured *fixed* shell to be exec'd to interpret all batch requests, cause the user's login shell as defined in the password file to be exec'd which in turn chooses and spawns the appropriate shell for running the batch shell script, or cause only the user's login shell

to be exec'd to interpret the script.

A shell strategy of *fixed* means that the same shell as chosen by the system administrator, will be used to execute **all** batch requests.

A shell strategy of *free* will run the batch request script *exactly* as would an interactive invocation of the script, and is the default NQS shell strategy.

The strategies of *fixed*, and *login* exist for host systems that are short on available free processes. In these two strategies, a single shell is exec'd, and that same shell is the shell that executes all of the commands in the batch request shell script.

When a shell strategy of *fixed* has been configured for a particular NQS system, then the "fixed" shell that will be used to run **all** batch requests at that host is displayed.

SEE ALSO

qdel(1), qdev(1), qpr(1), qstat(1), qsub(1), qmgr(1M)

NPSN HISTORY

Origin: Sterling Software Incorporated

May 1986 – Brent Kingsbury, Sterling Software
Original release.

NAME

qmgr – NQS queue manager program

SYNOPSIS

qmgr

DESCRIPTION

Qmgr is a program used by the System Administrator or System Operator to control NQS requests, queues, devices, and the general NQS configuration at the local machine.

Definitions

An NQS *request* is a request by a user or user program to perform a function that requires a delay in servicing (e.g., after a certain time). Examples of such functions are the the scheduling of a shared serial-access resource (e.g., a printer), and the scheduling of batch job requests. A *device queue* holds requests for resources such as printers and Computer Output Microfilm (COM) units. A *batch queue* holds requests for scheduled, perhaps delayed, processing by various subsystems in the NPSN. A *pipe queue* is a queue which can pass queued requests on to other *pipe queues*, *batch queues*, or *device queues*. An NQS *device* is a site at which a shared serial-access resource such as a printer is offered. A *daemon* is a process which is designed to run continuously, providing some service when needed. (See the **QUEUE TYPES** section below for more information concerning queues.) Lastly, an NQS *manager* identifies a person who is capable of changing any NQS characteristic on the local machine. An NQS *operator* identifies a person who can execute only the *operator commands* as a proper subset of all the commands provided by the *qmgr*(1m) utility.

Commands

The following paragraphs describe the syntax of each *Qmgr*(1m) command. All command keywords are recognized regardless of upper or lower case usage. Keyword characters shown in uppercase indicate the *smallest possible abbreviation of the keyword* for the particular command being described.

ABort Queue *queue* [*seconds*]

All requests in the named *queue* that are currently running are aborted as follows. A SIGTERM signal is sent to each process of each request presently running in the named *queue*. After the specified number of *seconds* of real time have elapsed, a SIGKILL signal is sent to all remaining processes for each request running in the named *queue*. If a *seconds* value is not specified, then the delay is sixty seconds. All requests aborted by this command are deleted, and all output files associated with the requests are returned to the appropriate destination.

NQS *operator* privileges are required to use this command.

ADd Queues = (*queue* [, *queue* ...]) *complex*

Add the specified *queue(s)* to the batch queue complex named *complex*.

Full NQS *manager* privileges are required to use this command.

ADd DEStination = *destination queue*
ADd DEStination = (*destination* [, *destination* ...]) *queue*

The specified *destination(s)* are added as valid destinations for a pipe queue named *queue*.

Full NQS *manager* privileges are required to use this command.

ADd DEVice = *device queue*
ADd DEVice = (*device* [, *device* ...]) *queue*

The specified *device(s)* are added as resources to service requests from *queue*. The *device(s)* must exist (see **Create DEVICE** below).

Full NQS *manager* privileges are required to use this command.

ADd Forms *form-name ...*

The specified *form-name(s)* are added to the list of valid forms.

Full NQS *manager* privileges are required to use this command.

ADd Groups = *group queue*
ADd Groups = (*group* [, *group* ...]) *queue*

The specified *group(s)* are added to the access list for *queue*. There are two ways to specify a group:

group name
[*group id*]

Full NQS *manager* privileges are required to use this command.

ADd Managers *manager ...*

The specified *manager(s)* are added to the list of authorized NQS managers with privileges as specified. A *manager* specification consists of an account name specification, followed by a colon, followed by either the letter *m* or the letter *o*. There are four ways to specify an account name:

local_account_name
[*local_user_id*]
[*remote_user_id*]@*remote_machine_name*
[*remote_user_id*]@[*remote_machine_mid*]

If the account name specification is followed by *:m*, then the account is designated as an NQS *manager* account, capable of using all **qmgr** commands. If the account name specification is followed by *:o*, then the account is designated as an NQS *operator* account, capable of only using those commands appropriate for an NQS *operator*.

Full NQS *manager* privileges are required to use this command.

ADd Users = *user queue*
ADd Users = (*user* [, *user* ...]) *queue*

The specified *user(s)* are added to the access list for *queue*. There are two ways to specify a user:

user name
[*user id*]

Full NQS *manager* privileges are required to use this command.

Create Batch_queue *queue* **PRiority** = *n* [**PIpeonly**]
[**Run_limit** = *n*]

Define a batch queue named *queue* with inter-queue priority *n* (0..63). If **PIpeonly** is specified, then requests may enter this *queue* only if their source is a pipe queue. The specification of a **Run_limit** sets a ceiling on the maximum number of requests allowed to run in the batch queue at any given time. The default run-limit is one. (See the **QUEUE TYPES** section below for more information.)

Full NQS *manager* privileges are required to use this command.

Create Complex = (*queue* [, *queue* ...]) *complex*
Create a queue *complex* consisting of the specified set of batch *queues*. NQS provides for the grouping of a set of batch queues into a queue complex which can have an associated **Run_limit**.

Full NQS *manager* privileges are required to use this command.

Create DEVICE *device* **FOrms** = *forms* **FUllname** = *filename*
Server = (*server*)
Define a *device* with the specified *forms* and associate it with a *server*. This is done by specifying an absolute path name to the program binary (*server*) and any arguments required by the program. *Filename* is the absolute path name of the device (special file) and is typically **/dev/***device*.

Full NQS *manager* privileges are required to use this command.

Create DEVICE_queue *queue* **PRiority** = *n* [**Device** = *device*]
[**Device** = (*device* [, *device* ...])]
[**PIpeonly**]
Define a device queue named *queue* with inter-queue priority *n* (0..63). If **PIpeonly** is specified, then requests may enter this *queue* only if their source is a pipe queue. After **Device** appears a list of one or more *devices* that may service this *queue*. (See the **QUEUE TYPES** section below for more information.)

Full NQS *manager* privileges are required to use this command.

Create Pipe_queue *queue* **PRiority** = *n* **Server** = (*server*)
[**Destination** = *destination*]
[**Destination** = (*destination* [, *destination* ...])]
[**PIpeonly**] [**Run_limit** = *n*]
Define a pipe queue named *queue* with inter-queue priority *n* (0..63) and associate it with a *server*. This is done by specifying an absolute path name to the program binary (*server*) and any arguments required by the program. After **Destination** appears a list of one or more *destination* queues that requests from this pipe *queue* may be sent to. If **PIpeonly** is specified, then requests may enter this *queue* only if their source is a pipe queue. **Run_limit** sets a ceiling on the maximum number of requests allowed to run in the pipe queue at any given time. The default run-limit is one. (See the **QUEUE TYPES** section below for more information.)

Full NQS *manager* privileges are required to use this command.

DElete Complex *complex*
Delete a queue *complex*.

Full NQS *manager* privileges are required to use this command.

DElete DEStination = *destination queue*
DElete DEStination = (*destination* [, *destination* ...]) *queue*
Delete the mappings from the pipe queue *queue* to the *destination* queues. All requests from the named *queue* being transferred to a deleted *destination* complete normally. If all *destinations* for a pipe *queue* are deleted in this manner, then the pipe *queue* is effectively stopped.

Full NQS *manager* privileges are required to use this command.

DElete DEVice *device*

Delete the specified *device*. A device must be disabled to delete it from the device set (see **DIsable Device** below).

Full NQS *manager* privileges are required to use this command.

DElete DEVice = *device queue*
DElete DEVice = (*device* [, *device* ...]) *queue*

Delete the mappings from the device queue *queue* to the *device(s)*. All requests from the named device *queue* running on any of the named *devices* are allowed to complete normally. If ALL queue-to-device mappings for the named device *queue* are removed by this command, then the *queue* is effectively stopped.

Full NQS *manager* privileges are required to use this command.

DElete Forms *form-name* ...

The specified *form-name(s)* are deleted from the list of valid forms.

Full NQS *manager* privileges are required to use this command.

DElete Groups = *group queue*
DElete Groups = (*group* [, *group* ...]) *queue*

The specified *group(s)* are deleted from the access list for *queue*. There are two ways to specify a group:

group name
[*group id*]

Full NQS *manager* privileges are required to use this command.

DElete Managers *manager* ...

The specified *manager(s)* are deleted from the list of authorized NQS managers. A *manager* specification consists of an account name specification, followed by a colon, followed by either the letter *m* or the letter *o*. There are four ways to specify an account name:

local_account_name
[*local_user_id*]
[*remote_user_id*]@*remote_machine_name*
[*remote_user_id*]@[*remote_machine_mid*]

If the account name specification is followed by *:m*, it is understood that the account is currently permitted to use all **qmgr** commands. If the account name specification is followed by *:o*, it is understood that the account is currently permitted to use only those commands appropriate for an operator to use. The *root* account always has full privileges.

Full NQS *manager* privileges are required to use this command.

DElete Queue *queue*

The *queue* is deleted. To delete a *queue*, no requests may be present in the *queue* and the *queue* MUST be disabled (see **DIsable Queue** below). Any queue-to-device mappings are updated accordingly.

Full NQS *manager* privileges are required to use this command.

DElete Request *requestid ...*

Delete the request(s) named by the *requestid(s)*. This command can delete both running and non-running requests. If a request is running, then all processes of the request are sent a SIGKILL signal.

NQS *operator* privileges are required to use this command.

DElete Users = *user queue*
DElete Users = (*user* [, *user* ...]) *queue*

The specified *user(s)* are deleted from the access list for *queue*. There are two ways to specify a user:

user name
[*user id*]

Full NQS *manager* privileges are required to use this command.

DIsable Device *device*

The current request will complete. After that, the *device* is prevented from handling any more requests until it is enabled (see **ENable Device** below). If the disabled *device* was the last enabled device in a queue-to-device mapping, then the device queue is effectively stopped.

NQS *operator* privileges are required to use this command.

DIsable Queue *queue*

Prevent any more requests from being placed in this *queue*.

NQS *operator* privileges are required to use this command.

ENable Device *device*

If the *device* is already enabled, then this is a no-op. Otherwise, the *device* becomes available to handle requests.

NQS *operator* privileges are required to use this command.

ENable Queue *queue*

If the *queue* is already enabled, then this is a no-op. Otherwise, the *queue* is enabled to accept new requests.

NQS *operator* privileges are required to use this command.

EXit

Exit from the NQS manager subsystem.

Help [*command*]

Get help information. **Help** without an argument displays information about what commands are available. **Help** with an argument displays information about that command. The command may be partially specified as long as it is unique. A more complete help request yields more detailed information.

The **Help** *command* provides information that is often more extensive than the command descriptions in this manual page! Use it.

Lock Local_daemon

Lock the NQS local daemon into memory. See **plock(2)**.

NQS *operator* privileges are required to use this command.

MODify Request [Nice_limit = *nice* **] [RTime_limit** = *Tlimit* **]**
[RMemory_limit = *Mlimit* **]** *requestid*

Modify parameters for the request specified by *requestid*. *Nice* is the initial nice value for the request. *Tlimit* is a per request CPU time limit. *Mlimit* is a per request memory limit. For the syntax of these limits, see the **LIMITS** section below.

