April 1996

Order Number: 312645-005

Paragon™ System Network Queueing System Manual

Intel® Corporation

WARNING

Some of the circuitry inside this system operates at hazardous energy and electric shock voltage levels. To avoid the risk of personal injury due to contact with an energy hazard, or risk of electric shock, do not enter any portion of this system unless it is intended to be accessible without the use of a tool. The areas that are considered accessible are the outer enclosure and the area just inside the front door when all of the front panels are installed, and the front of the diagnostic station. There are no user serviceable areas inside the system. Refer any need for such access only to technical personnel that have been qualified by Intel Corporation.

CAUTION

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

LIMITED RIGHTS

Preface

This manual describes the Network Queueing System (NQS) as it is implemented on the Paragon™ system. This manual shows you how to submit jobs to NQS, as well as how to set up and configure NQS for your site.

Organization

Chapter 1	Provides an overview of NQS and describes NQS concepts.
Chapter 2	Provides user-level procedures for submitting batch requests to the Paragon system and for monitoring the job's execution.
Chapter 3	Provides basic system administration procedures. Describes queue manipulation, displaying status, and using pipe queues.
Chapter 4	Provides advanced system administration procedures. Describes queue creation, batch area configurations, account mapping, and status reporting.
Chapter 5	A procedure for setting up and configuring NQS on your system.
Chapter 6	A command reference for all NQS commands.

Notational Conventions

This manual uses the following notational conventions:

Bold	Identifies command names and switches, system call names, reserved words, and other items that must be used exactly as shown.
Italic	Identifies variables, filenames, directories, processes, user names, and writer annotations in examples. Italic type style is also occasionally used to emphasize a word or phrase.
`Plain-Monospace`	Identifies computer output (prompts and messages), examples, and values of variables. Some examples contain annotations that describe specific parts of the example. These annotations (which are not part of the example code or session) appear in *italic* type style and flush with the right margin.
`Bold-Italic-Monospace`	Identifies user input (what you enter in response to some prompt).
`Bold-Monospace`	Identifies the names of keyboard keys (which are also enclosed in angle brackets). A dash indicates that the key preceding the dash is to be held down *while* the key following the dash is pressed. For example: **`<Break>`** **`<s>`** **`<Ctrl-Alt-Del>`**
[ ]	(Brackets) Surround optional items.
...	(Ellipsis dots) Indicate that the preceding item may be repeated.
|	(Bar) Separates two or more items of which you may select only one.
{ }	(Braces) Surround two or more items of which you must select one.

Applicable Documents

For more information, refer to the *Paragon™ System Technical Documentation Guide.*

Comments and Assistance

Intel Scalable Systems Division is eager to hear of your experiences with our products. Please call us if you need assistance, have questions, or otherwise want to comment on your Paragon system.

U.S.A./Canada Intel Corporation
Phone: 800-421-2823
Internet: support@ssd.intel.com

France Intel Corporation
1 Rue Edison-BP303
78054 St. Quentin-en-Yvelines Cedex
France
0590 8602 (toll free)

United Kingdom Intel Corporation (UK) Ltd.
Scalable Systems Division
Pipers Way
Swindon SN3 IRJ
England
0800 212665 (toll free)
(44) 793 491056
(44) 793 431062
(44) 793 480874
(44) 793 495108

Intel Japan K.K.
Scalable Systems Division
5-6 Tokodai, Tsukuba City
Ibaraki-Ken 300-26
Japan
0298-47-8904

Germany Intel Semiconductor GmbH
Dornacher Strasse 1
85622 Feldkirchen bei Muenchen
Germany
0130 813741 (toll free)

World Headquarters
Intel Corporation
Scalable Systems Division
15201 N.W. Greenbrier Parkway
Beaverton, Oregon 97006
U.S.A.
(503) 677-7600 (Monday through Friday, 8 AM to 5 PM Pacific Time)
Fax: (503) 677-9147

If you have comments about our manuals, please fill out and mail the enclosed Comment Card. You can also send your comments electronically to the following address:

techpubs@ssd.intel.com
(Internet)

Table of Contents

Chapter 2 Basic User Operations

Chapter 3
Basic System Administration

Chapter 4
Advanced System Administration

Chapter 5 NQS Setup and Configuration

Chapter 6 NQS Reference

List of Illustrations

List of Tables

Overview of NQS 1

The Network Queueing system (NQS) is the Paragon system's job queueing system. NQS allows you to submit parallel applications to the Paragon system for later execution.

During NQS setup, some or all of the Paragon system's compute nodes are configured to be under the control of NQS. Once a node is under the control of NQS, it is not available for interactive use using the standard Paragon partition management commands (such as **mkpart**).

The NQS manager creates *queues* to manage these compute nodes. Each queue is set up with attributes that control the type of application that can be submitted to it, such as the number of nodes an application requires, or the amount of CPU time the application requires. An application submitted to NQS is referred to as a *request*. Requests to NQS are submitted via the NQS **qsub** command.

This chapter provides an overview of NQS and the concepts relating to NQS operations on the Paragon system. If this is your first experience with NQS on a Paragon system, you should read this chapter and become familiar with the basic NQS concepts.

NOTE

While some versions of NQS are public domain software, NQS as implemented on the Paragon system has been greatly enhanced to support a parallel, multi-processor environment. Therefore, this manual documents NQS only as it is specifically implemented on the Paragon system.

Conversely, NQS functionality that is not appropriate to the Paragon system, such as device support, is not covered in this manual.

Basic NQS Concepts

The following discussions briefly describe some basic NQS concepts. These concepts are important in understanding the Network Queueing System.

Queues

NQS uses *queues* to group together jobs with similar attributes. An NQS queue has characteristics associated with it such as number of nodes, queue *priority* relative to other NQS jobs, and CPU time limits. NQS schedules jobs according to priority, job size, and aging factors.The NQS queues are defined by the NQS manager using the **qmgr** command. Also, the NQS manager can restrict some queues to certain users.

NQS uses two queue types—*batch* queues and *pipe* queues. A batch queue holds requests for scheduled or delayed processing. Batch queues are directly associated with the Paragon system's compute nodes. A pipe queue is a queue that can pass queued requests on to batch queues or other pipe queues. Pipe queues have no means to execute requests; they are only used to route requests to some other queue where compute power is available. Refer to Chapter 4 for an overview of both batch and pipe queues, as well as procedures to create them.

Run Limits

Queues can be assigned a *run limit* which controls how many jobs in a queue can run concurrently if compute nodes are available. For example, a two-node queue with a run limit of five will run five jobs concurrently if there are ten compute nodes available.

Node Groups

An NQS batch queue is associated with a specific set of nodes, called a *node group*. Node groups are defined during NQS setup and configuration, and associated with one or more queues using the **qmgr set node_group** subcommand. When NQS schedules a job, it determines if the nodes in the node group are available, creates a partition for that job on those nodes, runs the job, and removes the partition when the job completes.

Requests

A *request* is a shell script that contains all of the system-level commands required to invoke one or more parallel applications. A request is essentially a list of commands that is passed to NQS for execution. A request may be as simple as a single command line, or it may be as complex as a script written in the Bourne Shell, C-Shell, or Korn Shell. The NQS request is submitted to the Paragon system via the NQS **qsub** command. All of the commands in the shell script must meet the following requirements.

- The commands must not require the direct services of a physical device (other than the CPUs that are executing the batch request).

- The commands must be able to be executed without any user intervention when the proper command interpreter is invoked (such as */bin/csh*, */bin/sh*, or */bin/ksh*).

The user then submits the shell script using the **qsub** command. The **qsub** command line contains a **-q** flag that names the destination queue, any optional invocation flags, and then the name of the shell script that executes on the Paragon system. For example:

```
% qsub -q batchq myjob
Account = 120
request 127.gumshoe submitted to queue: batchq
```

This request places the shell script *myjob* in the queue named *batchq*. The request will run after the jobs that precede it in the queue have run (assuming all requests have the same *queue priority*).

The NQS output `Account = 120` indicates that the MACS account that will track CPU usage has an account ID of 120. The output `127.gumshoe` indicates the request ID assigned by NQS (`127`) and the host name of the system from which the job has been submitted (`gumshoe`).

Queue Status

You can use the **qstat** command or the **qmgr** subcommand **show queue** to see the status of requests in both batch and pipe queues.

Each request moves from one request state to another, depending on its status in the queue. The request states are as follows:

arriving	The request is in the process of being queued from another (possibly remote) pipe queue. After leaving this state, the request enters either the *waiting*, *queued*, *running*, or *routing* state.
queued	The request has been accepted in a queue, and is ready to enter the *running* or *routing* request state.
routing	The request resides in a pipe queue and is being routed and delivered to another queue destination.
running	The request resides in a batch queue and is being executed.
waiting	The request is waiting for some finite time interval to pass. After leaving this state, the request enters the *queued*, *running*, or *routing* state.

Refer to Chapter 2 for additional information about submitting requests, or for monitoring a request as it executes on the Paragon system.

CPU Time Limits

By default, an NQS batch request can run for an unlimited amount of time. However, the NQS queues can be configured by the system administrator to limit the time a request can run. This time limit can be enforced as either *wall-clock* hours or *node-hours*, depending on the setting of the *wallclock_limits* configuration parameter.

Wall-clock time is just that—how long a request runs as measured by the clock on the wall. Node-hours, on the other hand, are calculated as the length of the request in wall-clock time, multiplied by the number of nodes used by the request.

Submitting an NQS Request

When you submit an NQS request, you need to be aware of the queue characteristics of the NQS queues that exist on the Paragon system.

If your application requires the same nodes and CPU time as those defined for a given queue, you can submit a request to that queue.

If the parameters associated with a given queue do not correspond to the needs of your application, you have these options:

- If your application requires fewer system resources than the queue will allow, you can use **qsub** flags to change the necessary parameters in an existing queue (e.g., use the **-lT** flag to reduce the CPU time limit and the **-lP** flag to decrease the number of nodes reserved for the application).
- If your application requires more system resources than the queue will allow (i.e., more nodes, longer CPU time limit, or higher priority), the NQS network manager must create a queue that has parameters matching your requirements.

You use a shell script to invoke your application. The shell script is then included on the **qsub** command line. For example, assume you have an application program named *myapp* that you want to run on sixteen nodes. Create a shell script (named *myjob* in this example) that contains the appropriate job execution command, as follows:

```
myapp [args]
```

You then use the **qsub** command to submit the batch job to the NQS system for processing, as follows:

```
% qsub -q q2s myjob
Account = 0
request 127.skyline submitted to queue: q2s
```

In this example, you only provide the queue name (*q2s*) and the name of the shell script file. The queue to which you submit the job defines the number of nodes, the CPU time limit, and the priority that the request will inherit. Standard output and standard error messages will be placed in files named *myjob.oNN* and *myjob.eNN*, respectively, in the directory from which the request is submitted. The *NN* value is a job number assigned by NQS and shown when the **qsub** command executes. In this case the files would be named *myjob.o127* and *myjob.e127*, respectively.

In many cases, you will want to be notified when the request starts and finishes execution. The **qsub** command has flags and optional parameters that allow you to modify this and many other aspects of your request. The following example shows a submittal that uses several **qsub** flags:

```
% qsub -q q2s -lT 1200 -mb -me -nr myjob
request 127.skyline submitted to queue: q2s
```

This example submittal is the same as the earlier submittal, except additional conditions have been added to the job. The CPU time limit has been lowered to 1200 seconds (**-lT 1200**) or twenty minutes. The NQS software will now send mail to the user that indicates the beginning (**-mb**) and end (**-me**) of execution. If the job is interrupted, the **-nr** flag specifies that it will not be automatically restarted.

You can use the **qstat** command to check on the status of your job once it has been queued. The following example checks on the status of all jobs queued to the *q2s* queue.

```
% qstat -a q2s
=============================================================================
NQS Version:2        DEVICE PIPE REQUESTS on skyline
=============================================================================
    REQUEST         NAME    OWNER   QUEUE       PRI     NICE   CPU      MEM      STATE
   123.skyline   test_p.  sdsc     q2s        11.0     20  UNLIM.   UNLIM.  RUNNING
   124.skyline   test_p.  sdsc     q2s        11.0     20  UNLIM.   UNLIM.  RUNNING
   127.skyline   myjob    sdsc     q2s        11.0     20  UNLIM.   UNLIM.  RUNNING
```

If you want to delete a job that you have submitted, use the **qdel** command as follows:

```
% qdel 127.skyline
```

where *127.skyline* is the request ID that was obtained from a **qstat** command or the ID given when the job was first submitted.

To kill a job which is already executing, use the following command:

```
% qdel -k 127.skyline
```

or

```
% qdel -30 127.skyline
```

The first form of **qdel** sends a SIGKILL signal to the job, and the second form sends a specific signal (number *30*, SIGUSR1 in this case) to the job.

You can omit the machine name (*skyline* in this case) if your are logged on to the machine called *skyline*. For example:

```
% qdel -k 127
```

NQS Operation and Management

The NQS system recognizes three user types. Each user type has a different access to the **qmgr** command, which is used for NQS setup, reconfiguration, and system administration.

- An NQS *user* is anyone who uses NQS to submit jobs to the Paragon system, as opposed to someone who configures or maintains NQS. While the NQS user has access to a few **qmgr** subcommands, this functionality is duplicated by other NQS commands. Refer to Chapter 2 for basic user procedures.
- An NQS *operator* can execute only a subset of the **qmgr** subcommands. An NQS operator is able to submit batch queue requests, and perform other operations associated with these requests. The operator is someone who maintains the system during off hours, with less responsibility than a system administrator. Not all sites will have NQS operators. Refer to Chapter 3 for operator-level system administration procedures.
- An NQS *manager* has access to all of the **qmgr** subcommands and functions as the NQS system administrator. The NQS manager is able to define, configure, and manage queues in addition to performing all the functions that an NQS operator can perform. An NQS manager is capable of changing any NQS characteristic on the Paragon. Refer to Chapter 4 for manager-level system administration procedures.

NQS Daemons and Job Flow

Each machine in the NQS environment must set up and use three separate daemons: the local daemon (*nqsdaemon*), the log daemon (*logdaemon*), and the network daemon (*netdaemon*).

- The local daemon processes requests executing on the local machine. It manages the batch queues and schedules all requests for execution. It also detects and forwards any requests that are to be executed on a remote machine.
- The log daemon records a log of NQS requests and activities.
- The network daemon handles all remote requests. It is essentially the same as the local daemon that forwards requests, but it receives its requests through sockets from remote machines.

The daemons are permanent and are created when NQS starts up.

Figure 1-1 shows the typical job flow from a workstation to the Paragon system.

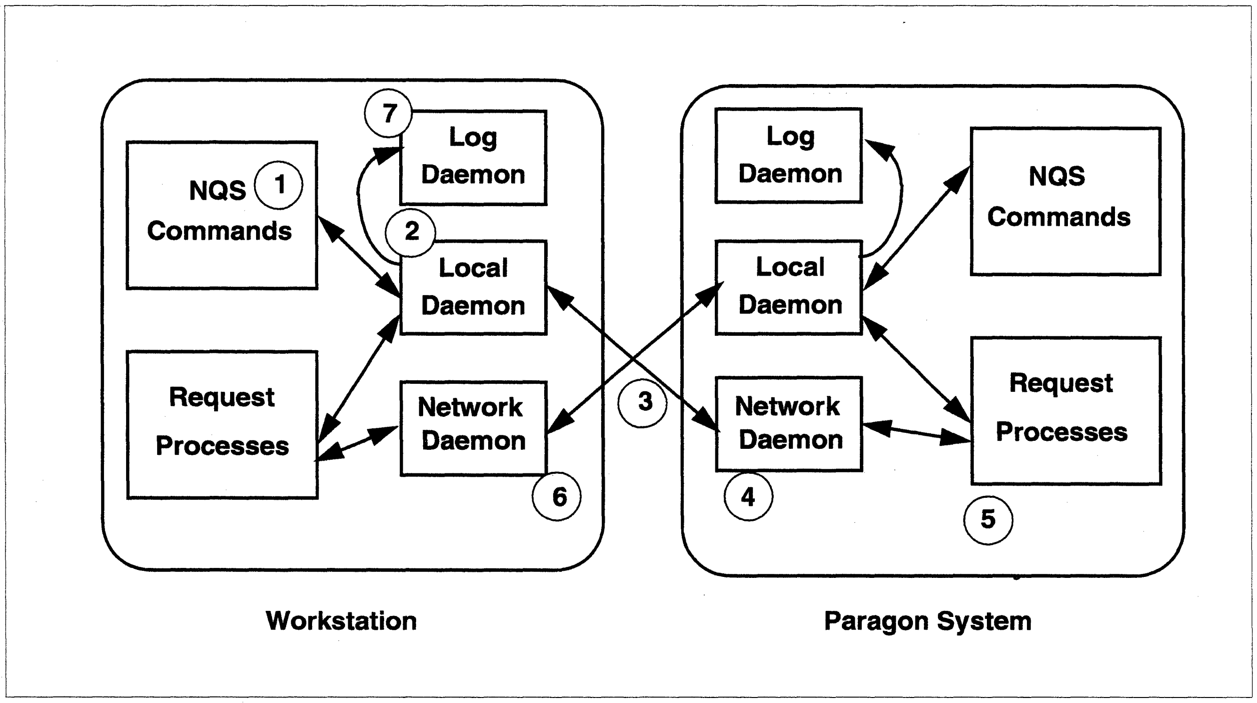


Figure 1-1. NQS Request Processing

The sequence of events in this job flow is as follows:

1. The user submits a request from the workstation to the Paragon system using the NQS **qsub** command:

```
% qsub -q remote_q_1 script1
```

2. The **qsub** command packages up the job and sends it to the local daemon.

3. The local daemon determines that the job is remote and sends it (via a socket call) to the network daemon on the Paragon system.

4. The network daemon on the Paragon system retrieves the shell script from the spool directory, and sets up the *stdout* and *stderr* file descriptors.

5. The Paragon system executes the shell script.

6. After executing the shell script, the Paragon system sends the *stderr* and *stdout* files back to the originating machine through instructions to the originating machine's network daemon.

7. A message is sent to the log daemon to create an entry for the request in the log file.

There are many more details that apply to request processing, but this general sequence shows the physical and chronological relationships that exist between the machines in a network.

Getting Started

Once you are familiar with the basic NQS concepts presented in this chapter, your next step depends on whether you are an NQS *user* or *system administrator.*

If you are a basic NQS user and need information about submitting job requests or monitoring the request as it executes on the Paragon system, refer to Chapter 2.

If you are an NQS operator or manager, refer to Chapter 3 for basic system administration procedures and Chapter 4 for advanced system administration procedures. If you are configuring NQS for your particular site, refer to Chapter 5. The manual pages for the NQS commands appear in Chapter 6 and are also available online, using the **man** command.