NQS *operator* privileges are required to use this command.

MOVe Queue *queue1 queue2*

Move all requests currently in *queue1* to *queue2*.

NQS *operator* privileges are required to use this command.

MOVe Request *requestid ... queue*

Move the request(s) named by the *requestid(s)* to the named *queue*.

NQS *operator* privileges are required to use this command.

Purge Queue *queue*

All queued requests are purged (dropped) from the *queue* and are irretrievably lost. Running requests in the *queue* are allowed to complete.

NQS *operator* privileges are required to use this command.

Remove Queue = **(** *queue* **[** , *queue* **...** **]** **)** *complex*

Remove the specified *queue(s)* from the batch queue complex named *complex*.

Full NQS *manager* privileges are required to use this command.

SEt COMplex Run_limit = *run-limit complex*

Change the *run-limit* of an NQS queue *complex*. The *run-limit* determines the maximum number of requests that will be allowed to run in the queue *complex* at any given time.

NQS *operator* privileges are required to use this command.

SEt CORefile_limit = **(** *limit* **)** *queue*

Set a per-process maximum core file size *limit* for a batch *queue* against which the per-process maximum core file size limit for a request may be compared. If the local host does not support per-process core file size limits, then this command will report an error. Otherwise, every batch queue on the local host will have a per-process maximum core file size limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it will be given a grandfather clause. A request specifying a per-process core file size limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full NQS *manager* privileges are required to use this command.

SEt DAta_limit = **(** *limit* **)** *queue*

Set a per-process maximum data segment size *limit* for a batch *queue* against which the per-process maximum data segment size limit for a request may be compared. If the local host does not support per-process data segment size limits, then this command will report an error. Otherwise, every batch queue on the local host will have a per-process maximum data segment size limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it will be given a grandfather clause. A request specifying a per-process maximum data segment size limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full NQS *manager* privileges are required to use this command.

SEt DEBug *level*

Set the debug *level*. The following values are valid:

0	No debug
1	Minimum debug
2	Maximum debug

Full NQS *manager* privileges are required to use this command.

SEt DEFault Batch_request Priority *priority*

Set the default intra-batch-queue *priority*. This is NOT the UNIX execution time priority. This is the priority used if the user does not specify an intra-queue priority parameter on the **qsub(1)** command.

Full NQS *manager* privileges are required to use this command.

SEt DEFault Batch_request Queue *queue*

Set the default batch *queue*. This is the queue used if the user does not specify a queue parameter on the **qsub(1)** command.

Full NQS *manager* privileges are required to use this command.

SEt DEFault DEStination_retry Time *retry_time*

Set the default number of hours that can elapse during which time a pipe queue destination can be unreachable, before being marked as completely failed.

Full NQS *manager* privileges are required to use this command.

SEt DEFault DEStination_retry Wait *interval*

Set the default number of minutes to wait before retrying a pipe queue destination that was unreachable at the time of the last attempt.

Full NQS *manager* privileges are required to use this command.

SEt DEFault DEVice_request Priority *priority*

Set the default intra-device-queue *priority*. This is the priority used if the user does not specify an intra-queue priority parameter on the **qpr(1)** command.

Full NQS *manager* privileges are required to use this command.

SEt DEFault Print_request Forms *form-name*

Set the default print forms to *form-name*. This is the forms used if the user does not specify a forms parameter on the **qpr(1)** command.

Full NQS *manager* privileges are required to use this command.

SEt DEFault Print_request Queue *queue*
Set the default print *queue*. This is the queue used if the user does not specify a queue parameter on the **qpr(1)** command.

Full NQS *manager* privileges are required to use this command.

SEt DEStination = *destination queue*
SEt DEStination = (*destination* [, *destination* ...]) *queue*
Associate one or more *destination* queues with a particular pipe *queue*.

Full NQS *manager* privileges are required to use this command.

SEt DEVICE = *device queue*
SEt DEVICE = (*device* [, *device* ...]) *queue*
Associate one or more *devices* with a particular *queue*.

Full NQS *manager* privileges are required to use this command.

SEt DEVICE_server = (*server*) *device*
Associate a *server* with a *device*. *Server* should consist of the absolute path name to the program binary followed by any arguments required by the program.

Full NQS *manager* privileges are required to use this command.

SEt Forms *form-name ...*
Specify the valid *form-name(s)*. Other valid forms may be added to this list (see **ADd Forms** above).

Full NQS *manager* privileges are required to use this command.

SEt Forms = *form-name device*
Set the *form-name* for a *device*.

Full NQS *manager* privileges are required to use this command.

SEt LIfetime *lifetime*
Set pipe-queue request *lifetime* in hours.

Full NQS *manager* privileges are required to use this command.

SEt LOg_file *filename*
Specify the name of the log file for NQS messages.

Full NQS *manager* privileges are required to use this command.

SEt MAIl *userid*
Specify the *userid* used to send NQS mail.

Full NQS *manager* privileges are required to use this command.

SEt MANagers *manager ...*

The list of authorized NQS managers is set to the specified *manager(s)*. A *manager* specification consists of an account name specification, followed by a colon, followed by either the letter *m* or the letter *o*. There are four ways to specify an account name:

local_account_name
[local_user_id]
[remote_user_id]@remote_machine_name
[remote_user_id]@[remote_machine_mid]

If the account name specification is followed by *:m*, then the account is designated as an NQS *manager* account, capable of using all **qmgr** commands. If the account name specification is followed by *:o*, then the account is designated as an NQS *operator* account, capable of only using those commands appropriate for an NQS *operator*. The *root* account always has full privileges. Also see **ADd Manager** above.

Full NQS *manager* privileges are required to use this command.

SEt MAXimum Copies *copies*

Set the maximum number of print *copies*.

Full NQS *manager* privileges are required to use this command.

SEt MAXimum Open_retries *retries*

Specify the maximum number of *retries* for a failed device open.

Full NQS *manager* privileges are required to use this command.

SEt MAXimum Print_size *size*

Specify the maximum *size* of an NQS print file in bytes.

Full NQS *manager* privileges are required to use this command.

SEt NEtwork Client = (*client* **)**

Specify the network client to be used. *Client* should consist of the absolute path name of the client followed by any arguments required by the client.

Full NQS *manager* privileges are required to use this command.

SEt NEtwork Daemon = (*daemon* **)**

Specify the network daemon to be used. *Daemon* should consist of the absolute path name of the daemon followed by any arguments required by the daemon.

Full NQS *manager* privileges are required to use this command.

SEt NEtwork Server = (*server* **)**

Specify the network server to be used. *Server* should consist of the absolute path name of the server followed by any arguments required by the server.

Full NQS *manager* privileges are required to use this command.

SEt NIce_value_limit = *nice-value queue*

Set the UNIX *nice-value* limit for a batch *queue*, against which the nice-value for a request may be compared. If a request already in the queue has asked for treatment more favorable than the new *nice-value*, then it will be given a grandfather clause. A request specifying a nice-value may only enter a batch queue if the queue's nice value is numerically less than (more willing to allow access to the CPU) or equal to the request's nice value. *Nice-value* is an integer preceded by an optional negative sign.

Full NQS *manager* privileges are required to use this command.

SEt NO_Access *queue*

Specify that no one will be allowed to place requests in *queue*. Root is an exception; requests submitted by root are always allowed into a queue, even if root is not explicitly given access.

Full NQS *manager* privileges are required to use this command.

SEt NO_Default Batch_request Queue

Indicate that there is to be no default batch request queue.

Full NQS *manager* privileges are required to use this command.

SEt NO_Default Print_request Forms

Indicate that there is to be no default print request forms.

Full NQS *manager* privileges are required to use this command.

SEt NO_Default Print_request Queue

Indicate that there is to be no default print request queue.

Full NQS *manager* privileges are required to use this command.

SEt NO_Network_daemon

Indicate that there is to be no network daemon.

Full NQS *manager* privileges are required to use this command.

SEt Open_wait *interval*

Specify the number of seconds to wait between failed device opens.

Full NQS *manager* privileges are required to use this command.

SEt PER_Process Cpu_limit = (*limit*) *queue*

Set a per-process maximum CPU time *limit* for a batch *queue* against which the per-process maximum CPU time limit for a request may be compared. If the local host does not support per-process CPU time limits, then this command will report an error. Otherwise, every batch queue on the local host will have a per-process maximum CPU time limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it will be given a grandfather clause. A request specifying a per-process maximum CPU time limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full NQS *manager* privileges are required to use this command.

SEt PER_Process Memory_limit = (*limit*) *queue*

Set a per-process maximum memory size *limit* for a batch *queue* against which the per-process maximum memory size limit for a request may be compared. If the local host does not support per-process memory size limits, then this command will report an error. Otherwise, every batch queue on the local host will have a per-process maximum memory size limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it will be given a grandfather clause. A request specifying a per-process maximum memory size limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full NQS *manager* privileges are required to use this command.

SEt PER_Process Permfile_limit = (*limit*) *queue*

Set a per-process maximum permanent file size *limit* for a batch *queue* against which the per-process maximum permanent file size limit for a request may be compared. If the local host does not support per-process permanent file size limits, then this command will report an error. Otherwise, every batch queue on the local host will have a per-process maximum permanent file size limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it will be given a grandfather clause. A request specifying a per-process maximum permanent file size limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full NQS *manager* privileges are required to use this command.

SEt PER_Process Tempfile_limit = (*limit*) *queue*

Set a per-process maximum temporary file size *limit* for a batch *queue* against which the per-process maximum temporary file size limit for a request may be compared. If the local host does not support per-process temporary file size limits, then this command will report an error. Otherwise, every batch queue on the local host will have a per-process maximum temporary file size limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it will be given a grandfather clause. A request specifying a per-process maximum temporary file size limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full NQS *manager* privileges are required to use this command.

SEt PER_Request Cpu_limit = (*limit*) *queue*

Set a per-request maximum CPU time *limit* for a batch *queue* against which the per-request maximum CPU time limit for a request may be compared. If the local host does not support per-request CPU time limits, then this command will report an error. Otherwise, every batch queue on the local host will have a per-request maximum CPU time limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it will be given a grandfather clause. A request specifying a per-request maximum CPU time limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full NQS *manager* privileges are required to use this command.

SEt PER_Request Memory_limit = (*limit*) *queue*

Set a per-request maximum memory size *limit* for a batch *queue* against which the per-request maximum memory size limit for a request may be compared. If the local host does not

support per-request memory size limits, then this command will report an error. Otherwise, every batch queue on the local host will have a per-request maximum memory size limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it will be given a grandfather clause. A request specifying a per-request maximum memory size limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full NQS *manager* privileges are required to use this command.

SEt PER_Request Permfile_limit = (*limit* **)** *queue*

Set a per-request maximum permanent file space *limit* for a batch *queue* against which the per-request maximum permanent file space limit for a request may be compared. If the local host does not support per-request permanent file space limits, then this command will report an error. Otherwise, every batch queue on the local host will have a per-request maximum permanent file space limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it will be given a grandfather clause. A request specifying a per-request maximum permanent file space limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full NQS *manager* privileges are required to use this command.

SEt PER_Request Tempfile_limit = (*limit* **)** *queue*

Set a per-request maximum temporary file space *limit* for a batch *queue* against which the per-request maximum temporary file space limit for a request may be compared. If the local host does not support per-request temporary file space limits, then this command will report an error. Otherwise, every batch queue on the local host will have a per-request maximum temporary file space limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it will be given a grandfather clause. A request specifying a per-request maximum temporary file space limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full NQS *manager* privileges are required to use this command.

SEt PIpe_client = (*client* **)** *queue*

Associate a *pipe client* with a pipe *queue*. *Client* should consist of the absolute path name to the program binary followed by any arguments required by the program.

Full NQS *manager* privileges are required to use this command.

SEt PRiority = *priority queue*

Specify the inter-queue *priority* of a *queue*.

Full NQS *manager* privileges are required to use this command.

SEt RESTriction_window *queue* **STArt_time = (** *time* **)**
STOp_time = (*time* **)**
Mode = (timeonly | timedate)
[Term = (true | false)]

Set a window during which a batch queue is to be operational. The window is specified by the **STArt_time** and **STOp_time** arguments; the times can be a time only--for example: 04:00:06--or a date with or without a time--for example, Jan 31 08:00:00. Specify "timeonly" if you use

only times; specify "timedate" if you include dates.

The syntax acceptable for the time/date specification is quite flexible, but there are some restrictions to eliminate possible ambiguities: If no date is specified, then the current day is assumed. If no time is specified, then the time of 0:00:00 for the local time zone is assumed.

A date can be specified as a month and numerical day (with the optional addition of a numerical year specification), or, alternatively, as the name of a weekday (for example, *Sunday*). It is illegal to specify a year without also specifying a month and day. It is illegal to specify a month and day (with optional year) *in addition to* specifying the name of a weekday, and vice versa. It is illegal to specify a month without also specifying the numerical day within the month, and vice versa.

A weekday is specified as any prefix of the name of a weekday that is three characters or longer, where the comparison is case insensitive. An optional period may follow a weekday name specification. Thus, *Sun.*, *sUN*, *Monda*, *Tues*, *tue*, and *tue.* are acceptable weekday-name specifications with the obvious interpretations. Three additional "weekday names" are also recognized regardless of case but must be completely spelled out to be recognized: *today*, *tomorrow*, *yesterday*.

The month portion of a month-and-day specification can be either numerical in the interval [1..12], or alphabetic, as the name of the month. Month names are recognized using rules identical for the recognition of weekday names. Thus, *Janu*, *jan*, *January*, *Jan.*, and *fEBrUAr* are all acceptable as month-name specifications with the obvious interpretations.

The permissible forms for specifying the numerical day and month are *MM/DD*, *Month DD*, *DD-Month*, where *Month* indicates that the name of the month has been specified (e.g. *Jan*).

If a year is specified as part of the time/date specification, then it must appear adjacent to the month and day specification, unless the year is specified as a four-digit number (e.g. 2001), in which case it can appear in any location within the time-and-date specification string, where the location must not violate any of the rules for a time-of-day or month/day specification. The permissable forms for specifying a year adjacent to the numerical date and month are *DD-Month-YY*, *DD-Month-YYYY*, *DD-Month YYYY*, *Month DD YYYY*, *MM/DD YYYY*, *MM/DD/YY*, *YYYY-MM-DD*, *YYYY DD-Month*, *YYYY Month DD*, and *YYYY MM/DD*.

A time-of-day specification can appear anywhere within the time-and-date specification string with the restrction that it cannot appear in places that would violate the syntax of a month/day (and optional year) specification. Thus, *22- 11pm January* is illegal, while *22-January, 11pm* is legal.

A time-of-day specification is allowed in the forms *HH <meridian>*,
HH:MM <meridian>, *HH:MM:SS <meridian>*, *noon*, and *midnight*, where *HH* is any one- or two-digit hour specification in the interval [0..24], *MM* is any one- or two-digit minutes specification in the interval [0..59], *SS* is any one- or two-digit seconds specification in the interval [0..59], *<meridian>* specifies *am*, *pm*, or *m* (meaning noon). A meridian is recognized regardless of uppercase and/or lowercase character usage. A meridian specification can also be optionally preceded with a dash. In the absence of a meridian specification, a 24-hour clock semantic interpretation is used.

The precise meanings of *am* and *pm* are not always agreed upon by the general populace when specifying times like 12am or 12pm; the interpretation of *am*, *pm*, and *m* used by **qmgr** are given by the table below.

Specified time:	24-hour clock interpretation:
0:00:00-11:59:59 AM	0:00:00-11:59:59
12:00:00 AM	0:00:00
12:00:01-12:59:59 AM	0:00:01-0:59:59
13:00:00-24:00:00 AM	Invalid
0:00:00-11:50:50 PM	12:00:00-23:59:59
12:00:00 PM	24:00:00
12:00:01-12:59:59 PM	12:00:01-12:59:59
13:00:00-24:00:00 PM	Invalid
0:00:00-11:59:59 M	Invalid
12:00:00-12:59:59 M	12:00:00-12:59:59
13:00:00-24:00:00 M	Invalid

Thus, these examples of time-of-day specifications are all legal: 11pm, 11:30-am, 12:00 (meaning noon), 11:59AM, 12:00-aM (meaning 0:00:00), 12:00-Pm (meaning 24:00:00), 12m (meaning noon), 23:00:01 (meaning 1 second past 11pm), 9 pm, 01:00:01am, noon, midnight.

set restriction_window does not allow a timezone specification. Specifying a timezone, e.g. EST, results in a syntax error. The local timezone is always used.

NQS handles daylight savings time according to the rules specified for the underlying system.

All tab, space, and comma characters are interpreted as whitespace and are, therefore, essentially ignored. A newline character, if encountered, terminates the time/date specification. Otherwise, the null character at the end of the string terminates the time-date specification.

Specify "term" for the optional **Term** argument to specify that NQS is to send a SIGTERM signal to a process that is executing when the queue is stopping. The default setting is "false": no signal is sent, and the process is allowed to finish executing before the queue stops. No matter what the setting, the queue ends up stopped but still enabled. Requests in the queue remain in the queue, but do not run. New requests can be added to the queue.

NQS *operator* privileges are required to use this command.

SEt Run_limit = *run-limit queue*

Change the *run-limit* of an NQS batch or pipe queue. The *run-limit* determines the maximum number of requests that will be allowed to run in the queue at any given time.

NQS *operator* privileges are required to use this command.

SEt SHell_strategy FIxed = (*shell*)

Specify that *shell* should be used to execute all batch requests. *Shell* must be the absolute path name of a command interpreter.

Full NQS *manager* privileges are required to use this command.

SEt SHell_strategy FRee

Specify that the **free** shell strategy should be used to execute all batch requests. The **free** shell strategy aims at duplicating the shell choice that would have been made if the batch request script had been executed interactively. Under this strategy, the user's *login shell* is allowed to determine the shell to be used to execute the batch request. The user's *login shell*

is the shell named within the user's entry in the password file (see **passwd(4)**).

Full NQS *manager* privileges are required to use this command.

SEt SHell_strategy Login
Specify that the **login** shell strategy should be used to execute all batch requests. Under the **login** shell strategy, the user's *login shell* is used to execute the batch request. The *login shell* is the shell named in the password file (see **passwd(4)**).

Full NQS *manager* privileges are required to use this command.

SEt STack_limit = (*limit* **)** *queue*
Set a per-process maximum stack segment size *limit* for a batch *queue* against which the per-process maximum stack segment size limit for a request may be compared. If the local host does not support per-process stack segment size limits, then this command will report an error. Otherwise, every batch queue on the local host will have a per-process maximum stack segment size limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it will be given a grandfather clause. A request specifying a per-process maximum stack segment size limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full NQS *manager* privileges are required to use this command.

SEt Unrestricted_access *queue*
Specify that no requests will be turned away from *queue* on the grounds of queue access restrictions.

Full NQS *manager* privileges are required to use this command.

SEt Working_set_limit = (*limit* **)** *queue*
Set a per-process maximum working set size *limit* for a batch *queue* against which the per-process maximum working set size limit for a request may be compared. If the local host does not support per-process working set size limits, then this command will report an error. Otherwise, every batch queue on the local host will have a per-process maximum working set size limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it will be given a grandfather clause. A request specifying a per-process maximum working set size limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full NQS *manager* privileges are required to use this command.

SHOw All
Display the standard amount of information about *devices*, *forms*, *limits supported*, *managers*, *parameters*, and *queues*. See below.

SHOw Device [*device-name*]
Display the status of all NQS devices on this host. If a *device-name* is specified, output will be limited to that device.

SHOw Forms
Display the list of valid forms.

SHOw Limits_supported
Display the list of NQS resource limit types which are meaningful on this machine. If a limit type is meaningful on a machine, then the corresponding **qmgr(1M)** commands will allow the association of a limit of that type with any batch queue on that machine. Note that users may request resource limits that are NOT meaningful on the machine where **qsub(1)** is invoked. If the the request is to be executed on a remote machine where the limit is meaningful, then NQS will honor it. Otherwise the unsupported limit is simply ignored.

SHOw LOng Queue [*queue-name* [*user-name*]]
Display in long format the status of all NQS queues on this host. If a *queue-name* is specified, output will be limited to that queue. If a *user-name* is specified, output will downplay any requests not belonging to that user.

SHOw Managers
Display the list of authorized NQS managers.

SHOw Parameters
Display the general NQS parameters.

SHOw Queue [*queue-name* [*user-name*]]
Display the status of all NQS queues on this host. If a *queue-name* is specified, output will be limited to that queue. If a *user-name* is specified, output will downplay any requests not belonging to that user.

SHUtdown [*seconds*]
Shutdown NQS on the local host. A SIGTERM signal is sent to each process of each request presently running. After the specified number of *seconds* of real time have elapsed, a SIGKILL signal is sent to all remaining processes for each request. If a *seconds* value is not specified, then the delay is sixty seconds. Unlike **ABort Queue, SHUtdown** requeues all of the requests it kills, provided that the initial SIGTERM signal is caught or ignored by the running request.

NQS *operator* privileges are required to use this command.

STArt Queue *queue*
If the *queue* is already started, then nothing happens. Otherwise, the *queue* is started and requests in the *queue* are eligible for selection.

NQS *operator* privileges are required to use this command.

STOp Queue *queue*
Any requests in the *queue* that are currently running are allowed to complete. All other requests are "frozen" in the *queue*. New requests can still be submitted to the *queue*, but will be "frozen" like the other requests in the *queue*.

NQS *operator* privileges are required to use this command.

Unlock Local_daemon
Remove a lock that has been keeping the NQS local daemon in memory. See **plock(2)**.

NQS *operator* privileges are required to use this command.

QUEUE TYPES
NQS supports four different queue types, that serve to provide four very different functions. These four

queue types are known as *batch*, *device*, *pipe*, and *network*.

The queue type of *batch* can only be used to execute NQS *batch requests*. Only NQS *batch requests* created by the *qsub*(1) command can be placed in a *batch queue*.

The queue type of *device* can only be used to execute NQS *device requests*. Only NQS *device requests* created by the *qpr*(1) command can be placed in a *device queue*.

Queues of type: *pipe*, are used to send NQS requests to other *pipe* queues, or to request destination queues of type *batch* or *device*, as appropriate for the request type. In general, *pipe queues* in combination with *network queues*, act as the mechanism that NQS uses to to transport both *batch* and *device* requests to distant queues on other remote machines. It is also perfectly legal for a *pipe queue* to transport requests to queues on the *same* machine.

When a *pipe queue* is defined, it is given a *destination set*, which defines the set of possible destination queues for requests entered in that *pipe queue*. In this manner, it is possible for a *batch* or *device* request to pass through many pipe queues on its way to its ultimate destination, which must eventually be a queue of type *batch* or *device* (matching the request type).

Each *pipe queue* has an associated *server*. For each request handled by a *pipe queue*, the associated server is spawned which must select a queue destination for the request being handled, based on the characteristics of the request, and upon the characteristics of each queue in the *destination set* defined for the pipe queue.