Basic User Operations 2

The procedures in this section show you how to create, run, and monitor batch requests within the NQS environment. You should be familiar with the concepts discussed in Chapter 1 before continuing with this chapter.

Batch Requests

The process of submitting a batch request involves two basic processes:

- Create a shell script that contains the request.
- Submit the request with the **qsub** command. (See Table 2-1 on page 2-9 for a functional listing of all of the **qsub** command's invocation flags.)

In addition, you can monitor the job request (with **qstat**) or delete the job request (with **qdel**).

Composing the Shell Script

The batch request is contained within a shell script. In its simplest form, the batch request consists of the commands that invoke your application. For example:

```
% cat myapp.sh
mxm -s 500 -v
```

The shell used to execute the script is often your login shell, although the shell is determined by the NQS manager. The script will execute in your home directory, unless the script explicitly changes the directory using **cd**.

If you are unfamiliar with shell scripts, the standard OSF/1 documentation gives user and reference information on developing and invoking shell scripts. The **qsub** manual page provides more specific information on including **qsub** invocation flags in the shell script.

NOTE

The *NX_DFLT_PART* environment variable provides the name of the default partition. Be aware that NQS modifies the *NX_DFLT_PART* environment variable before it executes your shell script. If you use Paragon system commands within your script that use this variable, your script should obtain the name with `echo $NX_DFLT_PART`, rather than using a name defined before the batch request was made.

Submitting the Batch Request

The batch request is contained within the shell script that you composed in the previous section. You submit the batch request to a batch queue using the NQS **qsub** command.

For example, assume that you have a program called *myjob* that you want to run on a 16-node partition, and you have a shell script called *job1* that runs the program. You would submit the job via a batch request to a 16-node queue.

You might also have a second job that you want to send to the same queue. In the following example, two jobs (*job1* and *job2*) have been queued to batch queue *q16s*:

```
% qsub -q q16s job1
Account = 0
Request 136.treebrd submitted to queue: q16s.
% qsub -q q16s job2
Account = 0
Request 137.treebrd submitted to queue: q16s.
```

As shown in the previous example, you can immediately submit another request without waiting for the first request to finish. With all other parameters being equal, batch requests to the same queue execute in the order they are submitted.

Specifying a Queue

The **-q** switch in the previous example specifies the queue that you are submitting the request to. If you don't use the **-q** switch, **qsub** will look for a default queue in your environment variable *QSUB_QUEUE*. If you leave the switch out and have not defined *QSUB_QUEUE*, NQS uses the default queue set up by the system administrator (using the **qmgr** command, as described on page 4-12). If **qsub** cannot find a default queue, it will fail.

To see what queues are available, use the **qstat -b** command:

```
% qstat -b
============================================
NQS Version: 2  BATCH QUEUES on prefect
============================================
QUEUE NAME       STATUS    TOTAL    RUNNING  QUEUED    HELD TRANSITION  NODE_GROUP
-------------------------------------------------------------------------------
q4-30            AVAILBL     0       0/1        0        0        0        0
q2-30            AVAILBL     0       0/1        0        0        0        2
q2-60            AVAILBL     0       0/1        0        0        0        1
```

To see the characteristics of a particular queue, include the **qstat -b** and **-f** flags:

```
% qstat -b -f q4-30

=====================================================
NQS Version:2  BATCH      QUEUE:  q4-30.prefect              status:  AVAILBL
=====================================================
                                        Priority: 0
ENTRIES:
        Total:    0       Running: 0
        Queued:   0       Held:    0       Transition: 0

RUN_LIMIT:
        Runlimit:  1

NODE_GROUP:
        Node_group: 0

COMPLEX MEMBERSHIP:


RESOURCES:
  Per-proc core file size limit= UNLIMITED <DEFAULT>
  Per-process data size limit  = UNLIMITED <DEFAULT>
  Per-proc perm file size limit= UNLIMITED <DEFAULT>
  Per-proc execution nice value= 0 <DEFAULT>
  Per-req number of cpus limit = 1 <DEFAULT>
  Per-process stack size limit = UNLIMITED <DEFAULT>
  Per-process CPU time limit   = UNLIMITED <DEFAULT>
  Per-request CPU time limit   = 1800.0
  Per-process working set limit= UNLIMITED <DEFAULT>

ACCESS
  Unrestricted access
```

Getting Job Request Start/Finish Notification

In many cases, you will want to be notified when the request starts and finishes execution. Use the **qsub -mb** switch to get notification when the request starts; use the **qsub -me** switch to get notification when the request ends. For example:

```
% qsub -q q2-10 -mb -me myapp
Account = 0
Request 127.prefect submitted to queue: q2-10
```

The NQS system will now send you mail at the beginning (**-mb**) and end (**-me**) of the job.

Limiting the Number of Nodes

If your application needs fewer nodes than the number of nodes allowed by the queue, you can use the **qsub -lP** switch to limit the number of nodes your application uses. The value specified must be less than or equal to the queue's node limit. For example, the following entry limits your application to twenty nodes:

```
% qsub -q q2-10 -lP 20 myapp
```

When specifying the number of nodes, keep in mind that the total node usage is determined by the length of time a request ran multiplied by the number of nodes of the request. Limiting the number of nodes can save system resources.

NOTE

The **-lP** switch is only available on Paragon systems. On a remote workstation, you can specify the number of nodes with the *NCPUS* environment variable, and then use the **qsub -x** switch to export the environment variable.

Exporting Environment Variables

The **-x** switch exports all of the user environment variables with the request.

Limiting CPU Time Usage

If your application doesn't need to run as long as the queue will allow, you can specify a shorter run time in seconds with the **qsub -lT** flag. For example:

```
% qsub -q q2-10 -lT600,60 myapp
```

The application will run for ten minutes, and you will receive a warning one minute before it is killed.

Specifying an Account Identification

If your site has *MACS* (Multi-User Accounting and Control System), the **qsub -c** switch lets you specify an account identification at the time of job submission. For example, the following entry bills the batch job to the account *debug*:

```
% qsub -q q2-10 -c debug myapp
Account = 120
```

If you do not specify an account using the **-c** switch, NQS uses your current account.

The **-c** switch is only available on Paragon systems. On a remote workstation, you can specify an account ID with the *ACCOUNT* environment variable, and then use the **qsub -x** switch to export the environment variable.

Finding Standard Output and Standard Error

Standard output and standard error messages are written to files named *jobname*.**o***reqID* and *jobname*.**e***reqID*, respectively, in the directory from which the request is submitted. The *reqID* value is a job number assigned by NQS when the **qsub** command executes. Note that the *jobname* is truncated to eight chatacters.

Depending on the shell strategy in place and the setting of the *use_login* configuration parameter, the *jobname*.**e***reqID* file might contain error messages created because an NQS job doesn't have an attached terminal. Refer to "use_login" on page 5-16 for information on the cause and prevention of these errors.

Monitoring Request Execution

You can monitor queue status and the completion status of your requests using the **qstat** command. For example:

```
% qstat
==============================================================
NQS Version:2    BATCH  PIPE REQUESTS on prefect
==============================================================
    REQUEST     NAME    OWNER   QUEUE        PRI  NICE CPU     MEM     STATE
   82.prefect  meshpp  dand    q2-10        2.0    20 600     UNLIM.  RUNNING
   83.prefect  debug   dand    q2-10        2.0    20 600     UNLIM.  QUEUED
```

Deleting a Batch Request

You may occasionally need to delete a batch request after you have submitted it. The most direct way to delete a batch request is with the **qdel** command (use **qdel -k** to kill a job that has started running). Refer to the **qdel** command description for more information. In the following example, two jobs are checked and then deleted:

```
$ qstat
==============================================================
NQS Version:2    BATCH  PIPE REQUESTS on prefect
==============================================================
    REQUEST     NAME    OWNER   QUEUE        PRI  NICE CPU     MEM     STATE
  118.prefect  job1    dand    q2-10        2.0    20 600     UNLIM.  RUNNING
  119.prefect  job2    dand    q2-10        20.3   20 600     UNLIM.  QUEUED

% qdel 119
Request 119 has been deleted.
% qdel -k 118
Request 118 is running, and has been signalled.
% qstat
==============================================================
NQS Version:2    BATCH  PIPE REQUESTS on prefect
==============================================================
    REQUEST     NAME    OWNER   QUEUE        PRI  NICE CPU     MEM     STATE
```

Modifying a Batch Request

In general, a batch request cannot be modified by the user once it has been queued, but you can delete it (as described previously) and then resubmit a modified request.

Using Pipe Queues

A pipe queue is a pathway to a batch queue. Pipe queues can only be created, deleted, or modified by an NQS manager.

Submitting a Request

You use the standard NQS **qsub** command when submitting a request to a pipe queue. You should be aware of the destinations of the pipe queues in your system before submitting requests to the queues.

```
% qsub -q pq job1
Account = 0
Request 144.treebrd submitted to queue: pq.
```

The **-q** *queue* option specifies the pipe queue that will accept the request. You must use this option since their are no default pipe queues. You can immediately submit another request without waiting for earlier requests to finish.

Monitoring Request Execution

You can monitor requests to pipe queues in the same manner as you would monitor requests to batch queues. Use the NQS **qstat** command to monitor the status of queues and requests in the NQS environment.

If, for some reason, the pipe or destination queues are unable to process a request, NQS sends mail to the originator and deletes the request. If the request fails to execute after being accepted, a message is sent to *stderr* and to the NQS log file. The following example shows the messages and mail returned when a request to a pipe queue fails to complete.

```
% qsub -q pq -x job1
Account = 0
Request 143.treebrd submitted to queue: pq.
You have new mail.

% mail
From root Wed Jan  9 18:05 GMT 1991
Subject:  NQS request:  143.treebrd failed.

Request name:   job1
Request owner:  dand
Mail sent at:   Wed Jan  9 10:05:06 PST 1991
Server for request did not return a completion code.
Request failed.
Request files placed in NQS failed directory.
```

Possible reasons for request failure at the destination include the following:

- Remote host failure.
- Queue type disagreements with the request type.
- Insufficient queue space.
- Destination queue disabled (unable to accept new requests).
- Lack of proper account authorization.

There are many other possible reasons for request failure, even after it is initially accepted. NQS supports *wait* and *retry* parameters that may allow an initially failed request to be ultimately successful.

Functional List of qsub and qstat Flags

Since the **qsub** and **qstat** commands are a basic component of most user operations, the following tables group the **qsub** and **qstat** flags into functional areas. For the complete syntax of the **qsub** and **qstat** commands and their associated invocation flags, refer to Chapter 6 or the online manual pages.

Table 2-1. Functional List of qsub Flags (1 of 2)

Function	Flag	Effect
General Invocation	**-c** *account*	Establish per-request account.
	-nr	Declare that batch request is not restartable.
	-q *queue name*	Queue request in the stated queue.
	-s *shell name*	Specify shell to interpret the batch request script.
	-x	Export all environment variables with request.
Job Notification	**-mb**	Send mail when the request begins execution.
	-me	Send mail when the request ends execution.
	-mu *username*	Send mail for the request to the stated user.
Controlling Output	**-e** *destination*	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.
	-re	Remotely access the *stderr* output file.
	-ro	Remotely access the *stdout* output file.
	-o *destination*	Direct *stdout* output to the stated destination.
	-r *request-name*	Assign stated request name to the request.
	-z	Submit the request silently.
Overriding Queue Limits	**-ln** *limit*	Establish per-request nice value limit.
	-lP *limit*	Establish per-request number of nodes.
	-lT *limit*	Establish per-request CPU time limits.
	-ls *limit*	Establish per-process stack-segment size limits.

Table 2-1. Functional List of qsub Flags (2 of 2)

Function	Flag	Effect
Unsupported, but Passed to Other Systems[1]	**-lc** *limit*	Establish per-process corefile size limit.
	-ld *limit*	Establish per-process data-segment size limits.
	-lf *limit*	Establish per-process permanent-file size limits.
	-lt *limit*	Establish per-process CPU time limits.
	-lw *limit*	Establish per-process working set limit.
	-lF *limit*	Establish per-request permanent-file space limits.
	-lm *limit*	Establish per-process memory size limits.
	-lM *limit*	Establish per-request memory space limits.
	-lv *limit*	Establish per-process temporary-file size limits.
	-lV *limit*	Establish per-request temporary-file space limits.

1. These flags are not supported on the Paragon system, but can be used when submitting NQS jobs from the Paragon system to other systems that do support the flag.

Table 2-2. Functional List of qstat Flags

Function	Flag	Effect
Queue Status	**-b**	Displays batch queues.
	-d	Displays device queues.
	-f	Shows queues or requests in a full format.
	-n	The queue header and trailer are not displayed.
	-p	Displays pipe queues.
Request Status	**-a**	Displays the status of requests belonging to all users.
	-h *host-name*	Displays requests for queues on the *host-name* host.
	-l	Same as **-f**.
	-s *state*	Displays only those requests with the specified *state* (**r**unning, **h**olding, or **q**ueued).
	-u *user*	Shows only those requests belonging to *user*.
	-U	Same as **-a**.
	-v	Displays output in 132-column format.

Basic System Administration 3

This chapter describes some basic NQS monitoring and maintenance tasks that can be performed by someone with NQS *operator* privileges. In general, an NQS operator is able to manage queues and requests, to get information, and to perform some network operations.

All other system administration functions are reserved for the NQS manager. (An NQS *manager* is able to define, configure, and manage queues in addition to performing all the functions that an NQS operator can perform. These tasks are described in Chapter 4.)

You should be familiar with the NQS concepts discussed in Chapter 1 and the basic user operations discussed in Chapter 2 before continuing with this chapter.

Using the qmgr Utility

Much of the system administration for NQS is done through the **qmgr** utility. Both NQS operators and NQS managers can use the **qmgr** utility, but operators can use only a restricted subset of the **qmgr** subcommands (see Table 3-1 on page 3-3). For example, manager privileges are required to create a queue. To see if you have manager privileges, invoke the **qmgr** utility and enter the **show managers** subcommand:

```
% qmgr
Mgr:show managers

  dand:o
  archer:m
  root:m
```

The o and m indicate operator and manager, respectively. If you do not have manager privileges, you will have to get an existing manager (or the system administrator) to add you as manager using the **add managers** subcommand. This command is used to add operators, add managers, or change the privilege level of NQS users.

Getting Help With qmgr Subcommands

You can use the **qmgr help** subcommand to get help information on all of the **qmgr** subcommands. For example:

```
% qmgr
Mgr: help move queue

    The command:

      MOVe Queue <from-queue-name> <to-queue-name>

    is used to move all requests from the queue specified by
    <from-queue-name> to the queue specified by <to-queue-name>.

    Operator privileges are required to use this command.
```

Automating qmgr Input

The **qmgr** command will accept input from a redirected file. This can be useful in speeding up large amounts of input to **qmgr**. For example:

```
% cat qmgr_queues.txt
create batch_queue q2-30 priority=0
enable queue q2-30
start queue q2-30

% qmgr < qmgr_queues.txt
Mgr: Queue q2-30 created.
NQS manager[TCML_COMPLETE ]: Transaction complete at local host.
Mgr: NQS manager[TCML_COMPLETE ]: Transaction complete at local host.
Mgr: NQS manager[TCML_COMPLETE ]: Transaction complete at local host.
Mgr:
%
```

Abbreviating qmgr Subcommands

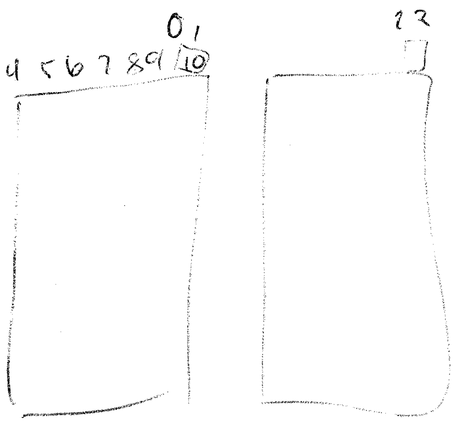
Note that you can enter an abbreviation for a subcommand instead of entering the entire subcommand. The subcommand abbreviations are indicated with capital letters in the syntax portion of the manual page for **qmgr**, which you can find in Chapter 5. For example, you can enter the subcommand **abort queue** as **a q**. In this chapter, however, the full name of the **qmgr** subcommands is used.

List of Operator-Level qmgr Subcommands

Table 3-1 lists all of the **qmgr** subcommands available to the operator.

Table 3-1. Functional List of qmgr Subcommands

Function	Subcommand
Queue Control	**start queue** **stop queue** **enable queue** **disable queue** **delete queue**
Job Control	**abort queue** **purge queue** **move queue** **move request**
Batch Queue Resource Limits	**set run_limit**
Queue Complexes	**set complex run_limit**
Status and Display	**show all** **show queue** **show parameters** **show managers** **show limits_supported** **show long queue**
Miscellaneous	**exit** **help** **shutdown** **set log_file** **lock local_daemon** **unlock local_daemon**



Displaying NQS Information

There are several ways to get information about the NQS operating parameters and information about the status of the NQS queues.

Displaying NQS Parameters

The NQS operating parameters include default values and paths that apply to the Paragon system and its operation in the NQS environment. You can use the **qmgr** command's **show all** or **show parameters** subcommands to determine the parameters that apply to the Paragon system. The following example uses the **show parameters** subcommand:

```
% qmgr
Mgr: show parameters

 Debug level = 0
 Default batch_request priority = 31
 Default batch_request queue = NONE
 Default destination_retry time = 72 hours
 Default destination_retry wait = 5 minutes
 Default device_request priority = 31
 No default print forms
 Default print queue = NONE
 Global batch run limit = 64
 Global pipe limit = 5
 (Pipe queue request) Lifetime = 168 hours
 Log_file = /dev/null
 Mail account = root
 Maximum number of print copies = 2
 Maximum failed device open retry limit = 2
 Maximum print file size = 1000000 bytes
 Netdaemon = //usr/lib/nqs/netdaemon
 Netclient = //usr/lib/nqs/netclient
 Netserver = //usr/lib/nqs/netserver
 (Failed device) Open_wait time = 5 seconds
 NQS daemon is not locked in memory
 Next available sequence number = 11
 Batch request shell choice strategy = FREE
 Soft user_limit = 32000
 Hard user_limit = 32000

Mgr:
```

Displaying Queue Status

The **qmgr show queue** and **show long queue** subcommands are useful ways to get information about the NQS queues.

```
Mgr: show queue
============================================
NQS Version: 2  BATCH  PIPE QUEUES on prefect
============================================
QUEUE NAME      STATUS   TOTAL   RUNNING  QUEUED   HELD TRANSITION  NODE_GROUP
-----------------------------------------------------------------------------
q4-30           AVAILBL    6       0/1       5       0       0         0
q2-30           AVAILBL    0       0/1       0       0       0         1
q2-60           AVAILBL    0       0/1       0       0       0         1
q2-10           AVAILBL    0       0/5       0       0       0         2


Mgr: show long queue q4-30

=====================================================
NQS Version:2  BATCH     QUEUE:  q4-30.prefect                 status:  AVAILBL
=====================================================
                                        Priority: 0
ENTRIES:
        Total:   11     Running: 1
        Queued:  10     Held:    0      Transition: 0

EFFECTIVE PRIORITY LIMITS:
        Non-prime time: 3
            Prime time: 1

RUN_LIMIT:
        Runlimit:  1

NODE_GROUP:
        Node_group: 0

COMPLEX MEMBERSHIP:


RESOURCES:
  Per-proc core file size limit= UNLIMITED <DEFAULT>
  Per-process data size limit  = UNLIMITED <DEFAULT>
  Per-proc perm file size limit= UNLIMITED <DEFAULT>
  Per-proc execution nice value= 0 <DEFAULT>
  Per-req number of cpus limit = 66
  Per-user # of requests limit = UNLIMITED <DEFAULT>
  Per-process stack size limit = UNLIMITED <DEFAULT>
  Per-process CPU time limit   = UNLIMITED <DEFAULT>
  Per-request CPU time limit   = UNLIMITED <DEFAULT>
  Per-process working set limit= UNLIMITED <DEFAULT>

ACCESS
  Unrestricted access
Mgr:
```

Queue Control

There are several operator-level **qmgr** subcommands that allow you to control the NQS queues. Depending on what you are trying to accomplish, you can either stop a queue or disable it.

stop queue	Stops the queue after allowing running jobs to finish. The other requests remain in the queue but don't run until **start queue**. When a queue is stopped, it will still accept requests.
start queue	Starts the queue, which then can execute the queued requests.
disable queue	Prevents a queue from accepting requests.
enable queue	Enables the queue, allowing it to accept new requests.