Since a different server can be configured for each pipe queue, and *batch* and *device* queues can be endowed with the *pipeonly* attribute that will only admit requests queued via another *pipe queue*, it is possible for respective NQS installations to use *pipe queues* as a *request class* mechanism, placing requests that ask for different resource allocations in different queues, each of which can have different associated limits and priorities.

It is also completely possible for a *pipe client* (pipe queue server) when handling a request, to discover that no *destination queue* will accept the request, for various reasons which can include insufficient resource limits to execute the request, or a lack of a corresponding account or privilege for queueing at a remote queue. In such circumstances, the request will be deleted, and the user will be notified by mail (see *mail*(1)).

The queue type of *network* as alluded to earlier, is implicitly used by *pipe* queues to pass NQS requests between machines, and is also used to handle queued file transfer operations.

QUEUE ACCESS

NQS supports queue access restrictions. For each queue of queue type other than *network*, access may be either *unrestricted* or *restricted*. If access is *unrestricted*, any request may enter the queue. If access is *restricted*, a request can only enter the queue if the requester or the requester's login group has been given access. Requests submitted by root are an exception; they are always queued, even if root has not explicitly been given access.

LIMITS

NQS supports many batch request resource limit types that can be applied to an NQS batch queue. The configurability of these limits allows an NQS manager to set batch queue-specific resource limits which all batch requests in the queue must adhere to.

The syntax of a *limit* in commands of the form **SEt Some_limit** = (*limit*) *queue* is quite flexible.

For *finite* CPU time limits, the acceptable syntax is as follows:

[[hours :] minutes :] seconds [.milliseconds]

Whitespace can appear anywhere between the principal tokens, with the exception that no whitespace can appear around the decimal point.

Example time *limit-values are:*

1234 : 58 : 21.29	– 1234 hrs 58 mins 21.290 secs
12345	– 12345 seconds
121.1	– 121.100 seconds
59:01	– 59 minutes and 1 second

For all other *finite* limits (with the exclusion of the *nice-value*), the acceptable syntax is:

.fraction [units]

or

integer [.fraction] [units]

where the *integer* and *fraction* tokens represent strings of up to eight decimal digits, denoting the obvious values. In both cases, the *units* of allocation may also be specified as one of the case insensitive strings:

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)

In the absence of any *units* specification, the units of *bytes* are assumed.

For all limit types with the exception of the *nice-value*, it is possible to state that no limit should be applied. This is done by specifying a *limit* of "unlimited", or any initial substring thereof.

The complications caused by *batch request* resource limits first show up when queueing a *batch request* in a *batch queue*. This operation is described in the following paragraphs.

If a batch request specifies a limit that cannot be enforced by the underlying UNIX implementation, then the limit is simply ignored, and the batch request will operate as though there were no limit (other than the obvious physical maximums), placed upon that resource type. (See the *qlimit*(1) command to find out what limits are supported by a given machine.)

For each remaining *finite* limit that can be supported by the underlying UNIX implementation that is **not** a CPU *time-limit*, or UNIX *nice-value*, the *limit-value* is internally converted to the units of *bytes* or *words*, whichever is more appropriate for the underlying machine architecture.

As an example, a *per-process memory size limit value* of 321 megabytes would be interpreted as 321 x 2^20 bytes, provided that the underlying machine architecture was capable of directly addressing single bytes. Thus the original limit *coefficient* of 321 would become 321 x 2^20. On a machine that was only capable of addressing words, the appropriate conversion of 321 x 2^20 *bytes / #of-bytes-per-word* would be performed.

If the result of such a conversion would cause overflow when the coefficient was represented as a *signed-long integer* on the supporting hardware, then the coefficient is replaced with the coefficient of: of 2^N-1 where N is equal to the number of bits of precision in a signed long integer. For typical 32-bit machines, this *default extreme limit* would therefore be $2^{31}-1$ bytes. For word addressable machines in the supercomputer class supporting 64-bit long integers, the *default extreme limit* would be $2^{63}-1$ words.

Lastly, some implementations of UNIX reserve coefficients of the form: 2^N-1 as synonomous with infinity, meaning no limit is to be applied. For such UNIX implementations, NQS further decrements the *default extreme limit* so as to not imply infinity.

The identical internal conversion process as described in the preceding paragraphs is also performed for all *finite limit-values* specified with a particular batch request.

After each applicable request *limit* has been converted as described above, the resulting *limit* is then compared against the corresponding *limit* as configured for the destination batch queue. If the corresponding batch queue *limit* for all batch request *limits is defined as unlimited*, or is *greater than* or *equal to* the corresponding batch request *limit*, then the request can be successfully queued, provided that no other anomalous conditions occur. For requests that ask for a *limit* of infinity, the corresponding queue *limit* must also be configured as infinity.

These resource limit checks are performed irrespective of the batch request arrival mechanism, either by a direct use of the *qsub*(1) command, or by the indirect placement of a batch request into a batch queue via a *pipe* queue. It is impossible for a batch request to be queued in an NQS batch queue if *any* of these resource limit checks fail.

Finally, if a request fails to specify a *limit* for a resource limit type that is supported on the execution machine, then the corresponding *limit* as configured for the destination queue becomes the *limit* for the request.

Upon the successful queueing of a request in a batch queue, the set of limits under which the request will execute is frozen, and will not be modified by subsequent *qmgr*(1M) commands that alter the limits of the containing batch queue.

SEE ALSO

qdel(1), qdev(1), qlimit(1), qpr(1), qstat(1), and qsub(1) plus passwd(4), plock(2)
Thinking Machines's Corporation *NQS for the CM-5*

NPSN HISTORY

Origin: Sterling Software Incorporated

August 1985 – Brent Kingsbury, Sterling Software
Original release.

May 1986
Second release.

NAME

qpr – submit a hardcopy print request to NQS

SYNOPSIS

qpr [**–a** date-time] [**–f** form-name] [**–mb**] [**–me**]
[**–mu** user-name] [**–n** number-of-copies] [**–p** priority]
[**–q** queue-name] [**–r** request-name] [**–z**] [files]

DESCRIPTION

Qpr places the named files in a *Network Queueing System* (NQS) queue to be printed by a device such as a line printer or laser printer. If no files are specified, *qpr* will read from the standard input.

In the absence of the **–z** flag, *qpr* will print a *request-id* on the standard output, upon successful queueing of a request. This *request-id* can be compared with what is reported by *qdev*(1) and *qstat*(1) to find out what happened to a request, and given as an argument to *qdel*(1) to delete a request. A *request-id* is always of the form: *seqno.hostname* where *seqno* refers to the sequence number assigned to the request by NQS, and *hostname* refers to the name of originating local machine. This identifier is used throughout NQS to uniquely identify the request, no matter where it is in the network.

The following options to *qpr* may appear in any order and may be intermixed with file names.

–a *date-time*

Submit at the specified date and/or time. In the absence of this flag, *qpr* will submit the request immediately.

If a *date-time* specification is comprised of two or more tokens separated by whitespace characters, then the *date-time* specification must be placed within double quotes as in: **–a** "*July, 4, 2026 12:31-EDT*", or otherwise escaped such that the shell will interpret the entire *date-time* specification as a single lexical token.

The syntax accepted for the *date-time* parameter is relatively flexible. Unspecified date and time values default to an appropriate value (e.g. if no date is specified, then the current month, day, and year are assumed).

A date can be specified as a month and day (current year assumed). The year can also be explicitly specified. It is also possible to specify the date as a weekday name (e.g. "Tues"), or as one of the strings "today" or "tomorrow". Weekday names and month names can be abbreviated by any three character (or longer) prefix of the actual name. An optional period can follow an abbreviated month or day name.

Time of day specifications can be given using a twenty-four hour clock, or "am" and "pm" specifications may be used alternatively. In the absence of a meridian specification, a twenty-four hour clock is assumed.

It should be noted that the time of day specification is interpreted using the precise meridian definitions whereby "12am" refers to the twenty-four hour clock time of 0:00:00, "12m" refers to noon, and "12-pm" refers to 24:00:00. Alternatively, the phrases "midnight" and "noon" are accepted as time of day specifications, where "midnight" refers to the time of 24:00:00.

A timezone may also appear at any point in the *date-time* specification. Thus, it is legal to say: "April 1, 1987 13:01-PDT". In the absence of a timezone specification, the local timezone is assumed, with daylight savings time being inferred when appropriate, based on the date specified.

All alphabetic comparisons are performed in a case insensitive fashion such that both "WeD" and "weD" refer to the day of Wednesday.

Some valid *date-time* examples are:

01-Jan-1986 12am, PDT
Tuesday, 23:00:00

11pm tues.
tomorrow 23-MST

–f *form-name*
Limit the set of acceptable devices to those devices which are loaded with the forms: *form-name*. In the absence of this flag, *qpr* will submit the request only to a device that is loaded with the *default* forms. If there is no *default* forms defined, the request will be submitted to the appropriate output device without regard to the forms configured for the device.

In any case, only those devices associated with the chosen queue will be considered.

–mb Send mail to the user on the originating machine when the request begins execution. If the **–mu** flag is also present, then mail is sent to the user specified for the **–mu** flag instead of to the invoking user.

–me Send mail to the invoker on the originating machine when the request has ended execution. If the **–mu** flag is also present, then mail is sent to the user specified for the **–mu** flag instead of to the invoking user.

–mu *user-name*
Specify that any mail concerning the request should be delivered to the user *user-name*. *User-name* may be formatted either as *user* (containing no '@' characters), or as *user@machine*. In the absence of this flag, any mail concerning the request will be sent to the invoker on the originating machine.

–n *number-of-copies*
Print *number-of-copies* copies. The default is one.

–p *priority* Assign an intra-queue priority to this request. The specified *priority* must be an integer, and must be in the range [0..63], inclusive. A value of 63 defines the highest *intra-queue* request priority, while a value of 0 defines the lowest. This priority does **not** determine the execution priority of the request. This priority is only used to determine the relative ordering of requests within a queue.

When a request is added to a queue, it is placed at a specific position within the queue such that it appears ahead of all existing requests whose priority is less than the priority of the new request. Similarly, all requests with a higher priority will remain ahead of the new request when the queueing process is complete. When the priority of the new request is equal to the priority of an existing request, the existing request takes precedence over the new request.

If no *intra-queue* priority is chosen by the user, then NQS assigns a default value.

–q *queue-name*
Specify the queue to which the device request is to be submitted. If no **–q** *queue-name* specification is given, then the user's environment variable set is searched for the variable: **QPR_QUEUE**. If this environment variable is found, then the character string value for **QPR_QUEUE** is presumed to name the queue to which the request should be submitted. If the **QPR_QUEUE** environment variable is not found, then the request will be submitted to the default device request queue, *if* defined by the local system administrator. Otherwise, the request cannot be queued, and an appropriate error message is displayed to this effect.

–r *request-name*
Assign a name to this request. In the absence of an explict **–r** *request-name* specification, the *request-name* defaults to the name of the first print file (leading path name removed) specified on the command line. If no print files were specified, then the default *request-name* assigned to the request is **STDIN**.

In all cases, if the *request-name* is found to begin with a digit, then the character 'R' is pre-pended to prevent a *request-name* from beginning with a digit. All *request-names* are

truncated to a maximum length of 15 characters.

Be sure not to confuse *request-name* with *request-id*.

-z Submit the request silently. If the request is submitted successfully, nothing will be written to stdout or stderr.

QUEUE ACCESS

NQS supports queue access restrictions. For each queue of queue type other than *network*, access may be either *unrestricted* or *restricted*. If access is *unrestricted*, any request may enter the queue. If access is *restricted*, a request can only enter the queue if the requester or the requester's login group has been given access to that queue (see *qmgr*(1M)). Requests submitted by root are an exception; they are always queued, even if root has not explicitly been given access.

Use *qstat*(1) to determine who has access to a particular queue.

SEE ALSO

mail(1), qdel(1), qdev(1), qlimit(1), qstat(1), qsub(1), qmgr(1M)

NPSN HISTORY

Origin: Sterling Software Incorporated

May 1986 – Robert Sandstrom, Sterling Software
Original release.

NAME

qstat – display status of NQS requests and queues.

SYNOPSIS

qstat [**–a**] [**–U**] [**–b**] [**–d**] [**–p**] [**–f**] [**–l**] [**–n**] [**–s** state -rqht-] [**–h** host-name] [**–u** user-name] [**–T** user-target-name] [queue-name ...] [queue-name@host-name ...]

DESCRIPTION

Qstat displays the status of Network Queueing System (NQS) requests and queues.

If no objects are specified, then the current state of each NQS request on the local host is displayed. Otherwise, information is displayed for the specified object only. Each entry displays information about a given request. Ordinarily, *qstat* shows only those requests belonging to the invoker.

If information about the queues is requested with the **-b**, **-d** or **-p** options, but no queues are specified, then the current state of each NQS queue on the local host is displayed. Otherwise, information is displayed for the specified queues only. Queues may be specified either as *queue-name* or *queue-name@host-name*. In the absence of a *host-name* specifier, the local host is assumed. You must have an account on the host specified in order for *qstat* to work. Also, *root* use of *qstat* is limited to the local machine.

For each selected queue, *qstat* displays information about the queue itself. The following flags are available:

-a Displays all requests. The **-U** (unrestricted) option is synonymous.

-b Displays batch queues.

-d Displays device queues.

-h *host-name*
Displays requests or queues on the specified host.

-p Displays pipe queues.

-f Queues are shown in a full format. The **-l** (long) option displays in the same format.

-n The queue header and trailer are not displayed.

-s *state* Show only those requests in the specified state: *r* (routing), *q* (queued), *h* (held), or *t* (in transition from one queue to another).

-u *user-name*
Shows only those requests belonging to *user-name*.

When a queue is being examined, the queue name, host machine, priority, number of requests in a given state, resource limits, and access are displayed.

QUEUE STATE

The general state of a queue is defined by two principal properties of the queue.

The first property determines whether or not requests can be submitted to the queue. If they can, then the queue is said to be *enabled*. Otherwise the queue is said to be *disabled*.

The second principal property of a queue determines if requests which are ready to run, but are not now presently running, will be allowed to run upon the completion of any currently running requests, and whether any requests are presently running in the queue.

If queued requests not already running are blocked from running, and no requests are presently executing in the queue, then the queue is said to be *stopped*. If the same situation exists with the difference that at least one request is running, then the queue is said to be *stopping*, where the requests presently executing will be allowed to complete execution, but no new requests will be spawned.

If queued requests ready to run are only prevented from doing so by the NQS request scheduler, and one or more requests are presently running in the queue, then the queue is said to be *running*. If the same circumstances prevail with the exception that no requests are presently running in the queue, then the

queue is said to be *inactive*. Finally, if the NQS daemon for the local host upon which the queue resides is not running, but the queue would otherwise be in the state of *running* or *inactive*, then the queue is said to be *shutdown*. The queue states describing the second principal property of a queue are therefore respectively displayed as **STOPPED, STOPPING, RUNNING, INACTIVE,** and **SHUTDOWN.**

REQUEST STATE

The state of a request may be *arriving*, *holding*, *waiting*, *queued*, *staging*, *routing*, *running*, *departing*, or *exiting*. A request is said to be *arriving* if it is being enqueued from a remote host. *Holding* indicates that the request is presently prevented from entering any other state (including the *running* state), because a *hold* has been placed on the request. A request is said to be *waiting* if it was submitted with the constraint that it not run before a certain date and time, and that date and time have not yet arrived. *Queued* requests are eligible to proceed (by *routing* or *running*). When a request reaches the head of a pipe queue and receives service there, it is *routing*. A request is *departing* from the time the pipe queue turns to other work until the request has arrived intact at its destination. *Staging* denotes a *batch* request that has not yet begun execution, but for which input files are being brought on to the execution machine. A *running* request has reached its final destination queue, and is actually executing. Finally, *exiting* describes a batch request that has completed execution, and will exit from the system after the required output files have been returned (to possibly remote machines).

Imagine a batch request originating on a workstation, destined for the batch queue of a computation engine, to be run immediately. That request would first go through the states *queued*, *routing*, and *departing* in a local pipe queue. Then it would disappear from the pipe queue. From the point of view of a queue on the computation engine, the request would first be *arriving*, then *queued*, *staging* (if required by the batch request), *running*, and finally *exiting*. Upon completion of the *exiting* phase of execution, the batch request would disappear from the batch queue.

IDENTIFICATION

CAVEATS

NQS is not finished, and continues to undergo development. Some of the request states shown above may or may not be supported in your version of NQS.

SEE ALSO

qcmplx(1), qdel(1), qdev(1), qlimit(1), qpr(1), qsub(1), qmgr(1M)

NPSN HISTORY

Origin: Sterling Software Incorporated

August 1985 – Brent Kingsbury, Sterling Software
Original release.

May 1986
Second release.

Feb. 1990 – Terrie Carver, Computer Sciences Corporation
Third release.

NAME

qsub – submit an NQS batch request.

SYNOPSIS

qsub [flags] [script-file]

DESCRIPTION

Qsub submits a batch request to the Network Queueing System (NQS).

If no *script-file* is specified, then the set of commands to be executed as a batch request is taken directly from the standard input file (*stdin*). In all cases however, the *script file* is spooled, so that later changes will **not** affect previously queued batch requests.

All of the flags that can be specified on the command line can also be specified within the first comment block inside the batch request *script file* as *embedded default flags*. Such flags appearing in the batch request *script file* set default characteristics for the batch request. If the same flag is specified on the command line, then the command line flag (and any associated value) takes precedence over the *embedded* flag. See the section entitled: **LONG DESCRIPTION** for more information on *embedded default flags*.

What follows is a terse definition of the flags implemented by the *Qsub* command (see the section: **LONG DESCRIPTION** for the complete definition and syntax used for each of these flags).

–a – run request after stated time
–e – direct stderr output to stated destination
–eo – direct stderr output to the stdout destination
–ke – keep stderr output on the execution machine
–ko – keep stdout output on the execution machine
–lc – establish per-process corefile size limit
–ld – establish per-process data-segment size limits
–lf – establish per-process permanent-file size limits
–lF – establish per-request permanent-file space limits
–lm – establish per-process memory size limits
–lM – establish per-request memory space limits
–ln – establish per-process nice execution value limit
–ls – establish per-process stack-segment size limits
–lt – establish per-process CPU time limits
–lT – establish per-request CPU time limits
–lv – establish per-process temporary-file size limits
–lV – establish per-request temporary-file space limits
–lw – establish per-process working set limit
–mb – send mail when the request begins execution
–me – send mail when the request ends execution
–mr – send mail when the request restarts
–mt – send mail when the request is being transported between queues
–mu – send mail for the request to the stated user
–nr – declare that batch request is not restartable
–o – direct stdout output to the stated destination
–p – specify intra-queue request priority
–q – queue request in the stated queue
–r – assign stated request name to the request
–re – remotely access the stderr output file
–ro – remotely access the stdout output file
–s – specify shell to interpret the batch request script
–x – export all environment variables with request
–z – submit the request silently

LONG DESCRIPTION

As described above, it is possible to specify *default* flags within the batch request *script file* that configure the default behavior of the batch request. The algorithm used to scan for such *embedded default flags* within an NQS batch request script file is as follows:

1. Read the first line of the *script file*.
2. If the current line contains only whitespace characters, or the first non-whitespace character of the line is ":", then goto step 7.
3. If the first non-whitespace character of the current line is not a "#" character, then goto step 8.
4. If the second non-whitespace character in the current line is *not* the "@" character, or the character immediately following the second non-whitespace character in the current line is *not* a "$"

   OR

   If the second non-whitespace character is not a "Q" followed immediately by the string "SUB", then goto step 7.
5. If no "-" is present as the first non-whitespace character *immediately* following the "@$" sequence or the "QSUB" sequence, then goto step 8.
6. Process the *embedded* flag, stopping the parsing process upon reaching the end of the line, or upon reaching the first unquoted "#" character.
7. Read the next *script file* line. Goto step 2.
8. End. No more *embedded* flags will be recognized.

Here is an example of the use of *embedded* flags within the *script file*.

```
#
# Batch request script example:
#
# @$-a "11:30pm EDT" -lt "21:10, 20:00"
#           # Run request after 11:30 EDT by default,
#           # and set a maximum per-process CPU time
#           # limit of 21 minutes and ten seconds.
#           # Send a warning signal when any process
#           # of the running batch request consumes
#           # more than 20 minutes of CPU time.
# QSUB -lT 1:45:00
#           # Set a maximum per-request CPU time limit
#           # of one hour, and 45 minutes. (The
#           # implementation of CPU time limits is
#           # completely dependent upon the UNIX
#           # implementation at the execution
#           # machine.)
# QSUB-mb -me # Send mail at beginning and end of
#           # request execution.
# @$-q batch1 # Queue request to queue: batch1 by
#           # default.
# @$          # No more embedded flags.
#
make all
```

The following paragraphs give the detailed descriptions of the *flags* supported by the *Qsub* command.

–a *date-time* Do not run the batch request before the specified date and/or time. If a *date-time* specification is comprised of two or more tokens separated by whitespace characters, then the *date-time* specification must be placed within double quotes as in: **–a** *"July, 4, 2026 12:31-EDT"*, or otherwise escaped such that *Qsub* and the shell will interpret the entire *date-time* specification as a single character string. This restriction also applies when an embedded default **–a** flag with accompanying *date-time* specification appears within the batch request *script file*.

The syntax accepted for the *date-time* parameter is relatively flexible. Unspecified date and time values default to an appropriate value (e.g. if no date is specified, then the current month, day, and year are assumed).

A date may be specified as a month and day (current year assumed), or the year can also be explicitly specified. It is also possible to specify the date as a weekday name (e.g. "Tues"), or as one of the strings: "today", or "tomorrow". Weekday names and month names can be abbreviated by any three character (or longer) prefix of the actual name. An optional period can follow an abbreviated month or day name.

Time of day specifications can be given using a twenty-four hour clock, or "am" and "pm" specifications may be used alternatively. In the absence of a meridian specification, a twenty-four hour clock is assumed.

It should be noted that the time of day specification is interpreted using the precise meridian definitions whereby "12am" refers to the twenty-four hour clock time of 0:00:00, "12m" refers to noon, and "12-pm" refers to 24:00:00. Alternatively, the phrases "midnight" and "noon" are accepted as time of day specifications, where "midnight" refers to the time of 24:00:00.

A timezone may also appear at any point in the *date-time* specification. Thus, it is legal to say: "April 1, 1987 13:01-PDT". In the absence of a timezone specification, the local timezone is assumed, with daylight savings time being inferred when appropriate, based on the date specified.

All alphabetic comparisons are performed in a case insensitive fashion such that both "WeD" and "weD" refer to the day of Wednesday.

Some valid *date-time* examples are:

01-Jan-1986 12am, PDT
Tuesday, 23:00:00
11pm tues.
tomorrow 23-MST

–e *[machine:][[/]path/] stderr-filename*

Direct output generated by the batch request which is sent to the *stderr* file to the named *[machine:][[/]path/] stderr-filename*.

The brackets "[" and "]" enclose optional portions of the *stderr* destination *machine*, *path*, and *stderr-filename*.