Request Control

There are several operator-level **qmgr** subcommands that allow you to control the job requests within a queue. The following discussions describe these subcommands.

Removing all Requests From a Queue

You can use **qmgr** subcommands to remove all of the job requests from a queue. The difference in using the **qmgr** subcommands (instead of the **qdel** command, discussed on page 3-7) is that they affect all requests in a given queue, while the **qdel** command allows deletion of a single request.

purge queue	Kills all non-executing requests in a queue.
abort queue	Kills all executing requests in the queue.
shutdown	Kills all executing requests in all queues, but the requests are requeued for later execution. Also shuts down the NQS daemon.

Deleting a Queued Request

To delete a user's request that is queued, use the **qdel -u** switch to specify the user, and then supply the request number (or *request-id*):

```
% qstat -a

==============================================================
NQS Version:2    BATCH  PIPE REQUESTS on prefect
==============================================================
    REQUEST     NAME     OWNER    QUEUE          PRI  NICE CPU      MEM      STATE
    39.prefect meshp    sues     q4-30          32.0   20 UNLIM.   UNLIM.   RUNNING
    40.prefect myapp    dand     q4-30          32.0   20 UNLIM.   UNLIM.   QUEUED

% qdel -u dand 40
Request 40 has been deleted.
```

Deleting a Running Request

To delete a user's request that is queued and running, use the **qdel -k** switch to send a kill signal to the running request, along with the **-u** switch to specify the user, and the request number:

```
% qstat -a
==============================================================
NQS Version:2    BATCH  PIPE REQUESTS on prefect
==============================================================
    REQUEST     NAME     OWNER    QUEUE          PRI  NICE CPU      MEM      STATE
    39.prefect meshp    sues     q4-30          32.0   20 UNLIM.   UNLIM.   RUNNING

% qdel -k -u sues 39
Request 39 is running, and has been signalled.
% qstat -a
==============================================================
NQS Version:2    BATCH  PIPE REQUESTS on prefect
==============================================================
    REQUEST     NAME     OWNER    QUEUE          PRI  NICE CPU      MEM      STATE
```

Moving all Requests From one Queue to Another

You can use the **qmgr move queue** subcommand to move all of the job requests from one queue to another queue. All requests retain their attributes in the new queue. For example:

```
Mgr: move queue q2-10 q4-30
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
```

Moving a Single Request From one Queue to Another

You can use the **qmgr move request** subcommand to move a single request from one queue to another queue. The request retains its attributes in the new queue.For example:

```
Mgr: move request =41 q2-60
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
```

Setting Run Limits

A *run limit* is the number of requests that a queue will allow to run concurrently. You can change the run limit with the **qmgr set run_limit** subcommand.

```
% qmgr
Mgr: set run_limit = 10 q2s
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
Mgr: exit
```

To enable this feature, the configuration parameter *chk_runlimit* must be set to `1` in the */usr/spool/nqs/conf/sched_param* file.

Advanced System Administration 4

This chapter provides information primarily for the NQS *manager*. The NQS manager controls the NQS environment to ensure that users can best utilize the Paragon system.

You should be familiar with the NQS concepts discussed in Chapter 1, the basic user operations discussed in Chapter 2, and the basic system administration discussed in Chapter 3 before continuing with this chapter.

At a minimum, there must be one NQS manager in the network. The NQS manager has the following system administration responsibilities:

- Establish and maintain queues on the Paragon system.
- Configure the Paragon system's compute nodes into a *batch area* under the control of NQS.
- Designate a *shell strategy* for the batch requests submitted to the Paragon system.
- Maintain and assign machine identification numbers (mid) for all machines in the NQS network.

For the most part, NQS system administration is performed using the **qmgr** utility. See "Using the qmgr Utility" on page 3-1 if you have never used **qmgr** before. Table 4-1 lists all of the **qmgr** subcommands. You can also find a manual page for the **qmgr** utility in Chapter 6, or on-line using the **man** command.

NOTE

In this chapter, the full name of the **qmgr** subcommands is used in the examples. You can, however, use subcommand abbreviations instead of the full text. The command abbreviations are indicated with capital letters in the syntax portion of the **qmgr** manual pages, which can be found in Chapter 6.

The following table provides a functional list of the **qmgr** subcommands.

Table 4-1. Functional List of qmgr Subcommands (1 of 3)

Function	Subcommand
Queue Access	**add users** **delete users** **add groups** **delete groups** **set no_access** **set unrestricted_access** **set queue_request_limit**
Queue Control	**start queue** **stop queue** **enable queue** **disable queue** **delete queue**
Job Control	**abort queue** **purge queue** **move queue** **move request**
Batch Queue Creation	**create batch_queue** **set node_group** **set default batch_request queue** **set no_default batch_request queue** **set shell_strategy fixed** **set shell_strategy free** **set shell_strategy login**
Batch Queue Resource Limits	**set run_limit** **set priority** **set global batch_limit** **set hardulimit** **set softulimit** **set nice_value_limit** **set per_request ncpus** **set per_request cpu_limit** **set queue_request_limit**
Batch Queue EPL	**set epl_nprime** **set epl_prime**

Table 4-1. Functional List of qmgr Subcommands (2 of 3)

Function	Subcommand
Pipe Queues	**create pipe_queue** **set destination** **add destination** **delete destination** **set pipe_client** **set default destination_retry time** **set default destination_retry wait** **set lifetime** **set network client** **set network server**
Queue Complexes	**create complex** **add queues** **delete complex** **remove queues** **set complex run_limit**
Status and Display	**show all** **show queue** **show long queue** **show parameters** **show managers** **show limits_supported**
Miscellaneous	**set managers** **add managers** **delete managers** **exit** **help** **shutdown** **set log_file** **set network daemon** **set no_network_daemon** **lock local_daemon** **unlock local_daemon** **set debug**

Table 4-1. Functional List of qmgr Subcommands (3 of 3)

Function	Subcommand
Devices[1]	**create device_queue** **create device** **show device** **show forms** **add device** **enable device** **disable device** **delete device** **add forms** **delete forms** **set default device_request priority** **set default print_request forms** **set no_default print_request forms** **set default print_request queue** **set no_default print_request queue** **set device** **set device_server** **set forms** **set maximum copies** **set maximum open_retries** **set maximum print_size** **set open_wait**

1. Devices are not documented in this manual because they are not useful in the Paragon system. You can, however, use the **qmgr help** subcommand to get information about the device subcommands.

Batch Queues

As an NQS manager, you are responsible for setting up the NQS queue configurations using **qmgr** subcommands. NQS recognizes *batch* queues and *pipe* queues.

A *batch* queue holds requests for scheduled or delayed processing. A *pipe* queue is a queue that can pass queued requests on to batch queues or other pipe queues.

Batch Queue Overview

All batch requests ultimately end up in a batch queue. The destination (target) batch queue can exist at the same location where the request originated, or at a remote location.

Batch Execution Sequence

The execution sequence of a batch request within the target batch queue is as follows:

1. A temporary copy of any output file is created in a location that is known to NQS. This allows the *stderr* and *stdout* output files to be spooled to other (possibly remote) destinations.

2. Any additional environment variables associated with the request are placed in the environment set for the shell that is about to be executed. These variables are optionally exported with the request from the originating host.

3. The local host for the destination batch queue interprets the request shell specifications and its own shell strategy policy, then chooses the proper shell (e.g., */bin/csh, /bin/ksh, /bin/sh*, etc.). The batch request will be executed within this chosen shell.

4. The resource limits, determined when the request was queued, are applied to this new shell process.

5. The proper shell is executed and the shell script that defines the batch request is executed.

6. The spooled output files (*stderr* and *stdout*) are returned to their (possibly remote) machine destinations.

Batch Resource Limits

Both the batch request and the destination batch queue have defined resource limits associated with them. The resource limits of the destination queue take priority over those of the request.

However, if the system manager changes resource limits for a batch queue, the limits for all existing requests are *grandfathered*—that is, requests that have been queued are allowed to complete regardless of their limits, so that older requests are not affected by the new limits.

Batch Run Limits

In addition to resource limits, NQS imposes a set of *run limits* on batch queues. These run limits ensure that the local host does not get swamped running batch requests. The following run limits apply to batch queues:

- The *global batch run limit* places a ceiling on the maximum number of batch requests allowed to execute simultaneously on the local host. This limit applies to the local host, and applies to all requests on this host, regardless of the queue destination.
- The *queue run limit* places a ceiling on the maximum number of batch requests allowed to execute simultaneously in the containing batch queue.

In order for NQS to enforce these run limits, the configuration parameter *chk_runlimit* must be set to `1`.

Spawning a Batch Request

NQS uses the following process when spawning batch requests:

1. As a batch request completes execution, the entire set of batch queues is examined in decreasing order of batch queue priority. This imposes a *priority* order among all batch queues, and a *chronological* order on the requests within each queue.
2. Each batch request in the batch queue is spawned as it is examined until either the queue run limit or the global batch run limit is reached.
3. The next lower priority batch queue is examined, and batch requests are spawned using the above criteria.
4. The process continues until all batch queues have been examined or until the global batch run limit is reached.
5. The process restarts at the highest priority batch queue when the next batch request completes execution.

15
.5 | 200
15
50

Creating a Batch Queue

The NQS manager creates batch queues using the **qmgr create batch_queue** subcommand. Once a queue has been created, it then must be enabled and started using the **qmgr enable queue** and **start queue** subcommands. For example:

```
# qmgr
Mgr: create batch_queue q2-30 priority=0
Queue q2-30 created.
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
Mgr: enable queue q2-30
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
Mgr: start queue q2-30
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
Mgr: show queue
============================================
NQS Version: 2  BATCH  PIPE QUEUES on prefect
============================================
QUEUE NAME       STATUS   TOTAL   RUNNING  QUEUED   HELD TRANSITION
-----------------------------------------------------------------------------
q2-30            AVAILBL    0       0/1       0       0       0
```

About Queue Names

The previous example creates a queue named *q2-30.* Queue names should be descriptive and consistent. For example, since users typically need to know the number of nodes in the queue and the length of time their job request can run, consider a queue naming scheme that conveys this information. For example, a queue name like *q4-30* might indicate a 4-node queue that will allow a request to run for 30 minutes, while a name like *q30-4* might indicate a 30-node queue that will allow a request to run for four minutes. Queue names are limited to 15 characters.

Specifying Queue Priority

NQS queue priority refers to an inter-queue priority, and not a Paragon system execution priority. There are two ways to set a queue's priority:

- When creating the queue with the **qmgr create batch_queue** subcommand, as described above.
- By using the **qmgr set priority** subcommand.

The range for queue priority is 0 to 63, with larger numbers indicating higher priority.

Assigning a Node Group

Every queue must be associated with a *node group*. A node group is a collection of one or more *node sets*, which are rectangular collections of nodes. Node sets and node groups are defined during NQS setup and configuration, as described in Chapter 5. In the following example, the queue *q4-30* is associated with node group 0:

```
Mgr: set node_group =0 q4-30
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
Mgr:
```

If you don't explicitly assign a queue to a node group, the queue belongs to node group 0 by default. If this node group does not exist, NQS will not run jobs in that queue.

Specifying the Number of Nodes

After the queue is created with **create batch_queue**, use the **qmgr set per_request ncpus** subcommand to specify the number of nodes for a queue. (If you don't specify the number of nodes, the default is one node per queue.) The user can use the **qsub** command's **-lP** flag to specify a specific number of nodes, but the value specified must be less than or equal to the number of nodes allowed by the queue.

```
Mgr: set per_request ncpus =4 q4-30
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
Mgr:
```

Specifying Time Limits

After the queue is created with **create batch_queue**, use the **qmgr set per_request cpu_limit** subcommand to specify how long a queue will allow a job request to run. The user can use the **qsub** command's **-lT** flag to specify a specific length of time, but it cannot exceed the time limit imposed by the queue. If a job exceeds the per-request maximum, NQS sends a SIGTERM, followed by a SIGKILL after the length of time specified by the *grace_time* configuration parameter. NQS then sends mail to the user explaining why the job was killed. The following example would set a time limit of 30 minutes (specified in seconds) for the queue named *q4-30*:

```
Mgr: set per_request cpu_limit=(1800) q4-30
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
Mgr:
```

In the previous example, 30 minutes was specified in seconds. Refer to "Limits" on page 6-21 for the syntax for specifying time in the **qmgr** subcommands.

Specifying Run Limits

In addition to resource limits, NQS imposes a set of *run limits* on batch queues. A run limit is the number of requests that will run at the same time if compute nodes are available. Run limits can be implemented as follows:

- The *global batch run limit* places a ceiling on the maximum number of batch requests allowed to execute concurrently on the Paragon system. This global run limit ensures that the Paragon system does not get overloaded running batch requests. You specify this system run limit using the **qmgr set global batch_limit** subcommand.

- The *queue run limit* controls how many jobs in a queue can run concurrently if compute nodes are available. For example, a 2-node queue with a run limit of five will run five jobs concurrently if there are ten unused compute nodes available in the node group associated with the queue. You specify a queue run limit while using the **qmgr create batch_queue** subcommand to create a queue, or with the **qmgr set run_limit** subcommand if the queue has already been created.

- The *complex run limit* controls how many jobs in a complex (a grouping of queues) can run concurrently if compute nodes are available.You specify a queue run limit with the **qmgr set complex run_limit** subcommand after the complex has been created.

  Depending on your NQS configuration, queue and complex run limits allow you to set up the queues efficiently. For example, small queues might have higher run limits, allowing several small jobs to run concurrently. A large queue might have a run limit of `1`, which would limit that queue to one job at a time.

NOTE

To enable this feature, the configuration parameter *chk_runlimit* must be set to `1` in the */usr/spool/nqs/conf/sched_param* file.

The following example specifies a global run limit of 20:

```
Mgr: set global batch_limit=20
NQS manager[TCML_COMPLETE   ]: Transaction complete at local host.
Mgr: show parameters

    .
    .
    .
  Global batch run limit = 20
    .
    .
    .

Mgr:
```

In the next example, a queue named *q2-10* is created with a queue run limit of 5:

```
Mgr: create batch_queue q2-10 priority=2 run_limit=5
Queue q2-10 created.
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
Mgr: enable queue q2-10
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
Mgr: start queue q2-10
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
Mgr: show queue
============================================
NQS Version: 2  BATCH  PIPE QUEUES on prefect
============================================
QUEUE NAME       STATUS   TOTAL   RUNNING  QUEUED   HELD TRANSITION
-----------------------------------------------------------------------------
q2-30            AVAILBL    0      0/1        0       0       0
q2-10            AVAILBL    0      0/5        0       0       0

Mgr:
```

The `0/5` under the `RUNNING` head shows that no jobs are running, and that the queue has a run limit of `5`.

Effect of Changing Batch Resource Limits

Both the batch request and the destination batch queue have defined resource limits associated with them. The resource limits of the destination queue take priority over those of the request.

However, if the system manager changes resource limits for a batch queue, the limits for all existing requests are *grandfathered*—that is, requests that have been queued are allowed to complete regardless of their limits, so that older requests are not affected by the new limits.

Adding NQS Queues to the MACS System

If your site uses the Multi-User Accounting and Control System (MACS), you will need to supply the NQS queue names during MACS setup, or later by editing the files */usr/spool/macs/conf/macs.conf* and */usr/spool/macs//conf/nqstable*. Refer to the *Paragon™ System Multi-User Accounting and Control System Manual* for more information.

Restricting Queue Access

When created, a queue has unrestricted access by default. After the queue is created, you can use the **set no_access** subcommand to restrict all access to the queue, and then selectively give users or groups access to the queue. In order for a user to have access to a queue, it is only necessary for *one* of the following to be true:

- Queue access is unlimited (the default). The **set no_access** and **set unrestricted_access** subcommands toggle access to the queue.
- The queue is restricted, but the user's login group has access. Use the **add groups** subcommand to give the group access to the queue.
- The queue is restricted, but the user's account has access. Use the **add users** subcommand to give the user access to the queue.

An additional way of restricting access to a queue is to use the **create batch_queue** subcommand's *pipeonly* parameter when creating the queue. The *pipeonly* parameter forces all requests entering the queue to come from a pipe queue. This means a request cannot come directly from a user.