If no explicit *machine* destination is specified, then the destination machine defaults to the machine that originated the batch request, or to the machine that will eventually run the request, depending on the respective absence, or presence of the **–ke** flag.

If no *machine* destination is specified, and the path/filename does not begin with a "/", then the current working directory is prepended to create a fully qualified path name, provided that the **–ke** (keep stderr) flag is also absent. In all other cases, any partial path/filename is interpreted relative to the user's home directory on the *stderr* destination machine.

This flag cannot be specified when the **–eo** flag option is also present.

If the **–eo** and **–e** *[machine:][[/]path/] stderr-filename* flag options are not present, then all *stderr* output for the batch request is sent to the file whose name consists of the first seven characters of the *request-name* followed by the characters: ".e", followed by the request sequence number portion of the *request-id* discussed below. In the absence of the **–ke** flag, this default *stderr* output file will be placed on the machine that originated the batch request in the current working directory, as defined when the batch request was first submitted. Otherwise, the file will be placed in the user's home directory on the execution machine.

–eo Direct all output that would normally be sent to the *stderr* file to the *stdout* file for the batch request. This flag cannot be specified when the **–e** *[machine:][[/]path/] stderr-filename* flag option is also present.

–ke In the absence of an explicit *machine* destination for the *stderr* file produced by a batch request, the *machine* destination chosen for the *stderr* output file is the machine that originated the batch request. In some cases however, this behavior may be undesirable, and so the **–ke** flag can be specified which instructs NQS to leave any *stderr* output file produced by the request on the machine that actually *executed* the batch request.

This flag is meaningless if the **–eo** flag is specified, and cannot be specified if an explicit *machine* destination is given for the *stderr* parameter of the **–e** flag.

–ko In the absence of an explicit *machine* destination for the *stdout* file produced by a batch request, the *machine* destination chosen for the *stdout* output file is the machine that originated the batch request. In some cases however, this behavior may be undesirable, and so the **–ko** flag can be specified which instructs NQS to leave any *stdout* output file produced by the request on the machine that actually *executed* the batch request.

This flag cannot be specified if an explicit *machine* destination is given for the *stdout* parameter of the **–o** flag.

–lc *per-process corefile size limit*

Set a *per-process* maximum *core file size limit* for all processes that constitute the running batch request. If any process comprising the running request attempts to exit creating a core file whose size would exceed the maximum *per-process core file size limit* for the request, then the core file image of the aborting process will be reduced to the necessary size by an algorithm dependent upon the underlying UNIX implementation.

Not all UNIX implementations support *per-process corefile size limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-process corefile size limit*.

–ld *per-process data-segment size limit [, warn-limit]*

Set a *per-process* maximum and an optional warning *data-segment size limit* for all processes that constitute the running batch request. If any process comprising the running request exceeds the maximum *per-process data-segment size-limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process data-segment warning size limits*. When a warning limit is exceeded, a signal as determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is comprised of two or

more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **–ld** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-process data-segment size limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-process data-segment size limit*.

–lf *per-process permanent-file size limit [, warn-limit]*

Set a *per-process* maximum and an optional warning *permanent-file size limit* for all processes that constitute the running batch request. If any process comprising the running request attempts to write to a permanent file such that the file size would increase beyond the maximum *per-process permanent-file size limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process warning permanent-file size limits*. When a warning limit is exceeded, a signal as determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **–lf** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-process permanent-file size limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

At the time of this writing, the author was unaware of any UNIX implementation that made a distinction at the **kernel** level, between *permanent*, and *temporary* files. While it is certainly possible to construct a *pseudo-temporary* file by first creating it, and then unlinking its pathname, the disk space allocated for such a file will be allocated from the same pool of disk space that all other UNIX files are allocated from. Furthermore, such a file will be subject to the same quota enforcement mechanisms, if any are provided by the underlying UNIX implementation, that all other UNIX files are created under.

For all UNIX implementations that do not support a distinction between *permanent*, and *temporary* files at the **kernel** level, this limit is interpreted as a *per-process file size limit*, with the word *permanent* removed from the definition.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-process permanent-file size limit*.

–lF *per-request permanent-file space limit [, warn-limit]*

Set a *per-request* maximum and an optional warning cumulative *permanent-file space limit* for all processes that constitute the running batch request. (Not available for requests running on the CM-5.) If any process comprising the running request attempts to write to a permanent file such that the file space consumed by all permanent files opened for writing by all of the processes in the batch request, would increase beyond the maximum *per-request permanent-file space limit* for the request, then all of the processes in

the request will be terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-request warning permanent-file space limits*. When such a warning limit is exceeded, a signal is delivered to one or more of the processes comprising the running request, depending upon the underlying UNIX implementation.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **-lF** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-request permanent-file space limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

At the time of this writing, the author was unaware of any UNIX implementation that made a distinction at the **kernel** level, between *permanent*, and *temporary* files. While it is certainly possible to construct a *pseudo-temporary* file by first creating it, and then unlinking its pathname, the disk space allocated for such a file will be allocated from the same pool of disk space that all other UNIX files are allocated from. Furthermore, such a file will be subject to the same quota enforcement mechanisms, if any are provided by the underlying UNIX implementation, that all other UNIX files are created under.

For all UNIX implementations that do not support a distinction between *permanent*, and *temporary* files at the **kernel** level, this limit is interpreted as a *per-request file space limit*, with the word *permanent* removed from the definition.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-request permanent-file space limit*.

-lm *per-process memory size limit [, warn-limit]*

Set a *per-process* maximum and an optional warning *memory size limit* for all processes that constitute the running batch request. (Not available for the CM-5.) If any process comprising the running request exceeds the maximum *per-process memory size limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process warning memory size limits*. When a warning limit is exceeded, a signal as determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **-lm** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-process memory size limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-process memory size*

limit.

–lM *per-request memory space limit [, warn-limit]*

Set a *per-request* maximum and an optional warning cumulative *memory space limit* for all processes that constitute the running batch request. (Not available for requests running on the CM-5.) If the sum of all memory consumed by all of the processes comprising the running request exceeds the maximum *per-request memory space limit* for the request, then all of the processes in the request will be terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-request warning memory size limits*. When such a warning limit is exceeded, a signal is delivered to one or more of the processes comprising the running request, depending upon the underlying UNIX implementation.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **–lM** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-request memory space limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-request memory space limit*.

–ln *per-process nice value limit*

Set a *per-process nice value* for all processes comprising the running batch request.

At present, all UNIX implementations support the use of an integer called the *nice* value, which determines the *execution-time* priority of a process relative to all other processes in the system. By letting the user set a limit on the *nice* value for all processes comprising the running request, a user can cause a request to consume less (or more) of the CPU resource presented by the execution machine.

This is particularly useful when a user wishes to execute a CPU intensive batch request on a machine running interactive processes. By setting a low *execution-time priority*, a user can make a long running batch request give way to more interactive processes during the daytime, while the coming of the nighttime hours with typically smaller process loads will allow such a request to gain more and more of the CPU resource. In this way, long running batch requests can be polite to their more transient, interactive neighbor processes.

The only quirk associated with this flag results from the peculiar choice of *nice* values, implemented by the standard UNIX implementations. In general, increasingly *negative* nice values cause the relative execution priority of a process to *increase*, while increasingly *positive* nice values causes the relative priority to *decrease*! Thus, a *nice value* limit specification of: "-ln -10" is greedier than a *nice value* limit specification of: "-ln 0".

Since varying UNIX implementations often support a different finite range of *nice values*, NQS allows the specification of *nice values* that can eventually turn out to be outside the limits for the UNIX implementation running at the *execution* machine. In such cases, NQS will simply bind the specified *nice value* limit to within the necessary range as appropriate.

Lastly, any *nice value* specified by the use of this flag must be acceptable to the batch queue in which the request is ultimately placed (see the section entitled **LIMITS** for more information).

–ls *per-process stack-segment size limit [, warn-limit]*

Set a *per-process* maximum and an optional warning *stack-segment size limit* for all processes that constitute the running batch request. If any process comprising the running request exceeds the maximum *per-process stack-segment size limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process warning stack-segment size limits*. When a warning limit is exceeded, a signal as determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **–ls** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-process stack-segment size limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-process stack-segment size limit*.

–lt *per-process CPU time limit [, warn-limit]*

Set a *per-process* maximum and an optional warning *CPU time limit* for all processes that constitute the running batch request. If any process comprising the running request exceeds the maximum *per-process CPU time limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process CPU warning time limits*. When a warning limit is exceeded, a signal as determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **–lt** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-process CPU time limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-process CPU time limit*.

–lT *per-request CPU time limit [, warn-limit]*

Set a *per-request* maximum and an optional warning cumulative *CPU time limit* for all of the processes that constitute the running batch request. (Not available for requests running on the CM-5.) If the sum of the CPU times consumed by all of the processes in the

batch request exceeds the maximum *per-request CPU time limit* for the request, then all of the processes in the request will be terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-request CPU warning time limits*. When such a warning limit is exceeded, a signal is delivered to one or more of the processes comprising the running request, depending upon the underlying UNIX implementation.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **–lT** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-request CPU time limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-request CPU time limit*.

–lv *per-process temporary file size limit [, warn-limit]*

Set a *per-process* maximum and an optional warning *temporary (volatile) file size limit* for all processes that constitute the running batch request. (Not available for requests running on the CM-5.) If any process comprising the running request attempts to write to a *temporary* file such that the file size would increase beyond the maximum *per-process temporary-file size limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process warning temporary-file size limits*. When a warning limit is exceeded, a signal as determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **–lv** flag with its associated limit value(s) appears within the batch request *script file*.

At the time of this writing, no UNIX operating system known to the author supported a distinction at the **kernel** level between *permanent* and *temporary files*. Certainly, a *pseudo-temporary* file can be constructed by creating it, and then unlinking its pathname. However, the file space allocated for such a file will be allocated from the same pool of disk space that all other UNIX files are allocated from.

Until a mechanism is implemented in the **kernel** that knows about *permanent* and *temporary* files, this limit cannot be supported in the sense most useful for batch requests, namely the strict enforcement of disk quotas for *permanent* versus *temporary* files.

Until such a time, this limit will simply be ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-process temporary-file size limit*.

–lV *per-request temporary file space limit [, warn-limit]*

Set a *per-request* maximum and an optional warning cumulative *temporary (volatile) file*

space limit for all processes that constitute the running batch request. (Not available for requests running on the CM-5.) If any process comprising the running request attempts to write to a *temporary* file such that the file space consumed by all *temporary* files opened for writing by all of the processes in the batch request would increase beyond the maximum *per-request temporary-file space limit* for the request, then all of the processes in the request will be terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-request warning temporary-file space limits*. When such a warning limit is exceeded, a signal is delivered to one or more of the processes comprising the running request, depending upon the underlying UNIX implementation.

If a maximum limit (and optional warning limit) specification is comprised of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes, or otherwise escaped such that *Qsub* and the shell will interpret the entire specification as a single character string token. This caveat also applies when an embedded default **-lV** flag with its associated limit value(s) appears within the batch request *script file*.

At the time of this writing, no UNIX operating system known to the author supported a distinction at the **kernel** level between *permanent* and *temporary files*. Certainly, a *pseudo-temporary* file can be constructed by creating it, and then unlinking its pathname. However, the file space allocated for such a file will be allocated from the same pool of disk space that all other UNIX files are allocated from.

Until a mechanism is implemented in the **kernel** that knows about *permanent* and *temporary* files, this limit cannot be supported in the sense most useful for batch requests, namely the strict enforcement of disk quotas for *permanent* versus *temporary* files.

Until such a time, this limit will simply be ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *temporary-file space limit*.

-lw *per-process working set size limit*

Set a *per-process* maximum *working set size limit* for all processes that constitute the running batch request.