In the following example, access is restricted to the queue *q4-30*. Note that root retains access to the queue even though it is restricted. Finally, the user *archer* is given access to the queue.

```
Mgr: set no_access q4-30
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
Mgr: show long queue q4-30

=====================================================
NQS Version:2  BATCH     QUEUE:  q4-30.prefect      status:  AVAILBL
=====================================================
   .
   .
   .
ACCESS
        Groups:
        Users: root

Mgr: add user =archer q4-30
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
Mgr: show long queue q4-30

=====================================================
NQS Version:2  BATCH     QUEUE:  q4-30.prefect      status:  AVAILBL
=====================================================
   .
   .
   .
ACCESS
        Groups:
        Users: root     archer
```

Creating a Default Queue

It is sometimes useful to have a default queue available to handle job requests that don't specify a particular queue. Refer to "Specifying a Queue" on page 2-2 for more information on default queues. The following example specifies that *q4-30* (which must already exist) will be the default batch queue:

```
Mgr: set default batch_request queue q4-30
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
Mgr: show parameters

  .
  .
  .
  Default batch_request queue = q4-30
  .
  .
  .

Mgr:
```

To disable the default queue feature:

```
Mgr: set no_default batch_request queue
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
Mgr: show parameters

  .
  .
  .
  Default batch_request queue = NONE
  .
  .
  .

Mgr:
```

Defining Limits

NQS supports an extensive set of batch request resource quota limits, but is limited by the underlying operating system in the limits it supports on a given machine. (You can check the supported limits using either the **qlimit** command or the **qmgr show all** or **show limits_supported** subcommands.) As an NQS manager, you can change the limits that will be enforced when the **qsub** command is invoked. The following example shows the supported limits, then changes two of the resource limits.

```
% qmgr
Mgr: show limits_supported

  Core file size limit (-lc)
  Data segment size limit (-ld)
  Per-process permanent file size limit (-lf)
  Nice value (-ln)
  Per-request # of cpus limit (-lP)
  Stack segment size limit (-ls)
  Per-process cpu time limit (-lt)
  Per-request cpu time limit (-lT)
  Working set limit (-lw)

Mgr: set nice_value_limit = 10 q4-30
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
Mgr: show long queue q4-30

=====================================================
NQS Version:2  BATCH     QUEUE:  q4-30.prefect                status:  AVAILBL
=====================================================
                                        Priority: 0
ENTRIES:
        Total:   0      Running: 0
        Queued:  0      Held:    0      Transitiion: 0

RUN_LIMIT:
        Runlimit:  1

COMPLEX MEMBERSHIP:


RESOURCES:
  Per-proc core file size limit= UNLIMITED <DEFAULT>
  Per-process data size limit  = UNLIMITED <DEFAULT>
  Per-proc perm file size limit= 1 megabytes
  Per-proc execution nice value= 10
  Per-req number of cpus limit = 2
  Per-user # of requests limit = UNLIMITED <DEFAULT>
  Per-process stack size limit = UNLIMITED <DEFAULT>
  Per-process CPU time limit   = UNLIMITED <DEFAULT>
  Per-request CPU time limit   = 30.0
  Per-process working set limit= UNLIMITED <DEFAULT>

ACCESS
        Groups:
        Users: root     archer
```

Creating a Queue Complex

A *queue complex* is a collection of queues defined with the **qmgr create complex** subcommand. Once created, a queue complex can be assigned a run limit, which determines the maximum number of requests from the queue complex that are allowed to run concurrently. In the following example, the queues *q4-30* and *q2-30* are collected into the queue complex *cmplx-30* which has a run limit of 6. The **qcmplx** command then shows the queue complex status.

```
Mgr: show queue
============================================
NQS Version: 2  BATCH  PIPE QUEUES on prefect
============================================
QUEUE NAME      STATUS   TOTAL   RUNNING  QUEUED   HELD TRANSITION  NODE_GROUP
------------------------------------------------------------------------------
q4-30           AVAILBL    0      0/1       0        0       0        0
q2-30           AVAILBL    0      0/1       0        0       0        2
q2-60           AVAILBL    0      0/1       0        0       0        1


Mgr: create complex = (q4-30, q2-30) cmplx-30
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.
Mgr: set complex run_limit =6 cmplx-30
NQS manager[TCML_COMPLETE  ]: Transaction complete at local host.

Mgr: exit
# qcmplx
=========================================
 COMPLEXES for prefect
=========================================
COMPLEX: cmplx-30    run-limit = 6

#
```

Pipe Queues

Pipe queues are responsible for routing and delivering requests to other (possibly remote) queue destinations. Pipe queues are similar in concept to a pipeline, because they transport requests to other queue destinations. Pipe queues accept batch requests and then transport these requests to batch queues or other pipe queues.

A "pipe only" queue is a batch queue or a pipe queue that only accepts requests from a pipe queue, that is, not directly from a user. In a network environment, you might set up a batch queue to handle incoming requests from other workstations. Users on those remote workstations would submit their requests to a pipe queue which would have this batch queue as its destination.

Spawning a Pipe Request

Each pipe queue has an associated server that is spawned to service each request that passes through the queue. Because of the use of the word *server* (in the context of a *client/server* network connection), the spawned instance of a *pipe queue server* is called a *pipe client*.

When a request is routed through the pipe queue, the pipe server spawns a pipe client, and this client must then route and deliver the request to a destination. For each attempted remote destination, a network server process must be created on the remote host to act as agent for the pipe queue request.

The pipe client can choose any of the destinations from the destination set defined for this pipe queue to receive the pipe queue request. If a destination does not accept the pipe queue request, the pipe client can try another destination for the request. The pipe client only needs to find one destination that will accept the request in order to be successful.

It doesn't matter what order you specify destinations in the destination list. The machine-id's are first numerically searched lowest to highest, and then queues are searched alphabetically.

A pipe queue request can be rejected at a destination for a variety of reasons. Some reasons for rejection are related to the underlying request type, and are not due to a pipe queue problem. For example, a batch request may conflict with a run limit for the destination.

Rejection reasons more directly related to a pipe queue problem can be divided into *temporary* and *permanent* types of problems. A temporary problem could be that a destination has crashed and will likely be rebooted in the future. A permanent problem might be a lack of proper account authorization at the destination.

Pipe queues are both powerful and complex. The NQS manager can use the **qmgr** subcommands to configure the pipe queues to perform many tasks. The pipe client can be set up to examine a request and determine an appropriate destination queue. For example, large batch requests might be sent to a queue that only runs at night, while small or high priority batch requests could be sent to fast batch queues. Pipe queues can be used to ease network and hardware reliability problems. If a destination machine is down or otherwise inaccessible, the request can be requeued for delivery at some later time.

Pipe Queue Run Limits

Pipe queue run limits are similar to the set of run limits placed on batch queues. These run limits ensure that the local host does not get swamped running pipe requests. The following run limits apply to pipe queues:

- The *global pipe limit* places a ceiling on the maximum number of pipe requests that can execute simultaneously on the local host. This global limit applies to the local host, and applies to all requests on this host, regardless of the queue destination.

- The *queue run limit* places a ceiling on the maximum number of pipe requests allowed to execute simultaneously in the containing pipe queue.

In order for NQS to enforce these run limits, the configuration parameter *chk_runlimit* must be set to `1`.

Creating a Pipe Queue

You must have NQS manager privilege to create a pipe queue. There is very little difference between setting up a batch queue and setting up a pipe queue. Batch queues have more associated limits, while a pipe queue must have a destination associated with it. For example, to create a pipe queue, issue the following command:

```
Mgr:create pipe_queue p1 priority=10 server=(/usr/lib/nqs/pipeclient) dest=q1
Queue p1 created.
NQS manager[TCML_COMPLETE   ]: Transaction complete at local host.
```

With each pipe queue, there is an associated server that is spawned to handle each request released from the queue for routing and delivery. In this example, the server is */usr/lib/nqs/pipeclient.* Its instance is called a pipe client.

As with the batch queue, a pipe queue must be both enabled and started before it can be used.

Controlling Other NQS Functions

NQS manager privileges are required to set up new queue structures, to add new users to restricted queues, and to change privileges of existing users. Other tasks that NQS managers need to perform include making NQS accessible to new Paragon system users, setting up the Paragon mail services in a form recognizable to NQS, and setting up the various log files that NQS maintains. The remaining sections in this chapter describe these other functions.

Setting Up Mail Service

NQS uses the Paragon mail facilities extensively. In order for NQS to send mail to a remote destination, the mail software must be configured correctly. Refer to the *Paragon™ System Administrator's Guide* for specific details on setting up mail.

Account Mapping

Because a user name or user-ID may not be unique in the network, NQS supports a database that can map user-ID's from machine to machine. This database is created by a mapping program named **nmapmgr**. Chapter 6 provides a manual page for the **nmapmgr** program. The NQS network server on the remote machine performs the account mapping.

Recording/Reporting NQS Status

NQS uses both NQS mail and an NQS log file (*/usr/spool/nqs/log.d/logfile*) to post the status of system-wide error messages. NQS supports status reporting on request, queue, and limit queries. Limit queries are used to determine the set of batch request resource limits supported by NQS on the Paragon system. The **qstat** and **qlimit** commands use these status functions when reporting on previously queued requests and their containing queues.

Understanding NQS Scheduling

The NQS scheduler provides efficient, automated job scheduling for the Paragon system. NQS scheduling considers all batch requests and executes them in an order that is designed to make the best use of the compute nodes in the NQS batch area.

The scheduler includes a configuration file (*/usr/spool/nqs/conf/sched_param*) containing tunable parameters. These parameters allow NQS managers to tailor the NQS scheduling environment to best serve the site's computing needs.

Scheduling Attributes

Each batch request is characterized by the following attributes, which are used by the scheduler to make scheduling decisions:

- The job size, which is the number of nodes required to run the application.
- A time limit, which is the maximum amount of time needed to run the application.
- The job priority, a combination of *base* and *dynamic* priority:
  - *nqs_base_pri*—the base priority, which is the priority associated with the queue to which the job is submitted.
  - *nqs_dyn_pri*—the dynamic priority, which is determined by a base priority (*nqs_base_pri*) plus a priority aging factor:

    ```
    nqs_dyn_pri = nqs_base_pri + (age_factor * time)
    ```

    where:

    time is the time in hours the job has been waiting in queue.

    age_factor is the aging factor defined in the */usr/spool/nqs/conf/sched_param* NQS configuration file.
- The request type, either a regular batch request or a dedicated batch request. A dedicated batch request will run at the time and for the duration specified.

Scheduling Large Requests

The *block_pri* parameter makes it easier for large requests to get scheduled. When this feature is disabled by setting *block_pri* to *-1* (the default), NQS tends to run smaller requests instead of larger requests if they fit the node availability better.

You can ensure that larger requests run by setting *block_pri* to a value greater than *-1*. If a job reaches the top of the queue (by aging or by queuing in a high priority queue) and if there is no room in the base layer to schedule this job, and if the base priority of this job is greater than or equal to the *block_pri* value, the scheduling of jobs behind it is blocked until the top job can be scheduled.

Preventing Problem Requests From Re-Starting

If an NQS job crashes the system, you can use the **qstart** command, in conjunction with the *nosched* configuration parameter, to prevent the job from restarting and crashing the system again:

1. Boot the system to single-user mode.
2. Edit the *sched_param* file and change *nosched* to `1`.
3. Continue booting to multi-user mode.
4. Remove the problem job from the queue using the **qdel** command.
5. Edit the *sched_param* file and change *nosched* to `0`.
6. Issue the **qstart** command to pick up the new value for *nosched*, which will allow NQS to schedule requests immediately. (If you don't issue **qstart**, NQS will begin scheduling requests within 15 minutes of changing *nosched* to `0`).

The **qstart** command submits a dummy request to NQS, which causes NQS to read the configuration parameters in the *sched_param* file. Otherwise, NQS reads the *sched_param* file every 15 minutes or whenever an NQS request starts or finishes.

Understanding the Log Files

The NQS logfiles are located in the directory */usr/spool/nqs/log.d.* The logfiles contain ERROR, FATAL, INFO, WARNING, DEBUG and LOG messages. The current logfile is named *logfile.* By default, NQS saves the five most recent old logfiles. The newest old logfile is named *logfile.1*, the next most recent is *logfile.2*, and so on up to *logfile.5.* When a new logfile is created, the previous logfile is renamed to *logfile.1*, the number of each remaining old logfile is incremented, and the oldest logfile is removed. The system administrator can specify the number of old logfiles to keep by editing the value of the variable N_OLDLOGS in the script */sbin/init.d/nqs.*

The system administrator can specify an additional logfile using the **qmgr set log_file** subcommand. If an additional log file is specified, the DEBUG and LOG messages that would normally appear in the standard logfile are redirected to the logfile specified with the **qmgr set log_file** subcommand. The date and time each message was written appears at the beginning of the line. This file also receives copies of all other messages. The additional logfile can be located anywhere on the system, and is not backed up like the standard logfiles in the directory */usr/spool/nqs/log.d.*

If you use the **qmgr set log_file** subcommand and specify the main logfile, *(/usr/spool/nqs/log.d/logfile)*, the main logfile will then contain both sets of messages.

NQS Setup and Configuration 5

This chapter shows you how to configure NQS for your site. During NQS setup, some or all of the Paragon system's compute nodes are configured to be under the control of NQS. Once a node is under the control of NQS, it is not available for interactive use using the standard Paragon partition management commands (such as **mkpart**).

Once the compute nodes have been configured into an NQS batch area, the NQS manager can then set up queues to manage the batch area. To configure NQS, you need to do the following:

- Understand the NQS configuration parameters and how you use them to customize NQS. These configuration parameters are discussed in "Understanding the NQS Configuration Parameters" on page 5-2.

- Determine the NQS configuration that works best for your site. For example, some sites will want a large interactive partition during the day, and a large batch partition during the night, while other sites will use the entire compute partition to run batch jobs both day and night.

- As root, run the *nqs_setup* script and enter the values for the NQS configuration parameters that are appropriate to your system. The appropriate values for the NQS configuration parameters vary from system to system, although you can typically accept most defaults.

  You will, however, need to determine how you want to configure the NQS batch area. See the discussion "Batch Area Configuration Parameters" on page 5-8 and "Batch Area Scheduling Parameters" on page 5-9 for help in understanding the parameters that configure the batch area. See the discussion "Job Scheduling and Control Parameters" on page 5-13 to see these parameters used in sample configurations.

- When prompted by the *nqs_setup* script, use the **nmapmgr** utility to create the NQS mapping database, including the machine ID and machine name.

- Create the queues that will run in the NQS batch area (see Chapter 4 of this manual). If your site uses the Multi-User Accounting and Control System (MACS), you will need to supply the NQS queue names during MACS setup, or later by editing the files */usr/spool/macs/conf/macs.conf* and */usr/spool/macs/conf/nqstable*. Refer to the *Paragon™ System Multi-User Accounting and Control System Manual* for more information.

Understanding the NQS Configuration Parameters

NQS is configured by the configuration file */usr/spool/nqs/conf/sched_param* that contains tunable parameters. NQS managers can change these parameters to tailor NQS to their environment. Table 5-1 provides a functional list of all of the configuration parameters.

All configuration parameters take effect immediately after NQS startup. Some are also scanned each time NQS initiates a scheduling event (whenever someone submits a batch job, a batch job finishes, or every 15 minutes if there is no NQS activity). This allows the NQS manager to change some parameters without taking NQS down.

Use Table 5-1 and the configuration parameter descriptions that follow it to help determine the configuration parameter values appropriate for your system. Once you have run the *nqs_setup* script, you can always change the values for the NQS configuration parameters by editing the */usr/spool/nqs/conf/sched_param* file.

Table 5-1. Functional List of NQS Configuration Parameters (1 of 6)

Function	Parameter	Effect	Default
Batch Area Configuration	node_set[1]	A *node set* is a rectangular grouping of nodes. Node sets are used to create batch partitions of varying sizes and to group together nodes of similar attributes (groups of 16MB and 32MB nodes, for example). Node sets also play a role in switching between prime and non-prime time.	None. You must define at least one node set.
	node_group[1]	A *node group* is a collection of one or more node sets. Each NQS batch queue is associated with a single node group using the **qmgr set node_group** subcommand.	None. You must define at least one node group.

Table 5-1. Functional List of NQS Configuration Parameters (2 of 6)

Function	Parameter	Effect	Default
Batch Area Scheduling	prime_list[1] nprime_list[1]	The *prime_list* defines which node sets in a given node group are to be used during prime time. The *nprime_list* defines which node sets are used during non-prime time. A node set can be assigned to both the *prime_list* and *nprime_list* of a node group, meaning it will be used during both prime and non-prime time.	None. You must supply values for each.
	cur_open_name[1] open_p_name[1] open_np_name[1]	Define the partition names of the open prime and open non-prime partitions, and are used when creating a prime time/non-prime time scheduling scheme. If you specify one of these parameters, you must specify all three.	None. You must supply a value if your site uses a prime/non-prime time scheme.
	cur_open_epl[1] ncur_open_epl[1]	Define the effective priority level (EPL) for the open partition. *cur_open_epl* defines the EPL for the active open partition (named by *cur_open_name*). *ncur_open_epl* defines the EPL for the inactive open partition (named by *open_np_name* during prime time, and *open_p_name* during non-prime time). The EPL values must be between `0` and `10`, inclusive.	EPL of `3` for *cur_open_epl.* EPL of `1` for *ncur_open_epl.*
	prime_start prime_end	Define the beginning and ending times for prime time and non-prime time. Each parameter requires a list of seven input values separated by spaces, one for each day of the week, with the first element specifying the value for Sunday. Fractional hours are specified in decimal increments. For example, 4:30 PM is specified as `16.5`.	A *prime_start* value of `08` (8 a.m.) and a *prime_end* value of `18` (6 p.m.) for all 7 days.

Table 5-1. Functional List of NQS Configuration Parameters (3 of 6)

Function	Parameter	Effect	Default
Batch Area Scheduling (cont.)	prime_script nprime_script	Specify the full pathname of an executable file (typically a shell script) which is run at the beginning of prime time (*prime_script*) or non-prime time (*nprime_script*). The script is run after all other processing has been performed for the prime/non-prime switchover.	Blank (no scripts are run).
	nosched	Prevents NQS from scheduling requests. If *nosched* is set to `1`, NQS will not schedule any jobs. If set to `0` or omitted, NQS schedules jobs normally. If *nosched* is changed from `1` to `0` while NQS is running, NQS will begin scheduling jobs in 15 minutes, when **qstart** is issued, or when a batch request is submitted. The *nosched* parameter is typically used to prevent NQS from rescheduling a job that has crashed the system.	0
	do_wall	Defines whether a broadcast message will be sent to all logged in users during a prime/non-prime time switch. If *do_wall* is set to 0, no message is sent. If *do_wall* is set to 1, a message is sent. This parameter is ignored if *cur_open_name*, *open_p_name*, and *open_np_name* are not defined.	0
Job Scheduling and Control	macs_flag	Allows MACS to distinguish between NQS batch jobs and interactive jobs. If MACS is running and *macs_flag* is set to `0`, all jobs recorded in the MACS reports appear as interactive jobs. If MACS is running and *macs_flag* is set to `1`, NQS jobs are differentiated from interactive jobs. If MACS is not running and *macs_flag* is set to `1`, all NQS jobs will fail.	0
	wallclock_limits	Determines if the CPU time limit for a queue or request is based on node-hours or wall-clock hours. If set to `1`, the CPU limit is enforced as wall-clock hours; if set to `0`, the CPU limit is enforced as node-hours.This setting affects all queues.	0
	age_factor	Determines how much weight is given to the age of a job (the length of time a job has waited in the queue) when calculating the job's priority.	`0.1`

Table 5-1. Functional List of NQS Configuration Parameters (4 of 6)

Function	Parameter	Effect	Default
Job Scheduling and Control (cont.)	block_pri	Makes it easier for large jobs to get scheduled. When a job reaches the top of the queue and if there is no room in the base layer to schedule it, and if the base priority of this job is greater than or equal to the *block_pri* value, the scheduling of jobs behind it is blocked until the top job can be scheduled. When this feature is disabled with `-1`, NQS will tend to run smaller jobs instead of larger jobs if they fit the node availability better.	`-1`
	chk_runlimit	Defines whether NQS *run limits* are checked before starting a job. The run limit determines the maximum number of jobs that are allowed to run globally, in a queue, or in a complex. If set to `1`, the queue and complex run limits are enforced; if set to `0`, the run limits are not enforced.	`0`
	grace_time	The grace period, in seconds, after which a batch request is killed if it exceeds its run time limit.	`10`
	time_zone[1]	Defines the time zone used by NQS. The string format follows the Paragon OSF/1 system convention. For example, Pacific time is indicated as `PST8PDT`.	The value set for the *TZ* environment variable

Table 5-1. Functional List of NQS Configuration Parameters (5 of 6)

Function	Parameter	Effect	Default
Job Scheduling and Control (cont.)	np_overrun	Determines if NQS will allow non-prime time jobs to run over into prime time on nodes designated as non-prime time only. If *np_overrun* is set to `0`, NQS will allow non-prime time jobs to run over into prime time. If *np_overrun* is set to `1`, NQS will not schedule jobs if the requested time plus the current time will overrun into prime time.	`0`
	tsched_pri	Currently unused.	`20`
	timesh_pri	Currently unused.	`10`
	timeshare	Currently unused.	`0` (leave default)
	use_login	Determines how the user's shell is spawned. If *use_login* is set to `1`, NQS spawns the user's shell as a login shell (*-sh*, *-csh*, or *-ksh*), causing the login shell to read its *.profile* or *.cshrc* and *.login* files as it starts up. If *use_login* is set to `0` or omitted, the user's shell is spawned as a non-login shell.	`0`
	rollin_quan	The rollin quantum is the time, in seconds, a batch job can execute before being rolled out into a time-sharing scheme. This value must be greater than the value set by MIN_RQ_ALLOWED= in the */etc/nx/allocator.config* file.	`600` seconds. This default is incompatible with the allocator default rollin quantum (one hour) set in the *allocator.config* file. This setting should be set to a value greater than that set for the allocator.

Table 5-1. Functional List of NQS Configuration Parameters (6 of 6)

Function	Parameter	Effect	Default
Dynamic Job Priority [2]	node_base	In the calculation to determine dynamic priority, used to determine the node priority of a job.	`0.0`
	node_weight	In the calculation to determine dynamic priority, determines how much weight is given to the node priority when calculating the priority of a job.	`1.023`
	time_base	In the calculation to determine dynamic priority, used to determine the time priority of a job.	`1.0027`
	time_weight	In the calculation to determine dynamic priority, determines how much weight is given to the time priority when calculating the priority of a job.	`0.0`

1. If you change the values for these parameters, NQS must be restarted before the new values take effect.
2. These parameters are not included in the NQS setup script unless the parameters have previousely been added to the *sched_param* file by the system administrator.

Batch Area Configuration Parameters

The following configuration parameters define the NQS batch area. Refer to "Sample NQS Node Configurations" on page 5-20 to see these configuration parameters used in sample configurations.

node_set

A *node set* is a rectangular grouping of nodes. Node sets are the "building blocks" of the NQS batch area. Node sets are used to create batch partitions of varying sizes and to group together nodes of similar attributes (groups of 16MB and 32MB nodes, for example). Node sets also play a role in switching between prime and non-prime time. Node sets are combined into *node groups* with the *node_group* configuration parameter.

You specify the *starting node* in terms of a logical node number of the parent partition. The format for node sets is as follows:

node_set*number* **:** **.***partition_name heightxwidth***:***starting node*

The node set *number* must be in the range 0 to 63. For example:

```
node_set0 : .compute 3x2:2
```

Keep the following in mind when specifying node sets:

1. Node sets cannot overlap.

2. A node set can be shared by more than one node group.

3. All node sets belonging to a node group must have a common root partition. This is because the NQS daemon may have to create partitions for jobs that span multiple node sets.

4. We recommend that all nodes in a node set have the same physical attributes (the same node type and memory size).

node_group

A *node group* is a collection of one or more node sets. Each NQS batch queue is associated with a single node group using the **qmgr set node_group** subcommand. A node group can be associated with more than one queue. The format for node groups is as follows:

node_group*number* **:** *node_set_number . . .*

The node group *number* must be in the range 0 to 63. For example:

```
node_group1 : 1 2
```

Batch Area Scheduling Parameters

The following scheduling parameters schedule nodes in the *.compute* partition for both NQS batch jobs and interactive jobs, and allow these partitions to change between prime and non-prime times. Refer to "Sample NQS Node Configurations" on page 5-20 to see these configuration parameters used in sample configurations.

prime_list
nprime_list

The *prime_list* defines which node sets in a given node group will run NQS batch requests during prime time. The *nprime_list* defines which node sets in a given node group will run NQS batch requests during non-prime time. A node set can be assigned to both the *prime_list* and *nprime_list* of a node group, meaning it will be used during both prime and non-prime time. If a node set is assigned to only the *nprime_list*, it will be used only during non-prime time.

The *prime_list* must be a subset of the *nprime_list*. If a node set is assigned to the *prime_list* but not the *nprime_list*, NQS will add it to the *nprime_list* and write a warning message to the NQS logfile.

The format is as follows:

```
prime_listnode_group_number : node_set_number...
nprime_listnode_group_number : node_set_number...
```

The following example shows the *prime_list* and *nprime_list* for *node_group0* and *node_group1*:

```
prime_list0 : 0
nprime_list0 : 0
prime_list1 : 2
nprime_list1 : 1 2
```

cur_open_name
open_np_name
open_p_name

These parameters define the partition names of the open prime and open non-prime partitions, and are used when creating a prime time/non-prime time scheduling scheme. These partitions must first be created with the **mkpart** command (refer to Figure 5-3 and Figure 5-5). Partition names can be absolute or relative. If you specify one of these parameters, you must specify all three.

During prime time, the partition named by *open_p_name* is active and the partition named by *open_np_name* is inactive. During non-prime time, the partition named by *open_p_name* is inactive and the partition named by *open_np_name* is active.

When the time changes from prime time to non-prime time, NQS renames the *cur_open_name* partition to the name defined by *open_p_name*, and then renames the partition defined by *open_np_name* to *cur_open_name*. The reverse is done when the time changes from non-prime time to prime time. Thus, the active open partition is always named by the value supplied by *cur_open_name*. This is done so that users always see a constant open partition name in both operating modes.

For example:

```
mkpart -nd 16x3:9 open_p
mkpart -nd 16x1:11 open_np
```

The entries in the *sched_param* file would be as follows:

```
open_p_name : open_p
open_np_name : open_np
cur_open_name : OPEN
```

When NQS renames partitions at the beginning and end of prime time, the active OPEN partition (named by *cur_open_name*) will always have the same permissions, and the inactive open partition (*open_np_name* during prime time and *open_p_name* during non-prime time) will always have the same permissions.

Keep the following in mind as you design a dynamic prime/non-prime time scheme for your site:

- The compute partition must be gang-scheduled with at least two layers.
- Both the *open_p* and *open_np* partitions must have a nonzero size. There is no way for the Paragon system to be 100 percent batch in a prime/non-prime time scheduling scheme, but it can be 100 percent interactive.
- The inactive partition doesn't go away. It is just "buried" under higher-priority partitions.

  For instance, using the example above suppose there is a request running in the partition OPEN. At the end of prime time, the partition OPEN is renamed to *open_p* and has its EPL lowered, then the partition *open_np* is renamed to OPEN and has its EPL raised. Any NQS requests that overlap the *open_p* partition and any jobs running in the non-prime time interactive partition (now called OPEN, formerly called *open_np*) will have a higher effective priority. But if there are no batch requests and no non-prime time interactive jobs, the job running in the prime time interactive partition (now called *open_p*, formerly called OPEN) continues to run.

cur_open_epl ncur_open_epl

These parameters define the effective priority level (EPL) for the open partition. The *cur_open_epl* parameter defines the EPL for the active open partition (named by *cur_open_name*). The *ncur_open_epl* parameter defines the EPL for the inactive open partition (named by *open_np_name* during prime time, and *open_p_name* during non-prime time).

(During prime time, the partition named by *open_p_name* is active and the partition named by *open_np_name* is inactive; during non-prime time, the partition named by *open_p_name* is inactive and the partition named by *open_np_name* is active.)

The EPL of a partition sets a maximum value for the allocator priority of the parallel applications in the partition. An application's allocator priority determines whether or not the application runs if the application overlaps with nodes that are in use by other applications. See the *Paragon™ User's Guide* for more information on EPLs.

The EPL values must be between `0` and `10`, inclusive. The default EPL for *cur_open_epl* is 3; the default EPL for *ncur_open_epl* is 1.

prime_start prime_end

These parameters define the beginning and ending times for prime time and non-prime time. Each parameter requires a list of seven input values separated by spaces, one for each day of the week, with the first element specifying the value for Sunday. Fractional hours are specified in decimal increments. The non-prime time period is between the end and start of the prime time period. The default configuration uses a *prime_start* value of `08` and a *prime_end* value of `18` for all seven days.

In the following example, prime time starts at 8:15 and ends at 6:30:

```
prime_start : 08.25 08.25 08.25 08.25 08.25 08.25 08.25
prime_end : 18.50 18.50 18.50 18.50 18.50 18.50 18.50
```

prime_script nprime_script

These parameters specify the full pathname of an executable file (typically a shell script) which is run at the beginning of prime time (*prime_script*) or non-prime time (*nprime_script*). The script is run after all other processing has been performed for the prime/non-prime switchover. The specified file is executed as *root*. If it is a shell script, the script needs to be properly protected from unauthorized modification.

Use these parameters if you want to do site-specific things during the prime/non-prime time switchover. For example, you could use scripts to:

- Change the permissions on the *open_p* and *open_np* partitions so that only certain users can run interactive jobs during non-prime time.
- Change the */etc/passwd* file so that only certain users can log in during non-prime time.
- Kill all interactive jobs at the beginning of non-prime time.

These tasks could also be accomplished with a **cron** job, but using the *prime_script* and *nprime_script* parameters ensures that these tasks are strictly coordinated with NQS. The default value for both parameters is blank, meaning that no script is executed.

NOTE

Since the NQS daemon waits for the script to complete, and NQS commands cannot be processed while the script is running, any NQS commands within the script will cause the daemon to hang (because the daemon is waiting for the script, and the script is waiting for the daemon). To prevent this problem, put any NQS commands in the script in the background by adding an ampersand (&) character to the end of the command.

nosched

Prevents NQS from scheduling requests. If *nosched* is set to `1`, NQS will not schedule any jobs. If set to `0` or omitted, NQS schedules jobs normally. If *nosched* is changed from `1` to `0` while NQS is running, NQS will begin scheduling jobs in 15 minutes. The *nosched* parameter is typically used to prevent NQS from rescheduling a job that has crashed the system. The default is `0`.

do_wall

This parameter defines whether a broadcast message will be sent to all logged-in users during a prime/non-prime time switch. If *do_wall* is set to 0, no broadcast message will be sent. If *do_wall* is set to 1, a broadcast message is sent. The default is 0. This parameter is ignored if *cur_open_name*, *open_p_name*, and *open_np_name* are not defined.

Job Scheduling and Control Parameters

The following discussions describe the NQS configuration parameters used for job scheduling and control. During NQS setup, you can often accept the default value for these parameters and then change them later as you fine tune your system. Use the sample configuration in Figure 5-4 on page 5-27 as a guide to using these parameters.

macs_flag

This parameter allows MACS to distinguish between NQS batch jobs and interactive jobs.

If MACS is running and *macs_flag* is set to `0`, all jobs recorded in the MACS reports appear as interactive jobs. If MACS is running and *macs_flag* is set to `1`, NQS jobs are differentiated from interactive jobs.

If MACS is not running and *macs_flag* is set to `1`, all NQS jobs will fail. The default value is `0`.

NOTE

If the **-MACS** flag is specified in the */sbin/init.d/allocator* script, or **USE_MACS=1** is specified in the *allocator.config* file, every job submitted to the Paragon system is checked for access permission, regardless of whether the job is submitted as a NQS batch request or as an interactive application.

wallclock_limits

Determines if the CPU time limit for a queue or request is based on node-hours or wall-clock hours. If set to `1`, the CPU limit is enforced as a wall-clock limit; if set to `0`, the CPU limit is enforced as a node-hour limit.This setting affects all queues. The default value is `0`.

The *wallclock_limits* configuration parameter affects the CPU limits set with the **qmgr set per_request cpu_limit** subcommand and the CPU limits specified by the user with the **qsub -lT** flag.

In a node-hour configuration (*wallclock_limits* set to `0`), if the request uses fewer nodes than those reserved by the queue, the node-hours for that queue are redistributed among the nodes actually used. This means that the request could run longer than the per-request time limit for the queue. For example, an application can use fewer nodes than those reserved for the queue if the application uses the *NX_DFLT_SIZE* environment variable and specifies fewer nodes, or is invoked with the **-sz** switch and specifies fewer nodes.

NOTE

If your site uses gang scheduling, the *wallclock_limits* setting should be 0. This will specify CPU limits based on node-hours, which will allow jobs that were rolled out for a time to finish running.

np_overrun

The *np_overrun* parameter determines if NQS will allow non-prime time jobs to run over into prime time on nodes designated as non-prime time only.

If *np_overrun* is set to `0`, NQS will allow non-prime time jobs to run over into prime time. If *np_overrun* is set to `1`, NQS will not schedule jobs if the requested time plus the current time will overrun into prime time. The default value is `0`.

time_zone

This parameter defines the time zone used by NQS. The string format follows the Paragon OSF/1 system convention. For example, Pacific Daylight Saving time is indicated as `PST8PDT`. The default configuration uses a *time_zone* value equal to the value set for the *TZ* environment variable.

age_factor

This *age-factor* value is a floating point value used to calculate the dynamic priority of a job. The dynamic priority is calculated using the following equation:

```
    nqs_dyn_pri = (nqs_base_pri + (age_factor * time))
```

where *nqs_base_pri* is the priority of the queue to which the job was submitted and *time* (a floating point value) is the time in hours that the job has been waiting in the queue. The equation is carried out in floating point arithmetic. The longer a job remains in a queue, the higher the *nqs_dyn_pri* becomes. The default configuration uses an *age-factor* of `0.1`.

block_pri

This parameter makes it easier for large jobs to get scheduled. When a job reaches the top of the queue (by aging or by queuing in a high priority queue) and if there is no room in the base layer to schedule this job, and if the base priority of this job is greater than or equal to the *block_pri* value, the scheduling of jobs behind it is blocked until the top job can be scheduled.

When this feature is disabled with `-1`, NQS will tend to run smaller jobs instead of larger jobs if they fit the node availability better.

chk_runlimit

This parameter defines whether NQS run limits should be checked before starting a job. In NQS, the *run_limit* determines the maximum number of jobs that are allowed to run globally, in a queue, or in a complex. With *chk_runlimit* set to `1`, the queue and complex run limits are enforced. If it is set to `0`, the run limits are not enforced. The default configuration uses a *chk_runlimit* value of `0`.

grace_time

This parameter is the grace period, in seconds, after which a batch job is killed if its run time exceeds the job's CPU run time limit. When the CPU time limit expires, a warning signal (SIGTERM) is sent to all processes of the job. After a grace period of *grace_time* seconds, the job is killed with a SIGKILL signal. The default configuration uses a *grace_time* value of 10 seconds.

timeshare

Unused. If the *timeshare* value is set to `1`, up to two NQS jobs can time share a set of nodes. If the *timeshare* value is set to `0`, no time sharing is allowed. (Setting the *timeshare* value to `0` does not prevent batch and interactive jobs from overlapping during a prime/non-prime time switch.) The default value for this parameter is `0` and should not be changed.

timesh_pri

Unused. This parameter defines the minimum dynamic priority required for a job to trigger time sharing with another job. This allows you to reserve some node time for high priority jobs. NQS will fill a maximum of two layers of the 2-D mesh. The first layer is filled without considering the minimum dynamic priority requirement so that the nodes are not idle. For a job to be allowed in the second layer, its dynamic priority must be greater than or equal to *timesh_pri*. If *timesh_pri* is set to zero, all jobs are allowed in the second layer.

The default configuration uses a *timesh_pri* value of `10`. The value of *timesh_pri* is not used if the *timeshare* value is set to `0`.

rollin_quan

This parameter defines the rollin quantum for a batch job's partition. The rollin quantum is the time, in seconds, a parallel job can execute before being rolled out in a time-share gang scheduling scheme. The default configuration uses a *rollin_quan* of `600` seconds. The value of *rollin_quan* is not used if the *timeshare* value is set to `0`. This value must be greater than the value set by *MIN_RQ_ALLOWED=* in the */etc/nx/allocator.config* file.

use_login

Determines *how* the user's shell is spawned (the **qmgr** subcommand **set shell_strategy** determines *which* shell is actually spawned).

If *use_login* is set to 1, NQS spawns the user's shell as a login shell, causing the shell to read its *.profile* or *.login* file as it starts up. If *use_login* is set to 0 or omitted, the user's shell is spawned as a non-login shell. The default is 0.

Since an NQS job doesn't have an attached terminal, some commands in the user's dot files can cause error messages if *use_login* is set to 1. These error messages appear in the NQS *jobname.ereqID* file.

For example, the default *.login* file created for a user by the **adduser** command includes the following lines:

```
stty crt new
tset -I -Q
```

If *use_login* is set to 1 and the shell determined by the current shell strategy is the C shell, these lines cause the following error messages:

```
tcgetattr: Operation not supported on socket
stty: Not a terminal
```

To avoid this problem, edit the *.login* file so that it does not execute these lines if it is being run under NQS. One way to do this is to test for the existence of the environment variable QSUB_REQID. If the shell is being run under NQS, this variable is set to the NQS request ID and the value of $?QSUB_REQID is 1; if the shell is being run normally, this variable does not exist and the value of $?QSUB_REQID is 0.

For example, if you change the above lines in your *.login* file as follows:

```
if ( $?QSUB_REQID == 0 ) then
   # interactive shell, not being run under NQS
   stty crt new
   tset -I -Q
endif
```

These two commands will now not be executed when the shell is run under NQS.