Not all UNIX implementations support *per-process working set size limits*, and such a limit only makes sense in the context of a paged virtual memory machine. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

See the section entitled **LIMITS** for more information on the implementation of batch request limits, and for a description of the precise syntax of a *per-process working set size limit*.

-mb Send mail to the user on the originating machine when the request begins execution. If the **-mu** flag is also present, then mail is sent to the user specified for the **-mu** flag instead of to the invoking user.

-me Send mail to the user on the originating machine when the request has ended execution. If the **-mu** flag is also present, then mail is sent to the user specified for the flag instead of to the invoking user.

-mr Send mail to the user on the originating machine when the request restarts. If the **-mu** flag is also present, then mail is sent to the user specified for the **-mu** flag instead of to the invoking user.

-mt Send mail to the user on the originating machine when the request is being transported between queues. If the **-mu** flag is also present, then mail is sent to the user specified for

the **-mu** flag instead of to the invoking user.

-mu *user-name*

Specify that any mail concerning the request should be delivered to the user *user-name*. *User-name* may be formatted either as *user* (containing no '@' characters), or as *user@machine*. In the absence of this flag, any mail concerning the request will be sent to the invoker on the originating machine.

-nr Declare that the request is non-restartable. If this flag is specified, then the request will not be restarted by NQS upon system boot if the request was running at the time of an NQS shutdown or system crash.

By default, NQS assumes that all requests are restartable, with the caveat that it is the responsibility of the user to ensure that the request will execute correctly if restarted, by the use of appropriate programming techniques.

Requests that are not running are always preserved across host crashes and NQS shutdowns for later requeueing, with or without this flag.

When NQS is shutdown via an operator command to the *qmgr*(1M) NQS control program, a **SIGTERM** signal is sent to all processes comprising all running NQS requests on the local host, and all queued NQS requests are barred from beginning execution. After a finite number of seconds have elapsed (typically sixty, but this value can be overridden by the operator), all remaining processes comprising all remaining running NQS requests are killed by the signal: **SIGKILL**.

For an NQS request to be properly restarted after an NQS shutdown, the **-nr** flag must not be specified, and the spawned batch request shell must ignore **SIGTERM** signals (which is done by default). The spawned batch request shell must also not exit before the final **SIGKILL** arrives. Since the batch request shell is simply spawning commands and programs, waiting for their completion, this implies that the commands and programs being executed by the batch request shell must also be immune to **SIGTERM** signals, saving state as appropriate before being killed by the final **SIGKILL** signal.

See the **CAVEATS** section below for more discussion concerning the restartability of NQS batch requests.

-o *[machine:][[/]path/] stdout-filename*

Direct output generated by the batch request which is sent to the *stdout* file to the named *[machine:][[/]path/] stdout-filename*.

The brackets "[" and "]" enclose optional portions of the *stdout* destination *machine*, *path*, and *stdout-filename*.

If no explicit *machine* destination is specified, then the destination machine defaults to the machine that originated the batch request, or to the machine that will eventually run the request, depending on the respective absence, or presence of the **-ko** flag.

If no *machine* destination is specified, and the path/filename does not begin with a "/", then the current working directory is prepended to create a fully qualified path name, provided that the **-ko** (keep stdout) flag is also absent. In all other cases, any partial path/filename is interpreted relative to the user's home directory on the *stdout* destination machine.

If no **-o** *[machine:][[/]path/] stdout-filename* flag is specified, then all *stdout* output for the batch request is sent to the file whose name consists of the first seven characters of the *request-name* followed by the characters: ".o", followed by the request sequence number portion of the *request-id* discussed below. In the absence of the **-ko** flag, this default *stdout* output file will be placed on the machine that originated the batch request in the current working directory, as defined when the batch request was first submitted. Otherwise, the file will be placed in the user's home directory on the execution machine.

–p *priority* Explicitly assign an *intra-queue* priority to the request. The specified *priority* must be an integer, and must be in the range [0..63], inclusive. A value of 63 defines the highest *intra-queue* request priority, while a value of 0 defines the lowest. This priority does **not** determine the execution priority of the request. This priority is only used to determine the relative ordering of requests within a queue.

When a request is added to a queue, it is placed at a specific position within the queue such that it appears ahead of all existing requests whose priority is less than the priority of the new request. Similarly, all requests with a higher priority will remain ahead of the new request when the queueing process is complete. When the priority of the new request is equal to the priority of an existing request, the existing request takes precedence over the new request.

If no *intra-queue* priority is chosen by the user, then NQS assigns a default value.

–q *queue-name*

Specify the queue to which the batch request is to be submitted. If no **–q** *queue-name* specification is given, then the user's environment variable set is searched for the variable: **QSUB_QUEUE**. If this environment variable is found, then the character string value for **QSUB_QUEUE** is presumed to name the queue to which the request should be submitted. If the **QSUB_QUEUE** environment variable is not found, then the request will be submitted to the default batch request queue, *if* defined by the local system administrator. Otherwise, the request cannot be queued, and an appropriate error message is displayed to this effect.

–r *request-name*

Assign the specified *request-name* to the request. In the absence of an explict **–r** *request-name* specification, the *request-name* defaults to the name of the *script file* (leading path name removed) given on the command line. If no *script file* was given, then the default *request-name* assigned to the request is **STDIN**.

In all cases, if the *request-name* is found to begin with a digit, then the character 'R' is prepended to prevent a *request-name* from beginning with a digit. All *request-names* are truncated to a maximum length of 15 characters.

–re By default, all output generated by a batch request sent to the *stderr* file is temporarily into a file residing in a protected directory on the machine that executes the request. When the batch request completes execution, this file is then spooled to its final destination, possibly on a remote machine.

This default spooling of the *stderr* output file is done to reduce the network traffic costs incurred by the submitter (owner) of a batch request which is to return its *stderr* output to a remote machine upon completion. In some cases, this behavior is not desired. If it is necessary to override this behavior, then the **–re** flag can be specified which says that *stderr* output produced by the request is to be *immediately* written to the final destination file, as output is generated, no matter what the networking cost.

Circumstances may not allow a given NQS implementation to support this flag, in which case it will be ignored, and the *stderr* output file will simply be spooled as it ordinarily would without this flag.

–ro By default, all output generated by a batch request sent to the *stdout* file is temporarily spooled into a file residing in a protected directory on the machine that executes the request. When the batch request completes execution, this file is then spooled to its final destination, possibly on a remote machine.

This default spooling of the *stdout* output file is done to reduce the network traffic costs incurred by the submitter (owner) of a batch request which is to return its *stdout* output to a remote machine upon completion. In some cases, this behavior is not desired. If it is necessary to override this behavior, then the **–ro** flag can be specified which says that

stdout output produced by the request is to be *immediately* written to the final destination file, as output is generated, no matter what the networking cost.

Circumstances may not allow a given NQS implementation to support this flag, in which case it will be ignored, and the *stdout* output file will simply be spooled as it ordinarily would without this flag.

-s *shell-name*

Specify the absolute path name of the shell which will be used to interpret the batch request script. This flag unconditionally overrides any *shell strategy* configured on the execution machine for selecting which shell to spawn in order to interpret the batch request script.

In the absence of this flag, the NQS system at the execution machine will use one of three (3) distinct *shell choice strategies* for the execution of the batch request. Any one of the three strategies can be configured by a system administrator for each NQS machine.

The three shell strategies are called:

fixed,
free, and
login.

These shell strategies respectively cause the configured *fixed* shell to be exec'd to interpret all batch requests, cause the user's login shell as defined in the password file to be exec'd which in turn chooses and spawns the appropriate shell for interpreting the batch request script, or cause only the user's login shell to be exec'd to interpret the script.

A shell strategy of *fixed* means that the same shell (as configured by the system administrator), will be used to execute **all** batch requests.

A shell strategy of *free* will run the batch request script *exactly* as would an interactive invocation of the script, and is the default NQS shell strategy.

The strategies of *fixed* and *login* exist for host systems that are short on available free processes. In these two strategies, a single shell is exec'd, and that same shell is the shell that executes all of the commands in the batch request script.

The *shell strategy* configured for a particular NQS system can be determined by the *qlimit*(1) command.

-x Export all environment variables. When a batch request is submitted, the current values of the environment variables: **HOME**, **SHELL**, **PATH**, **LOGNAME** (not all systems), **USER** (not all systems), **MAIL**, and **TZ** are saved for later recreation when the batch request is spawned, as the respective environment variables: **QSUB_HOME**, **QSUB_SHELL**, **QSUB_PATH**, **QSUB_LOGNAME**, **QSUB_USER**, **QSUB_MAIL**, and **QSUB_TZ**. Unless the **-x** flag is specified, no other environment variables will be exported from the originating host for the batch request. If the **-x** flag option is specified, then all remaining environment variables whose names do not conflict with the automatically exported variables, are also exported with the request. These additional environment variables will be recreated under the same name when the batch request is spawned.

-z Submit the batch request silently. If the request is submitted successfully, then no messages are displayed indicating this fact. Error messages will, however, always be displayed.

If the batch request is successfully submitted and the **-z** flag has not been specified, the *request-id* of the request is displayed to the user. A *request-id* is always of the form: *seqno.hostname* where *seqno* refers to the sequence number assigned to the request by NQS, and *hostname* refers to the name of originating local machine. This identifier is used throughout NQS to uniquely identify the request, no

matter where it is in the network.

The following events take place in the following order when an NQS *batch* request is spawned:

The process that will become the head of the *process group* for all processes comprising the batch request is created by NQS.

Resource limits are enforced.

The real and effective group-id of the process is set to the group-id as defined in the local password file for the request owner.

The real and effective user-id of the process is set to the real user-id of the batch request owner.

The user file creation mask is set to the value that the user had on the originating machine when the batch request was first submitted.

It the user explicitly specified a shell by use of the **–s** flag (discussed above), then that user-specified shell is chosen as the shell that will be used to execute the batch request script. Otherwise, a shell is chosen based upon the *shell strategy* as configured for the local NQS system (see the earlier discussion of the **–s** flag for a description of the possible *shell strategies* that can be configured for an NQS system).

The environment variables of **HOME, SHELL, PATH, LOGNAME** (not all systems), **USER** (not all systems), and **MAIL** are set from the user's password file entry, as though the user had logged directly into the execution machine.

The environment string: **ENVIRONMENT=BATCH** is added to the environment so that shell scripts (and the user's **.profile** (*Bourne shell*) or **.cshrc** and **.login** (*C-shell*) scripts), can test for batch request execution when appropriate, and not (for example) perform any setting of terminal characteristics, since a batch request is not connected to an input terminal.

The environment variables of **QSUB_WORKDIR, QSUB_HOST, QSUB_REQNAME,** and **QSUB_REQID** are added to the environment. These environment variables equate to the obvious respective strings of the working directory at the time that the request was submitted, the name of the originating host, the name of the request, and the request *request-id*.

All of the remaining environment variables saved for recreation when the batch request is spawned are added at this point to the environment. When a batch request is initially submitted, the current values of the environment variables: **HOME, SHELL, PATH, LOGNAME** (not all systems), **USER** (not all systems), **MAIL,** and **TZ** are saved for later recreation when the batch request is spawned. When recreated however, these variables are added to the environment under the respective names: **QSUB_HOME, QSUB_SHELL, QSUB_PATH, QSUB_LOGNAME, QSUB_USER, QSUB_MAIL,** and **QSUB_TZ,** to avoid the obvious conflict with the local version of these environment variables. Additionally, all environment variables exported from the originating host by the **–x** option are added to the environment at this time.

The current working directory is then set to the user's home directory on the execution machine, and the chosen shell is exec'd to execute the batch request script with the environment as constructed in the steps outlined above.