The following messages always appear if the selected shell is the C-shell and *use_login* is set to 1:

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

The *Warning...* message appears at the beginning of the *jobname.ereqID* file; the *logout* message appears at the end.

tsched_pri

Unused. This parameter defines the minimum queue priority (*nqs_base_pri*) required for a batch queue to become a dedicated (TSCHED) batch queue. You submit a job to a dedicated batch queue with the **qsub -a** flag. If you do not use the **-a** flag, the job is treated as a normal batch job. The default configuration uses a value of `20`.

Dynamic Job Control Parameters

By default, NQS schedules queues in a first-in, first-out scheme. If two queues need to use some of the same nodes, the job with the higher priority gets to run first. Typically, the priority of a job is determined only by the base priority of the queue and how long the job has been waiting in the queue. The NQS manager can, however, use the dynamic job control parameters to change the priority of a request based on the number of nodes a request uses or the duration of the request set by the per-request CPU time limit, allowing smaller or shorter jobs to run first.

By default, these parameters are not placed in the *sched_param* file by the *nqs_setup* script. Once these parameters are added to the *sched_param* file, however, they will appear during NQS setup.

Also, keep in mind that the default parameter values turn the feature off completely, so that the NQS manager can ignore these parameters if this feature is not needed.

NQS uses *priorities* to determine the order in which to run jobs. Jobs are sorted in queues by priority order, and if jobs from two different queues are set to run on the same nodes, the job with the higher priority gets to run first. A job's priority is determined by the following equation:

$$\begin{aligned}\text{priority} \quad = \quad & (\texttt{node\_weight} * \texttt{node\_base}^{-\mathit{node\_limit}}) + \\ & (\texttt{time\_weight} * \texttt{time\_base}^{-\mathit{time\_limit}}) + \\ & (\texttt{age\_factor} * \mathit{time\_in\_queue}) + \\ & \mathit{base\_priority}\end{aligned}$$

The priority calculation has four parts: the node part (first line of the equation), time part (second line), age part (third line), and base part (fourth line). These four parts are added together to give the job's priority.

node_base
node_weight

The node part of the priority is determined by the number of nodes the job requires.

node_limit The per-request number of nodes for the job. This is set by the user with the **qsub -lP** switch when submitting the job. If the user did not specify the number of nodes, the node limit is set by the NQS manager with the **qmgr set per_request ncpus** subcommand.

`node_base` This configuration parameter is used in calculating the "node priority" of a job—that is, the part of a job's priority that is based on the number of nodes in the job. The node priority of a job is calculated as $\texttt{node\_base}^{-node\_limit}$, which is always a value between 0 and 1; the larger the *node_limit*, the smaller the node priority.

This calculation means that the larger the `node_base`, the faster the node priority decreases as the size of the job increases. For example, the following table shows the approximate node priorities of jobs with three different *node_limit*s under three different `node_bases`:

Table 5-2. node_base Applied to Job Size

node_base	1 Node	10 Nodes	100 Nodes
1.01	0.99	0.91	0.37
1.023 (default)	0.97	0.79	0.10
1.05	0.95	0.61	0.01

`node_weight` This configuration parameter determines how much weight is given to the node priority when calculating the priority of a job. The node priority is multiplied by the `node_weight`, so the larger this value, the more important the number of nodes is. With the default value of 0, the number of nodes in a job does not affect its priority at all.

time_base time_weight

The time part of the priority is determined by the job's CPU time limit.

time_limit	The per-request CPU time limit for the job, in minutes. This is set by the user with the **qsub -lT** switch when submitting the job. If the user did not specify a time limit, the time limit is set by the NQS manager with the **qmgr set per_request cpu_limit** subcommand. (An unlimited time limit is treated as an infinite value.)
`time_base`	This configuration parameter is used in calculating the "time priority" of a job—that is, the part of a job's priority that is based on the job's CPU time limit. The time priority of a job is calculated as $\texttt{time\_base}^{-time\_limit}$, which is always a value between 0 and 1; the larger the time_limit, the smaller the time priority.

This calculation means that the larger the `time_base`, the faster the time priority decreases as the time limit of the job increases. For example, the following table shows the approximate time priorities of jobs with three different *time_limits* under three different `time_bases`:

Table 5-3. time_base Applied to Job Length

time_base	60 Minutes	240 Minutes	840 Minutes
1.001	0.94	0.79	0.43
1.0027 (default)	0.85	0.52	0.10
1.005	0.74	0.30	0.02

`time_weight`	This configuration parameter determines how much weight is given to the time priority when calculating the priority of a job. The time priority is multiplied by the `time_weight`, so the larger this value, the more important the time limit is. With the default value of 0, the time limit of a job does not affect its priority at all.

The age part of the priority is determined by how long the job has waited in the queue.

time_in_queue	The time, in hours, since the job was submitted to the queue.
`age_factor`	This configuration parameter determines how much weight is given to the *time_in_queue* when calculating the priority of a job. The *time_in_queue* is multiplied by the `age_factor`, so the larger this value, the more important the age of the job is. The default `age_factor` is 0.1.

The base part of the priority is determined by the queue's base priority.

base_priority	The base priority of the queue, which is set by the NQS manager when the queue is created and can be changed with the **qmgr set priority** subcommand.

NOTE

Once a request is running, it cannot be unscheduled to make way for jobs with higher priorities.

Sample NQS Node Configurations

NQS node configurations can be simple or complex, depending on your site computing strategy. Similarly, the level of understanding required to implement these strategies can be correspondingly simple or complex. At a minimum, before configuring NQS you should have a good understanding of the NQS configuration parameters and be familiar with node sets (discussed on page 5-8) and node groups (discussed on page 5-8).

The discussions that follow present some typical NQS configurations. Use these sample configurations as a guide in determining the NQS configuration for your site. The configurations presented include the following:

- The entire *.compute* partition is a NQS batch area, handling batch requests 24 hours a day.
- Part of the *.compute* partition is an NQS batch area, and the rest is left open for interactive jobs. The batch and interactive areas are static (do not change in size).
- Part of the *.compute* partition is an NQS batch area, and the rest is left open for interactive jobs. The batch and interactive areas are dynamic (change in size) during the prime/non-prime time switch.
- Part of the *.compute* partition is an NQS batch area, and the rest is left open for interactive jobs. The batch/interactive areas change in size during prime/non-prime time changes. Certain nodes are grouped together into node sets, which are then associated with specific queues.

NQS-Only Configuration

An NQS-only configuration is fairly simple, and involves editing the *sched_param* file or supplying values for certain configuration parameters during NQS setup. The basic steps are as follows.

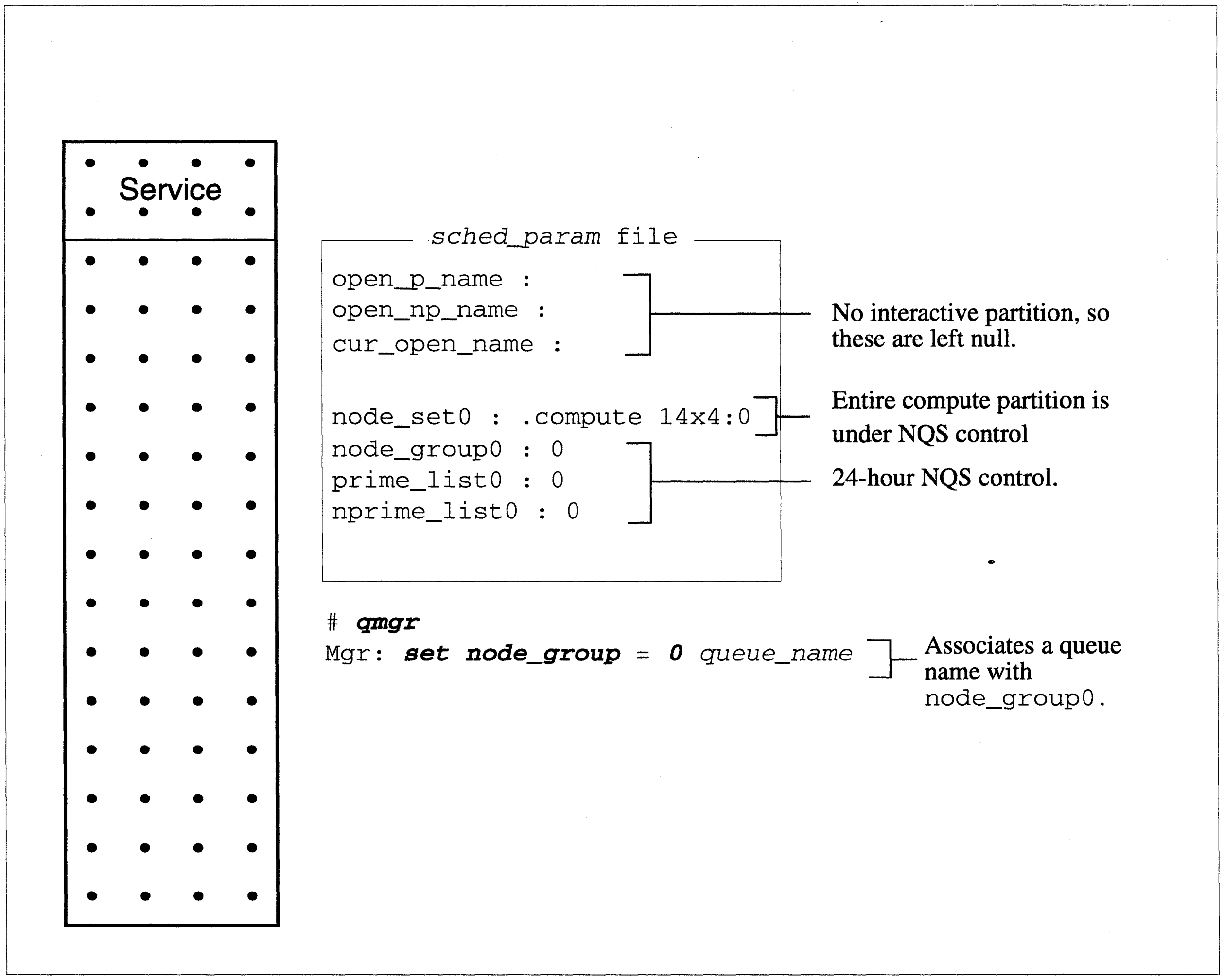
1. Use the *node_set* parameter to create one or more node sets that encompass the entire *.compute* partition. More than one node set is required if the *.compute* partition isn't rectangular.

2. Use the *node_group* parameter to make that node set (or sets) part of a node group.

3. Include all of the node sets of *node_group0* in both the *prime_list* and *nprime_list* parameters, which means that the node sets in *node_group0* encompass all of *.compute* and will run NQS batch requests 24 hours a day.

4. Once the configuration is complete, use the **qmgr set node_group** subcommand to associate the NQS queues with *node_group0.*

Figure 5-1 shows an example of an NQS-only configuration.


Service
No interactive partition, so these are left null.
Entire compute partition is under NQS control
24-hour NQS control.
Associates a queue name with node_group0.

```
sched_param file
open_p_name :
open_np_name :
cur_open_name :

node_set0 : .compute 14x4:0
node_group0 : 0
prime_list0 : 0
nprime_list0 : 0
```

```
# qmgr
Mgr: set node_group = 0 queue_name
```

Figure 5-1. 24-Hour NQS-Only Configuration

Static NQS/Interactive Configuration

A static NQS/interactive configuration involves using the **mkpart** command to reserve nodes for the NQS batch area, and editing the *sched_param* file or supplying values for certain configuration parameters during NQS setup. This configuration is static because the sizes of the batch area and the interactive partition do not change. The basic steps are as follows.

1. Use the **mkpart** command to reserve nodes for the NQS batch area.

2. Use the *node_set* parameter to create one or more node set that encompasses the portion of *.compute* specified with **mkpart**. More than one node set is required if the *.compute* partition isn't rectangular

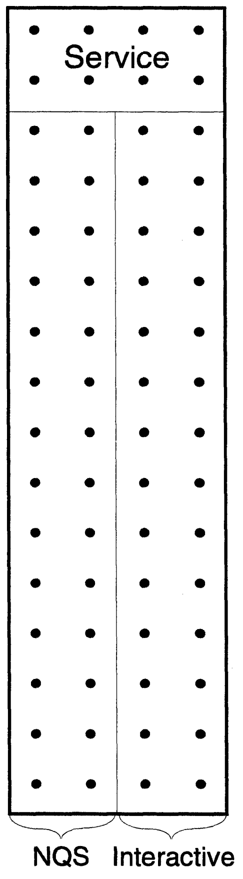
3. Use the *node_group* parameter to make that node set (or sets) part of a node group.

4. Include all of the node sets of *node_group0* in both the *prime_list* and *nprime_list* parameters, which means that the node sets in *node_group0* will run NQS batch requests 24 hours a day.

5. Once the configuration is complete, use the **qmgr set node_group** subcommand to associate the NQS queues with *node_group0.*

The remainder of the nodes are free to run interactive applications.

Figure 5-2 shows an example of a static NQS/interactive configuration.



```
# mkpart -mod 744 -nd 14x2:0 nqs
```

Reserves nodes for the NQS batch area. The remaining nodes are available for interactive use.

```
———— sched_param file ————
open_p_name :
open_np_name :
cur_open_name :

node_set0 : .compute.nqs 14x2:0
node_group0 : 0
prime_list0 : 0
nprime_list0 : 0
```

No prime/non-prime time switch, so these are left null.

Nodes under NQS control.

24-hour NQS control.

```
# qmgr
Mgr: set node_group = 0 queue_name
```

Associates a queue name with `node_group0`.

Use the **qstat** command to list jobs in the NQS batch area.
Use the **pspart** command to list jobs in the interactive partition.

Figure 5-2. Part NQS, Part Interactive Configuration with no Overlap

Dynamic NQS/Interactive Configuration

The NQS/interactive configuration shown in Figure 5-3 is considered *dynamic* because the sizes of the NQS batch area and the interactive partition change during the prime time/non-prime time switch.

This configuration uses two **mkpart** commands to reserve nodes for the NQS batch area. Since this is a prime time/non-prime time configuration, the NQS batch area is composed of two partitions that overlap. This overlap is configured into a node set and is used as part of the prime/non-prime time switch over.

The basic steps for this type of configuration are as follows.

1. Use two **mkpart** commands to reserve nodes for the interactive partition, and to divide the interactive area into two overlapping partitions (named *open_p* and *open_np* in this example).

2. Supply values for the *open_p_name* and *open_np_name* configuration parameters. These must be the partition names created with **mkpart**. Supply a value for *cur_open_name*, which becomes the partition name for interactive users.

3. Use two *node_set* parameters to create node sets that encompass the nodes to be used for batch requests during prime and non-prime time.

4. Use the *node_group* parameter to make both node sets part of a single node group.

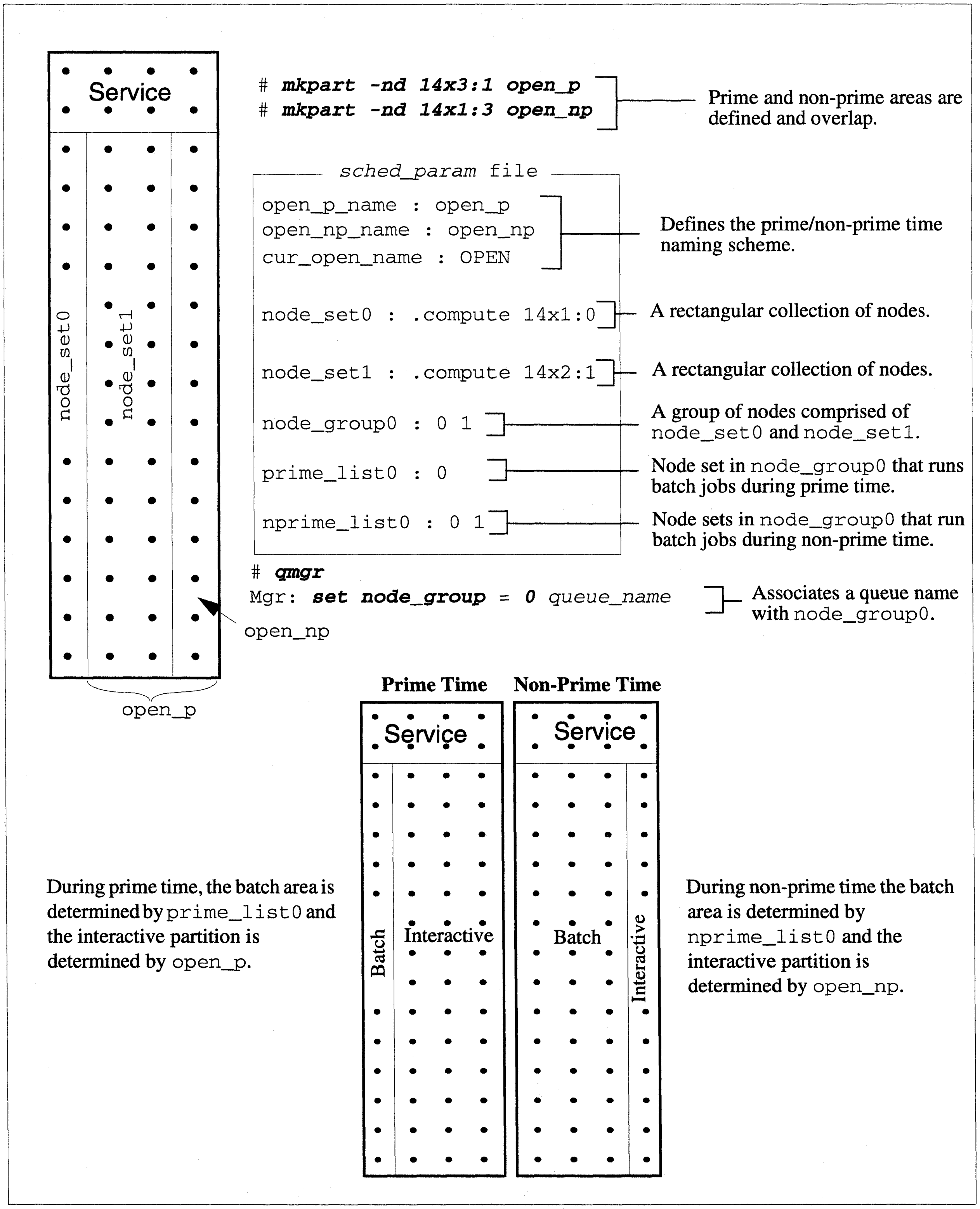
5. Include *node_set0* of *node_group0* in both the *prime_list* and *nprime_list* parameters. Include *node_set1* of *node_group0* in the *nprime_list* parameter only.

6. Once the configuration is complete, use the **qmgr set node_group** subcommand to associate the NQS queues with *node_group0.*

During prime time, the location of the NQS batch area is determined by the node sets belonging to *prime_list0* and the interactive partition is determined by *open_p*. During non-prime time, the location of the NQS batch area is determined by the node sets belonging to *nprime_list0* and the interactive partition is determined by *open_np.*

During prime time, 42 nodes are available for interactive use and 14 nodes are reserved for NQS batch jobs. During non-prime time, 42 nodes are reserved for batch jobs and 14 are available for interactive jobs.

Figure 5-4 on page 5-27 shows an example of this configuration as it would appear in a complete *sched_param* file.


```
# mkpart -nd 14x3:1 open_p
# mkpart -nd 14x1:3 open_np
```
Prime and non-prime areas are defined and overlap.

sched_param file
```
open_p_name : open_p
open_np_name : open_np
cur_open_name : OPEN

node_set0 : .compute 14x1:0

node_set1 : .compute 14x2:1

node_group0 : 0 1

prime_list0 : 0

nprime_list0 : 0 1
```
Defines the prime/non-prime time naming scheme.

A rectangular collection of nodes.

A rectangular collection of nodes.

A group of nodes comprised of node_set0 and node_set1.

Node set in node_group0 that runs batch jobs during prime time.

Node sets in node_group0 that run batch jobs during non-prime time.

```
# qmgr
Mgr: set node_group = 0 queue_name
```
Associates a queue name with node_group0.

During prime time, the batch area is determined by prime_list0 and the interactive partition is determined by open_p.

During non-prime time the batch area is determined by nprime_list0 and the interactive partition is determined by open_np.

Figure 5-3. Basic Node Configuration for Prime/Non-Prime Time Scheduling

Sample Batch/Interactive Configuration

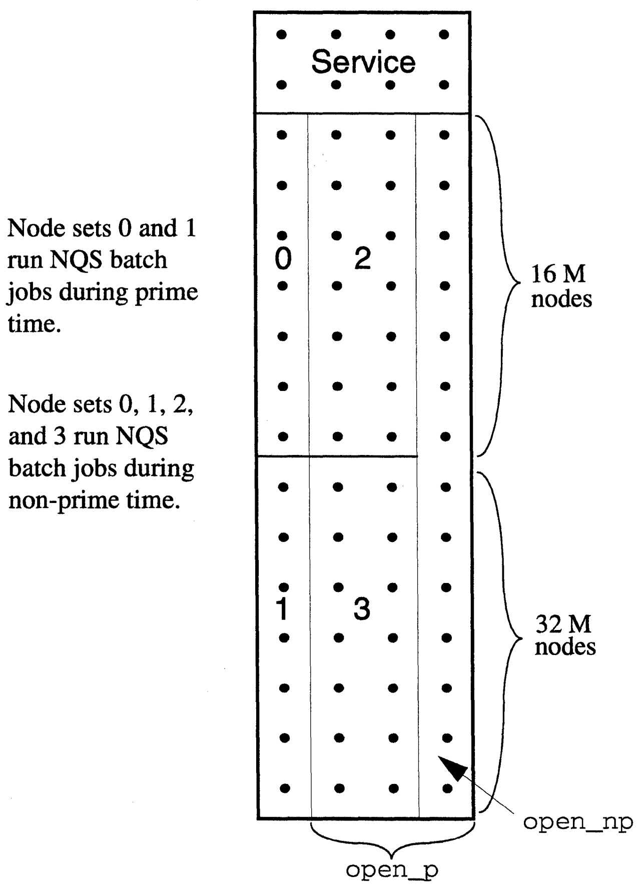
This sample configuration expands on the node configuration presented in Figure 5-3.

```
time_zone : PST8PDT                    #Default. Pacific Standard Time
prime_start : 08 08 08 08 08 08 08  #Default. Begin prime time at 8 AM
prime_end : 18 18 18 18 18 18 18    #Default. End prime time at 6 PM
open_np_name : open_np       #Name the non-prime time open partition open_np
open_p_name : open_p         #Name the prime time open partition open_p
cur_open_name : OPEN                #Name the open partition OPEN
rollin_quan : 600                   #Default. Currently unused
age_factor : 0.1                   #Default. Determines dynamic priority
timeshare : 0                       #Default. Timesharing is turned off
block_pri : -1                      #Default. Small jobs tend to run first
timesh_pri : 10                     #Default. Currently unused
tsched_pri : 20                     #Default. Currently unused
chk_runlimit : 0                    #Default. Run limits are not checked
grace_time : 10       #Default. Kill jobs after 10 seconds if they overrun
do_wall : 0         #Default. No user message during prime/non-prime switch
np_overrun : 0     #Default. Non-prime time jobs can run over into prime time
macs_flag : 1                  #MACS is enabled at this site
use_login : 0                  #Default. User's shell is a non-login shell
nosched : 0                    #Default. NQS schedules jobs normally
prime_script :                 #Default. No script is specified
nprime_script :                #Default. No script is specified
wallclock_limits : 0           #Default. CPU limits enforced as node-hours
node_set0 : .compute 14x1:0    #Name (0), location, and size of a node set
node_set1 : .compute 14x2:1    #Name (1), location, and size of a node set
node_group0 : 0 1   #Name of a node group (0) and its node sets (0,1)
prime_list0 : 0       #Node set in node group 0 that run in prime time (0)
nprime_list0 : 0 1    #Node sets in node group 0 that run in non-prime time
                      #(0,1)
```

Figure 5-4. sched_param File for a Batch/Interactive System

Complex NQS/Interactive Configuration

The configuration example in Figure 5-5 shows a single-cabinet Paragon system with the same prime/none-prime time configuration as in Figure 5-3. This time, however, the compute nodes are grouped into node sets that allow queues to access either 16MB or 32MB nodes, or both.


Node sets 0 and 1 run NQS batch jobs during prime time.
Node sets 0, 1, 2, and 3 run NQS batch jobs during non-prime time.
Service
0
2
1
3
16 M nodes
32 M nodes
open_np
open_p

```
# mkpart-nd 14x3:1 open_p
# mkpart-nd 14x1:3 open_np
```

```
-----------sched_param -----------
open_p_name : open_p
open_np_name : open_np
cur_open_name : OPEN

node_set0 : .compute 7x1:0
node_set1 : .compute 7x1:28
node_set2 : .compute 7x2:1
node_set3 : .compute 7x2:29

node_group0 : 0 2 #16 MB
node_group1 : 1 3 #32 MB

prime_list0 : 0
nprime_list0 : 0 2

prime_list1 : 1
nprime_list1 : 1 3

node_group2 : 0 1 2 3 #All

prime_list2 : 0 1
nprime_list2 : 0 1 2 3
```

```
# qmgr
Mgr: set node_group = 0 q16MB
Mgr: set node_group = 1 q32MB
Mgr: set node_group = 2 qAll
```

Figure 5-5. Node Configuration Using Different Node Types

Running the nqs_setup Script

The following example shows a sample session using the *nqs_setup* script. Before running the script, you should have some idea what configuration parameter values are appropriate for your system.

NOTE

You need to be logged in as root to run the *nqs_setup* script.

```
# cd /usr/lib/nqs/setup
# nqs_setup
Making NQS support directories.
Making NQS spool directories.  This will take a few minutes

Please specify or validate the NQS scheduling parameters given in the
following prompts.  The values displayed in brackets are the default
values.  To use the defaults, just press the Return key.  If you want
to change the default or when no default is given, enter the desired
value followed by the Return key.  If you want more information on
a parameter, enter a question mark (?).

time_zone : [PST8PDT] <Enter>
                 S  M  T  W  T  F  S
prime_start : [08 08 08 08 08 08 08] <Enter>
                 S  M  T  W  T  F  S
prime_end   : [18 18 18 18 18 18 18] <Enter>
open_np_name : [] open_np <Enter>
open_p_name : [] open_p <Enter>
cur_open_name : [] OPEN <Enter>
rollin_quan : [600] 4000 <Enter>
age_factor : [0.1] <Enter>
timeshare : [0] <Enter>
block_pri : [-1] <Enter>
'timesh_pri' paramter has no effect with 'timeshare' set to 0
tsched_pri : [20] <Enter>
chk_runlimit : [0] <Enter>
grace_time : [10] <Enter>
do_wall : [0] <Enter>
np_overrun : [0] <Enter>
macs_flag : [0] 1 <Enter>
use_login : [0] <Enter>
nosched : [0] <Enter>
prime_script : [] <Enter>
nprime_script : [] <Enter>
wallclock_limits: [0] <Enter>
```

```
We are defining the node sets for the Paragon system,
rectangles of nodes with homogeneous attributes in a
single partition.
To end node_set definitions enter 'Q'
Please enter the name of the base partition for the node_set
.compute <Enter>
Please enter node specification for node_set0
14x1:8 <Enter>
node_set0 : .compute 14x1:8
Current base partition == .compute
Do you wish to change base partitions? [y/n] n
Please enter node specification for node_set1
14x2:9 <Enter>
node_set1 : .compute 14x2:9
Current base partition == .compute
Do you wish to change base partitions? [y/n] n
Please enter node specification for node_set2
q <Enter>
We are defining the node_groups as lists of node_sets.
To end node_group definitions enter 'Q'
Please enter the list of node_sets for node_group0
0 1 <Enter>
node_group0 : 0 1
Please enter the list of node_sets for node_group1
q <Enter>
We are defining the Prime-time and Non-Prime-time attribute of
the node_sets per node_group
Enter 'Q' to terminate list entry
Please enter the list of node_sets to be used in Prime time
for node_group0.
0 <Enter>
Please enter the list of node_sets to be used in Non-Prime time
for node_group0.
0 1 <Enter>
prime_list0 : 0
nprime_list0 : 0 1

These are the parameter values that have been set:
time_zone      : PST8PDT
prime_start    : 08 08 08 08 08 08 08
prime_end      : 18 18 18 18 18 18 18
open_np_name   : open_np
open_p_name    : open_p
cur_open_name  : OPEN
rollin_quan    : 4000
age_factor     : 0.1
timeshare      : 0
block_pri      : -1
timesh_pri     : 10
```

```
tsched_pri     : 20
chk_runlimit   : 0
grace_time     : 10
do_wall        : 0
np_overrun     : 0
macs_flag      : 1
use_login      : 0
nosched        : 0
prime_script   :
nprime_script  :
wallclock_limits : 0
node_set0 : .compute 14x1:8
node_set1 : .compute 14x2:9
node_group0 : 0 1
prime_list0 : 0
nprime_list0 : 0 1
Do you want to change any of these now (y/n)? n <Enter>
Running consistency check
Commit these parameters to the configuration file (y/n)? y <Enter>
NQS configuration parameters have been updated.
```

Next, you will be using the **nmapmgr** command to add the Intel supercomputer to the NQS database. In this example, ***mid 1*** stands for machine ID 1 and ***supercomputer*** is the machine name. We show a machine ID of 1 since, in this example, the Paragon system is the only machine in the NQS database (which means that users will be logging in remotely to submit jobs). If you add other machines to the NQS database, each machine must have a unique machine ID. For more information on the **nmapmgr** command, refer to Chapter 6.

```
The nmapmgr(1M) utility must be run to create a database that holds the
names and machine ID numbers of every machine that will be interacting
in the NQS environment.

Would you like the nmapmgr(1M) utility be run now (y/n)? y <Enter>
NMAPMGR>: create <Enter>
NMAP_SUCCESS:  Successful completion
NMAPMGR>: add mid 1 supercomputer <Enter>
NMAP_SUCCESS:  Successful completion.
NMAPMGR>: add name supercomputer.ssd.intel.com 1 <Enter>
NMAP_SUCCESS:  Successful completion.
NMAPMGR>: quit <Enter>
Should NQS be started automatically at system boot (y/n)? y <Enter>
Do you want to start NQS now (y/n)? y <Enter>

NQS services provided.

NQS setup is complete.
#
```

What's Next?

Once you have configured the NQS software, you can add NQS operator and manager accounts and set up queues to manage the NQS batch area. If your site uses MACS, you will need to supply the queue names when you configure MACS. Refer to "Adding NQS Queues to the MACS System" on page 4-10 for information on how to do this.

NQS Reference 6

Introduction

This chapter describes the Paragon system's NQS commands. The NQS commands are presented in alphabetical order. Where appropriate, the command description text notes any commands or options that are only available to NQS network managers.

NQS Commands

The following list shows all of the NQS commands:

nmapmgr	Configures the NQS network database.
qcmplx	Displays the status of NQS queue complexes.
qdel	Deletes requests from the specified queue. You can also send a signal to any request that has already started to run.
qdev	Displays the status of NQS devices.
qlimit	Displays the supported resource limits and the shell strategy for the local host.
qmgr	Defines, configures, and manages queues.
qpr	Queues user files for printing.
qstart	Causes NQS to read the configuration parameters in the *sched_param* file.
qstat	Displays queue status.
qsub	Submits requests to queues.
que_update	Updates the R1.3 queues database with a new format compatible with R1.4.

Also included is a manual page for the *sched_param* file.

NOTE

Dedicated batch requests (sometimes referred to as "tennis court scheduling") are not supported in this release of the Paragon system software. You should ignore these references in the manual pages.

Commands (such as the **qmgr** command) that include a number of subcommands have the subcommands arranged alphabetically under the base command. Commands (such as the **qpr** and **qsub** commands) that include a number of flags have the descriptions of the flags arranged alphabetically under the base command.

NOTE

The NQS commands that have subcommands allow abbreviations for these subcommands. These abbreviations are shown in the syntax statement and are indicated by capital letters.

nmapmgr nmapmgr

NQS network mapping program.

Synopsis

nmapmgr

Description

The **nmapmgr** utility is used by the system administrator when initially configuring NQS, or when changing the NQS configuration. The **nmapmgr** utility creates, views, and/or changes the network mapping for the NQS environment.

NQS refers to a database that holds the names and machine ID numbers of every machine in the NQS environment. If a machine does not have an entry in this NQS database, it is unable to use the NQS resources connected to the network. The NQS database is located in the directory */etc/nmap* of each machine in the NQS environment.

Each machine in the NQS environment maintains its own version of the NQS database. Each machine in the system must also have a unique machine ID number and a unique machine name. Once a machine is identified by a specific ID and name, that ID and name must be used everywhere in the NQS environment to identify that machine.

There is a set of subcommands available to you once you invoke the **nmapmgr** utility. The subcommands allow you to add or delete machines, users, or groups to the database that describes the NQS environment.

Add Gid

Add Gid *from_mid from_gid to_gid*

from_mid	The machine ID number (decimal integer) to which this subcommand applies.
from_gid	The group ID number (decimal integer) of an existing group for this machine.
to_gid	The group ID number (decimal integer) of the group that is being added to this machine.

This subcommand adds the mapping for a group ID number (*to_gid*) to the machine identified by the *from_mid* parameter. The *from_gid* parameter implies an existing GID for every machine in the NQS database. You must be logged in as *root* in order to use this subcommand.

nmapmgr *(cont.)* nmapmgr *(cont.)*

Add Mid

Add Mid *mid principal-name*

mid	The machine ID (mid) for the new machine.
principal-name	The principal name that is to be used by the new machine.

Create a new machine mapping with the specified machine ID (*mid*) and assign the principal name (*principal-name*) to this new machine. The principal name should be a simple name, that is, one without any network mapping information. For example, "orbison" is an appropriate principal name, while "orbison.leah.com" is not. You must be logged in as *root* in order to use this subcommand.

Add Name

Add Name *name to_mid*

name	The new name that is being added to the machine identified by *to_mid.*
to_mid	The machine ID number (decimal integer) to which the new machine name will be added.

Add new mapping for *name* to the machine identified by *to_mid.* You must be logged in as *root* in order to use this subcommand.

Add Uid

Add Uid *from_mid from_uid to_uid*

from_mid	The machine ID number (decimal integer) to which this subcommand applies.
from_uid	The user ID number (decimal integer) of an existing user of this machine.
to_uid	The user ID number (decimal integer) of the user that is being added to this machine.

This subcommand adds the mapping for a user ID number (*to_uid*) to the machine identified by the *from_mid* parameter. The *from_uid* parameter implies an existing UID for every machine in the NQS database. You must be logged in as *root* in order to use this subcommand.

nmapmgr *(cont.)* nmapmgr *(cont.)*

Change Name

CHange Name *mid new-name*

mid	The machine ID (*mid*) for the machine to which this command applies.
new-name	The new principal name that is to be used by this machine.

Change the principal name of the machine identified by (*mid*) to the name identified as *new-name*. You must be logged in as *root* in order to use this subcommand.

Create

CReate

This subcommand creates the NQS mapping database on this machine. No **nmapmgr** subcommands will work until this database has been created. You must be logged in as *root* in order to use this subcommand.

Delete Defgid

Delete DEFGid *from_mid*

from_mid	The machine ID number (decimal integer) to which this subcommand applies.

Delete and disable the default group ID mapping for the machine identified as *from_mid*. You must be logged in as *root* in order to use this subcommand.

Delete Defuid

Delete DEFUid *from_mid*

from_mid	The machine ID number (decimal integer) to which this subcommand applies.

Delete and disable the default user ID mapping for the machine identified as *from_mid*. You must be logged in as *root* in order to use this subcommand.

nmapmgr *(cont.)* nmapmgr *(cont.)*

Delete Gid

Delete Gid *from_mid from_gid*

from_mid	The machine ID number (decimal integer) to which this subcommand applies.
from_gid	The group ID number (decimal integer) of the existing group for this machine.

This subcommand deletes a group ID number (*from_gid*) from the machine identified by the *from_mid* parameter. You must be logged in as *root* in order to use this subcommand.

Delete Mid

Delete Mid *mid*

mid	The machine ID (*mid*) of the machine to which this subcommand applies.

This subcommand deletes all mappings associated with the machine identified as *mid*. This means that all user ID, group ID, name-to-mid, and mid-to-name mappings for the specified machine ID will be deleted when this subcommand executes. You must be logged in as *root* in order to use this subcommand.

Delete Name

Delete Name *name*

name	The principal name of the machine to which this subcommand applies.

Delete the name to machine ID mapping for the machine with principal name *name*. You must be logged in as *root* in order to use this subcommand.

Delete Uid

Delete Uid *from_mid from_uid*

from_mid	The machine ID number (decimal integer) to which this subcommand applies.
from_uid	The user ID for which the machine-to-user mapping is being requested.

This subcommand deletes a user ID number (*from_uid*) from the machine identified by the *from_mid* parameter. You must be logged in as *root* in order to use this subcommand.

Exit

Exit

Leave the **nmapmgr** command environment. Issuing the **Quit** subcommand or pressing the `<CTRL-D>` keys will have the same effect.

Get Gid

Get Gid *from_mid from_gid*

from_mid	The machine ID number (decimal integer) to which this subcommand applies.
from_gid	The group ID for which the machine-to-group mapping is being requested.

Show the machine-to-group mapping from the machine identified as *from_mid* to the group identified as *from_gid*.

Get Mid

Get Mid *name*

name	The principal name of the machine to which this subcommand applies.

Determine the machine ID of the machine whose principal name is *name*.

Get Name

Get Name *mid*

mid	The machine ID number (decimal integer) to which this subcommand applies.