In all cases, the chosen shell is exec'd as though it were the *login* shell. If the *Bourne* shell is chosen to execute the script, then the **.profile** file is read. If the *C-shell* is chosen, then the **.cshrc** and **.login** scripts are read.

If the user did not specify a specific shell for the batch request, then NQS chooses which shell should be used to execute the shell script, based on the *shell strategy* as configured by the system administrator (see the earlier discussion of the **–s** flag).

In such a case, a *free* shell strategy instructs NQS to execute the login shell for the user (as configured in the password file). The login shell is in turn instructed to examine the shell script file, and fork another shell *of the appropriate type* to interpret the shell script, behaving *exactly* as an interactive invocation of the script.

Otherwise no additional shell is spawned, and the chosen *fixed* or *login* shell sequentially executes the commands contained in the shell script file until completion of the batch request.

QUEUE TYPES

NQS supports four different queue types that serve to provide four very different functions. These four queue types are known as *batch*, *device*, *pipe*, and *network*.

The queue type of *batch* can only be used to execute NQS *batch requests*. Only NQS *batch requests* created by the *qsub*(1) command can be placed in a *batch queue*.

The queue type of *device* can only be used to execute NQS *device requests*. Only NQS *device requests* created by the *qpr*(1) command can be placed in a *device queue*.

Queues of type *pipe* are used to send NQS requests to other *pipe* queues, or to request destination queues of type *batch* or *device*, as appropriate for the request type. In general, *pipe queues*, in combination with *network queues*, act as the mechanism that NQS uses to transport both *batch* and *device* requests to distant queues on other remote machines. It is also perfectly legal for a *pipe queue* to transport requests to queues on the *same* machine.

When a *pipe queue* is defined, it is given a *destination set* which defines the set of possible destination queues for requests entered in that *pipe queue*. In this manner, it is possible for a *batch* or *device* request to pass through many pipe queues on its way to its ultimate destination, which must eventually be a queue of type *batch* or *device* (matching the request type).

Each *pipe queue* has an associated *server*. For each request handled by a *pipe queue*, the associated server is spawned which must select a queue destination for the request being handled, based on the characteristics of the request, and upon the characteristics of each queue in the *destination set* defined for the pipe queue.

Since a different server can be configured for each pipe queue, and *batch* and *device* queues can be endowed with the *pipeonly* attribute that will only admit requests queued via another *pipe queue*, it is possible for respective NQS installations to use *pipe queues* as a *request class* mechanism, placing requests that ask for different resource allocations in different queues, each of which can have different associated limits and priorities.

It is also completely possible for a *pipe queue server*, when handling a request, to discover that no *destination queue* will accept the request, for various reasons which can include insufficient resource limits to execute the request, or a lack of a corresponding account or privilege for queueing at a remote queue. In such circumstances, the request will be deleted, and the user will be notified by mail (see *mail*(1)).

The queue type of *network*, as alluded to earlier, is implicitly used by *pipe* queues to pass NQS requests between machines, and is also used to handle queued file transfer operations.

QUEUE ACCESS

NQS supports queue access restrictions. For each queue of queue type other than *network*, access may be either *unrestricted* or *restricted*. If access is *unrestricted*, any request may enter the queue. If access is *restricted*, a request can only enter the queue if the requester or the requester's login group has been given access to that queue (see *qmgr*(1M)). Requests submitted by root are an exception; they are always queued, even if root has not explicitly been given access.

Use *qstat*(1) to determine who has access to a particular queue.

LIMITS

NQS supports many batch request resource limit types that can be applied to an NQS batch request. The existence of configurable resource limits allows an NQS user to set resource limits within which his or her request must execute. In many instances, smaller limit values can result in a more favorable

scheduling policy for a batch request.

The syntax used to specify a *limit-value* for one of the *limit-flags* (*–llimit-letter-type*), is quite flexible, and describes values for two general limit categories. These two general categories respectively deal with time related limits, and those limits are not time related.

For *finite* CPU time limits, the *limit-value* is expressed in the reasonably obvious format:

[[hours :] minutes :] seconds [.milliseconds]

Whitespace can appear anywhere between the principal tokens, with the exception that no whitespace can appear around the decimal point.

Example time *limit-values are:*

1234 : 58 : 21.29–	1234 hrs 58 mins 21.290 secs
12345	– 12345 seconds
121.1	– 121.100 seconds
59:01	– 59 minutes and 1 second

For all other *finite* limits (with the exclusion of the *nice limit-value* *–ln*), the acceptable syntax is:

.fraction [units]

or

integer [.fraction] [units]

where the *integer* and *fraction* tokens represent strings of up to eight decimal digits, denoting the obvious values. In both cases, the *units* of allocation may also be specified as one of the case insensitive strings:

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)

In the absence of any *units* specification, the units of *bytes* are assumed.

For all limit types with the exception of the *nice limit-value* (*–ln*), it is possible to state that no limit should be applied. This is done by specifying a *limit-value* of "unlimited", or any initial substring thereof. Whenever an *infinite limit-value* is specified for a particular resource type, then the batch request operates as though no explicit limits have been placed upon the corresponding resource, other than by the limitations of the physical hardware involved.

The complications caused by *batch request* resource limits first show up when queueing a *batch request* in a *batch queue*. This operation is described in the following paragraphs.

If a batch request specifies a limit that cannot be enforced by the underlying UNIX implementation, then the limit is simply ignored, and the batch request will operate as though there were no limit (other than the obvious physical maximums), placed upon that resource type. (See the *qlimit*(1) command to find out what limits are supported by a given machine.)

For each remaining *finite* limit that can be supported by the underlying UNIX implementation that is **not** a CPU *time-limit* or UNIX *execution-time nice-value-limit*, the *limit-value* is internally converted to the units of *bytes* or *words*, whichever is more appropriate for the underlying machine architecture.

As an example, a *per-process memory size limit value* of 321 megabytes would be interpreted as 321 x 2^20 bytes, provided that the underlying machine architecture was capable of directly addressing single bytes. Thus the original limit *coefficient* of 321 would become 321 x 2^20. On a machine that was only capable of addressing words, the appropriate conversion of 321 x 2^20 *bytes / #of-bytes-per-word* would be performed.

If the result of such a conversion would cause overflow when the coefficient was represented as a *signed-long integer* on the supporting hardware, then the coefficient is replaced with the coefficient of: of 2^*N*-1 where *N* is equal to the number of bits of precision in a signed long integer. For typical 32-bit machines, this *default extreme limit* would therefore be 2^*31*-1 bytes. For word addressable machines in the supercomputer class supporting 64-bit long integers, the *default extreme limit* would be 2^*63*-1 words.

Lastly, some implementations of UNIX reserve coefficients of the form: 2^*N*-1 as synonymous with infinity, meaning no limit is to be applied. For such UNIX implementations, NQS further decrements the *default extreme limit* so as not to imply infinity.

The identical internal conversion process as described in the preceding paragraphs is also performed for each *finite limit-value* configured for a particular batch queue using the *qmgr*(1M) program.

After all of the applicable *limit-values* have been converted as described above, each such resulting *limit-value* is then compared against the corresponding *limit-value* as configured for the destination batch queue. If, for every type of limit, the batch queue *limit-value* is *greater than* or *equal to* the corresponding batch request *limit-value*, then the request can be successfully queued, provided that no other anomalous conditions occur. For request *infinity limit-values*, the corresponding queue *limit-value* must also be configured as infinity.

These resource limit checks are performed irrespective of the batch request arrival mechanism, either by a direct use of the *qsub(1)* command, or by the indirect placement of a batch request into a batch queue via a *pipe* queue. It is impossible for a batch request to be queued in an NQS batch queue if *any* of these resource limit checks fail.

Finally, if a request fails to specify a *limit-value* for a resource limit type that is supported on the execution machine, then the corresponding *limit-value* configured for the destination queue becomes the *limit-value* for the unspecified request limit.

Upon the successful queueing of a request in a batch queue, the set of limits under which the request will execute is frozen, and will not be modified by subsequent *qmgr*(1M) commands that alter the limits of the containing batch queue.

CAVEATS

When an NQS batch request is spawned, a new *process-group* is established such that all processes of the request exist in the same *process-group*. If the *qdel*(1) command is used to send a signal to an NQS batch request, the signal is sent to all processes of the request in the created *process-group*. However, should one or more processes of the request choose to successfully execute a *setpgrp*(2) system call, then such processes will **not** receive any signals sent by the *qdel*(1) command. This can lead to "rogue" requests whose constituent processes must be killed by other means such as the *kill*(1) command. However, NQS takes advantage of any UNIX implementations that provide a mechanism of "locking" a process, and all of its subsequent children in a particular *process-group*. For such UNIX implementations, this problem does not occur.

It is extremely wise for all processes of an NQS request to catch any **SIGTERM** signals. By default, the receipt of a **SIGTERM** signal causes the receiving process to die. NQS sends a **SIGTERM** signal to all processes in the established *process-group* for a batch request as a notification that the request should be prepared to be killed, either because of an *abort queue* command issued by an operator using the *qmgr*(1M) program, or because it is necessary to shutdown NQS and all running requests as part of a

general shutdown procedure of the local host.

It must be understood that the spawned *shell* ignores **SIGTERM** signals. If the current immediate child of the *shell* does not ignore or catch **SIGTERM** signals, then it will be killed by the receipt of such, and the shell will go on to execute the next command from the script (if there is one). In any case, the shell will not be killed by the **SIGTERM** signal, though the executing command will have been killed.

After receiving a **SIGTERM** signal delivered from NQS, a process of a batch request typically has sixty seconds to get its "house in order" before receiving a **SIGKILL** signal (though the sixty second duration can be changed by the operator).

All batch requests terminated because of an operator *NQS shutdown request* that did not specify the **–nr** flag are considered restartable by NQS, and are requeued (provided that the batch request shell process is still present at the time of the **SIGKILL** signal broadcast as discussed above), so that when NQS is rebooted, such batch requests will be respawned to continue execution. It is however, up to the user to make the request restartable by the appropriate programming techniques. NQS simply spawns the request again as though it were being spawned for the first time.

Upon completion of a batch request, a mail message can be sent to the submitter (see the discussion of the **–me** flag above). In many instances, the completion code of the spawned *Bourne* or *C-Shell* will be displayed. This is merely the value returned by the shell through the *exit*(2) system call.

Lastly, there is no good way to echo commands executed by unmodified versions of the *Bourne* and *C* shells. While the *C-shell* can be spawned in such a fashion as to echo the commands it executes, it is often very difficult to tell an echoed command from genuine output produced by the batch request, because no "magic" character such as a '$' is displayed in front of the echoed command. The *Bourne* shell does not support any echo option whatsoever.

Thus, one of the better ways to write the shell script for a batch request is to place appropriate lines in the shell script of the form:

```
echo "explanatory-message"
```

where the echoed message should be a meaningful message chosen by the user.

LIMITATIONS AND IMPLEMENTATION NOTES

Network queues have not yet been implemented.

In the present implementation, it is **not** possible to see the *stderr* or *stdout* files produced by the batch request while the request is **running**, unless the **-re** and **-ro** flags have been respectively specified.

Lastly, the strange "@$" syntax used to introduce *embedded argument* flags was chosen because it rarely conflicts with anything else present in a shell script file. NQS users with better minds will (rightly) suggest improved alternatives to this convention.

SEE ALSO

qdel(1), qdev(1), qlimit(1), qpr(1), qstat(1), qmgr(1M), plus mail(1), kill(2), setpgrp(2), signal(2)

NPSN HISTORY

Origin: Sterling Software Incorporated

August 1985 – Brent Kingsbury, Sterling Software
Original release.

May 1986
Second release.

Index

R

S

T

U