Return the principal host name for the machine specified as *mid*.

Get Uid

Get Uid *from_mid* *from_uid*

from_mid	The machine ID number (decimal integer) to which this subcommand applies.
from_uid	The user ID for which the machine-to-user mapping is being requested.

Show the machine-to-user mapping from the machine identified as *from_mid* to the user identified as *from_uid.*

Help

Help

Typing **help** while the **nmapmgr** utility is active results in the following display:

```
Commands:
  Add Uid <from_mid> <from_uid> <to_uid>
  Add Gid <from_mid> <from_gid> <to_gid>
  Add Name <name> <to_mid>
  Add Mid <mid> <principal-name>
  CHange Name <mid> <new-name>
  CReate
  Delete Uid <from_mid> <from_uid>
  Delete Gid <from_mid> <from_gid>
  Delete DEFUid <from_mid>
  Delete DEFGid <from_mid>
  Delete Name <name>
  Delete Mid <mid>
  Exit
  Get Uid <from_mid> <from_uid>
  Get Gid <from_mid> <from_gid>
  Get Mid <name>
  Get Name <mid>
  Help
  Quit
  Set Nfds <#-of-file-descrs>
  Set DEFUid <from_mid> <defuid>
  Set DEFGid <from_mid> <defgid>
 ^D (to exit nmapmgr.)
```

nmapmgr *(cont.)*

Quit

Quit

Leave the **nmapmgr** command environment. Issuing the **Exit** subcommand or pressing the `<CTRL-D>` keys will have the same effect.

Set Defgid

Set DEFGid *from_mid defgid*

from_mid	The machine ID number (decimal integer) to which this subcommand applies.
defgid	The group ID that is to be used as the default group ID for this machine.

This command enables and sets the default group *defgid* ID mapping for the *from_mid* machine. Any default group ID mapping for this machine is replaced by this command. You must be logged in as *root* in order to use this subcommand.

Set Defuid

Set DEFUid *from_mid defuid*

from_mid	The machine ID number (decimal integer) to which this subcommand applies.
defuid	The user ID that is to be used as the default user ID for this machine.

This command enables and sets the default user ID (*defuid*) mapping for the *from_mid* machine. Any default user ID mapping for this machine is replaced by this command. You must be logged in as *root* in order to use this subcommand.

Set Nfds

Set Nfds *#-of-file-descrs*

#-of-file-descrs	The number of file descriptors (0, 1, or 2) used to control file usage by the **nmapmgr** command.

This is an internal command that affects the processing time of the **nmapmgr** command.

Error Messages

The following error messages may be returned to indicate the success or failure of any of the **nmapmgr** subcommands.

`Integer overflow.`

The maximum supported integer size was exceeded. For the machine ID, the mid cannot exceed 13 decimal digits in size.

`Invalid NMAP return code.`

The value returned by one of the **nmapmgr** procedures was not one of the valid values.

`Missing arguments.`

One or more of the arguments to the **nmapmgr** procedure was missing. All arguments to **nmapmgr** procedures are required. There are no optional arguments.

`Missing <mid>.`

The machine ID argument required by the **nmapmgr** procedure was missing.

`Missing <name>.`

The principal name of the machine required by the **nmapmgr** procedure was missing.

`Missing verb modifier.`

One of the command verbs (Add, Change, Delete, Get, or Set) was not followed by a modifier such as Mid or Uid.

`NMAP_DEFMAP: Successful completion using default mapping.`

No mapping existed for the specified mid/uid(gid) pair, but the default uid(gid) mapping was enabled, and so the default uid(gid) mapping for the specified machine has been returned.

`NMAP_EBADNAME: Null name or name too long.`

A null name or a name too long for mapping was specified.

## nmapmgr *(cont.)*	nmapmgr *(cont.)*

`NMAP_ECONFLICT: Already exists.`

A mapping already existed for the source target which defined a mapping result target different from the new target.

`NMAP_ENOMAP: No such mapping.`

No mapping exists for the specified machine ID and remote user ID or remote group ID pair.

`NMAP_ENOMID: No such machine.`

There is no such machine ID in the current database .

`NMAP_ENOPRIV: No privilege for operation.`

The current user has insufficient privilege to perform the prior operation on the database.

`NMAP_EUNEXPECT: Fatal error in mapping software.`

This message is returned if an unanticipated error occurred in the mapping software.

`NMAP_SUCCESS: Successful completion.`

Return message if the prior operation was successful.

`NMAPMGR command line too long.`

The command line exceeded the maximum line size supported for **nmapmgr**.

`NMAPMGR error reading command input.`

An error was detected in the command line.

`No such mid.`

No mapping was found for the stated machine ID.

`Unrecognized command verb:`

The stated command verb was not one of the recognized command verbs (Add, Change, Create, Delete, Exit, Get, Help, Quit, or Set).

nmapmgr *(cont.)* nmapmgr *(cont.)*

```
Unrecognized command verb modifier.
```

One of the command verbs (Add, Change, Delete, Get, or Set) was not followed by a recognized modifier such as Mid or Uid.

```
Unsigned integer number expected.
```

Signed integers are not supported.

qcmplx qcmplx

Display the NQS queue complexes.

Synopsis

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

Description of Parameters

-h *hostname*	Performs the request on the *hostname* host.
-n	Eliminates the **qcmplx** header display.
-Q	Displays the queues within the complex.
complex-name	The name of the complex for which status information is requested.
complex-name@host-name	An alternate method of specifying the host and complex names.

Discussion

qcmplx displays the NQS queue complexes. A queue complex is a group of queues with a group run limit. In the absence of a **-h** *hostname* 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. To successfully use **-h**, you must have account authorization on the remote host.

Caveats

Although this command is supported on the Paragon system, this command may or may not be supported on all machines in your network.

See Also

qdel, qdev, qlimit, qpr, qstat, qsub, qmgr

qdel qdel

Delete or signal NQS batch requests.

Synopsis

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

Description of Parameters

-k	Send the SIGKILL signal to the request.
-*signo*	Send a specified signal to the identified running request.
-h *hostname*	Delete or signal requests on *hostname.*
-u *username*	Delete or signal requests owned by *username.*
request-id	The unique identity of a request in the NQS system.

Discussion

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. Signals applicable for the OSF/1 **kill** command are also applicable for the **-***signo* flag. 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.

qdel *(cont.)* qdel *(cont.)*

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 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 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** 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** command. This can lead to "rogue" request processes which must be killed by other means such as the **kill** command.

See Also

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

qdev

Display the status of NQS devices.

Synopsis

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

Description of Parameters

device-name	One form of specifying the device or devices for which the status is being requested. The local host is assumed.
device-name@host-name	An alternate way of specifying the device or devices for which the status is being requested. The *host-name* is part of the specification.

Discussion

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()** 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, **qlimit**, **qmgr**, **qpr**, **qstat**, **qsub**

qlimit qlimit

Show supported batch limits and shell strategy for the local host.

Synopsis

qlimit

Discussion

The **qlimit** command displays the batch request resource limit types that can be directly enforced on the local host, and also the batch request shell strategy defined for the local host.

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

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 operating system. If a batch request specifies a limit that cannot be enforced by the underlying operating system, 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 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** 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** commands that alter the limits of the containing batch queue.

qlimit *(cont.)*

This command also displays the shell strategy as configured for the local host. 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. The *use_login* configuration parameter determines if the user's shell is spawned as a login or non-login shell.

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, the "fixed" shell that will be used to run all batch requests is displayed.

See Also

qdel, qdev, qmgr, qpr, qstat, qsub

qmgr qmgr

NQS queue manager program.

Synopsis

qmgr

Description

The **qmgr** program is an interactive program used to control NQS requests, queues, and the general NQS configuration of the Paragon system.

Discussion

A request is a parallel application submitted to the Paragon system by a user or user program via NQS. Requests are submitted to either batch or pipe queues. A batch queue holds requests for scheduled or delayed processing. A pipe queue is a queue that can pass queued requests on to other pipe queues or batch queues. A Paragon NQS manager can change any NQS characteristic on the Paragon system. The NQS operator can execute only a subset of the **qmgr** subcommands.

Queue Types

The Paragon system's NQS supports *batch* and *pipe* queues.

A *batch* queue can only be used to execute NQS batch requests. Only NQS batch requests created by the **qsub** command can be placed in a batch queue.

A *pipe* queue is to send NQS requests to other pipe queues, or to request destination batch queues. In general, pipe queues act as the mechanism that NQS uses to transport batch requests to distant queues on other remote machines. It is also possible 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 request to pass through many pipe queues on its way to its ultimate destination, which must eventually be a batch queue.

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 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.

Queue Access

NQS supports queue access restrictions. For each queue type, 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 the subcommands is of the form:

set some_limit = *(limit) queue*

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

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

White space can appear anywhere between the principal tokens, with the exception that no white space 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.

If a batch request specifies a limit that cannot be enforced by the underlying NQS 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.

Each applicable request limit is 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** 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** subcommands that alter the limits of the containing batch queue.

Subcommands

The following paragraphs describe each **qmgr** subcommand. The subcommand keywords are not case sensitive; however, the keyword can be abbreviated to the characters shown in uppercase in the subcommand's syntax.

All subcommands of the **qmgr** program can be used when the user has NQS manager privileges or Paragon system root privileges. A subset of these subcommands are also accessible to users that only have NQS operator privileges.

Abort Queue

ABort Queue *queue* [*seconds*]

queue	A batch or pipe queue.
seconds	A real value corresponding to the delay time.

All requests in the 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 subcommand are deleted, and all output files associated with the requests are returned to the appropriate destination.

NQS operator privileges are required to use this subcommand.

qmgr *(cont.)* qmgr *(cont.)*

Add Destination

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

destination	The name of a valid destination.
queue	A pipe queue.

The specified destination is added as a valid destination for the pipe queue.

Full NQS manager privileges are required to use this subcommand.

Add Groups

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

group	The login group name of the requester.
queue	A batch or pipe queue.

The specified group(s) are added to the access list for *queue*. It is not permitted to add a group when queue access is unrestricted. You can specify a group either by the group name or by the numeric group ID enclosed in square brackets.

For example, if there is a group *archers* on your machine with group-id 10001, you may specify it with *archers* or [10001]. In order for a user to have access to a queue, it is only necessary for ONE of the following to be true:

- Queue access is unlimited. (See the **Set Unrestricted_Access** description on page 6-53.)
- The user's login group has access.
- The user's account has access. (See the **Add Users** description on page 6-26.)

Full NQS manager privileges are required to use this subcommand.

Add Managers

ADd Managers *user . . .*

user	The valid account name of an NQS user.

The specified user(s) are added to the list of authorized NQS managers with the specified privileges. 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** subcommands. If the account name specification is followed by **:o**, then the account is designated as an NQS operator account, capable of only using those subcommands appropriate for an NQS operator. The following examples show the use of the above four parse constructs for an NQS manager account name:

```
1. kingsbur                        ; Local account of "kingsbur"
2. [149]                         ; Local account (user-id = 149)
3. [149]@Hal9000                ; Remote account (user-id = 149)
                                  ; on remote machine: "Hal9000"
4. [149]@[9000]                 ; Remote account (user-id = 149)
                               ; on remote machine with machine-
                                                    ; id = 9000.
```

Full NQS manager privileges are required to use this subcommand.

Add Queues

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

queue	A batch or pipe queue.
complex	A named batch complex.

Add the specified queues to the batch queue complex named by *complex.*

Full NQS manager privileges are required to use this subcommand.

Add Users

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

user	The valid name of an NQS user.
queue	A batch or pipe queue.

The specified user(s) are added to the access list for *queue.* Users cannot be added to a queue when queue access is unrestricted.

You can specify a user either by the username or by the numeric user ID enclosed in square brackets.

For example, if there is a user *robin* on your machine with *user-id* 1001, you may specify the user with *robin* or [1001]. In order for a user to have access to a queue, it is only necessary for one of the following to be true:

- Queue access is unlimited. (See the **Set Unrestricted_Access** description on page 6-53.)
- The user's login group has access. (See the **Add Groups** description on page 6-24.)
- The user's account has access.

Full NQS manager privileges are required to use this subcommand.

Create Batch_Queue

Create Batch_queue *queue* **PRiority** = *n* [**PIpeonly**] [**Run_limit** = *n*] [**EPL_Nprime** = *non-prime-epl*] [**EPL_Prime** = *prime-epl*]

queue	A named batch queue.
n	An inter-queue priority in the range 0 to 63, with 0 being the lowest priority.
non-prime-epl	An EPL value in the range 0 to 10.
prime-epl	An EPL value in the range 0 to 10.

Defines a batch queue. A queue name can contain any printable non-blank character with the exception that a queue name cannot start with a decimal digit (0–9), and it cannot contain any of the **qmgr** subcommand punctuation characters of ",", "=", " (", and ")". Queue names are limited to 15 characters.

The queue is created in a disabled and stopped state, and must be explicitly enabled and started by the **enable queue** and **start queue** subcommands.

By default, NQS creates queues with one node. You must use the **set per_request ncpus** subcommand to specify more nodes.

If the optional **pipeonly** attribute is specified, then no user is allowed to submit requests directly to the batch queue. A "pipe-only" queue can only receive requests submitted from another pipe queue.

The optional **run_limit** parameter defines the maximum number of requests that will be allowed to run in the queue at any given time. If no run limit is given, the queue is assigned a default run-limit of 1.

The optional **epl_nprime** parameter defines the effective priority limit for this queue's partitions during the non-prime time period. If no non-prime EPL is given, the queue is assigned a default non-prime time effective priority limit of 3.

The optional **epl_prime** parameter defines the effective priority limit for this queue's partitions during the prime time period. If no prime EPL is given, the queue is assigned a default prime time effective priority limit of 1.

Full NQS manager privileges are required to use this subcommand.

qmgr *(cont.)* **qmgr** *(cont.)*

Create Complex

Create Complex = (*queue* [, *queue*. . .]) *complex*

queue	A named batch queue.
complex	A named batch 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 subcommand.

Create Pipe_Queue

Create Pipe_queue *queue* **PRiority** = *n* **Server** = (*server*)
[**Destination** = *destination*]
[**Destination** = (*destination* [, *destination* . . .])]
[**PIpeonly**] [**Run_limit** = *n*]

queue	A named pipe queue.
n	An integer. See the following text.
server	The name of the server program associated with this pipe queue.
destination	The name of a valid destination queue.

Define a pipe queue named *queue* with inter-queue priority *n* (0 to 63) and associate it with a *server*. This is done by specifying an absolute path name to the program binary (specified by *server*) and any arguments required by the program. The queue name can contain any printable non-blank character except it cannot start with a decimal digit (0–9), and it cannot contain any of the **qmgr** subcommand punctuation characters of "@", ",", "=", " (", and ")".

After **destination** is a list of one or more destination queues that requests from this pipe queue may be sent to. The syntax of a pipe queue destination (specified by *destination*) if specified, must conform to one of the following four forms:

- *local_queue_name*
- *local_queue_name*@*local_machine_name*
- *remote_queue_name*@*remote_machine_name*
- *remote_queue_name*@[*remote_machine_mid*]

where the *local_machine_name* or *remote_machine_name* must be defined in the network host table of the local system. The destination syntax form:

- *remote_queue_name@[remote_machine_mid]*

requires the explicit specification of the destination machine by its machine-id, entered as an integer:

```
pipequeue@[123]
```

If **pipeonly** is specified, then requests may enter this queue only if their source is a pipe queue. The 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 `1`. (See the "See Also" discussion on page 6-56 for more information.) The queue is created in a disabled and stopped state, and must be explicitly enabled and started by the **enable queue** and **start queue** subcommands.

Full NQS manager privileges are required to use this subcommand.

Delete Complex

DElete Complex *complex*

complex	A named batch complex.

This subcommand deletes a queue complex.

Full NQS manager privileges are required to use this subcommand.

Delete Destination

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

queue	A named pipe *queue*.
destination	The name of a valid destination queue.

Delete the mappings from the pipe queue to the destination queue(s). 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, although its actual status will remain unchanged. Thus, the addition of a new destination for a pipe queue that has been effectively stopped in the manner described, will immediately cause the queue to start running again.

The syntax of a pipe queue destination (*destination*) if specified, must conform to one of the following four forms:

- *local_queue_name*
- *local_queue_name@local_machine_name*
- *remote_queue_name@remote_machine_name*
- *remote_queue_name@[remote_machine_mid]*

where the *local_machine_name* or *remote_machine_name* must be defined in the network host table of the local system. The destination syntax form:

- *remote_queue_name@[remote_machine_mid]*

requires the explicit specification of the destination machine by its machine-id, entered as an integer:

```
pipequeue@[123]
```

Full NQS manager privileges are required to use this subcommand.

Delete Groups

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

group	The login group name of the requester.
queue	A batch or pipe queue.

The specified group(s) are deleted from the access list for *queue*. This subcommand can only delete groups that have been added with the **add groups** subcommand. It is not permitted to delete a group from a queue where access is unrestricted.

You can specify a group either by the group name or by the numeric group ID enclosed in square brackets.

For example, if there is a group *archers* on your machine with group-id 10001, you may specify it with *archers* or [10001]. If the *group* name does not appear in the system group file, the latter form must be used. In order for a user to have access to a *queue*, it is only necessary for one of the following to be true:

- Queue access is unlimited. (See the **Set No_Access** description on page 6-46.)
- The user's login group has access.
- The user's account has access. (See the **Delete Users** description on page 6-32.)

Full NQS manager privileges are required to use this subcommand.

Delete Managers

DElete Managers *manager . . .*

manager	A valid NQS manager specification.

The specified manager(s) are deleted from the access list of authorized NQS managers. Attempts to exclude the system root account from the NQS manager set will be silently ignored. 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*]

Each of the following four examples demonstrates the use of the above four parse constructs for an NQS *manager* name.

```
kingsbur                          ; Local account of "kingsbur"
[149]                            ; Local account (user-id = 149)
[149]@Hal9000                   ; Remote account (user-id = 149)
                                  ; on remote machine: "Hal9000"
[149]@[9000]                    ; Remote account (user-id = 149)
                                ; on remote machine with machine
                                                      ; id = 9000.
```

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

Full NQS manager privileges are required to use this subcommand.

Delete Queue

DElete Queue *queue*

queue	A batch or pipe queue.

The queue is deleted. To delete a queue, no requests may be present in the queue and the queue must be disabled (see the **Disable Queue** description on page 6-33).

Full NQS manager privileges are required to use this subcommand.

Delete Users

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

user	The valid name of an NQS user.
queue	A batch or pipe queue.

The specified user(s) are deleted from the access list for *queue*. This subcommand can only delete users who have been added with the **add users** subcommand. It is not permitted to delete a user from a queue where access is unrestricted.

You can specify a user either by the username or by the numeric user ID enclosed in square brackets.

For example, if there is a user *robin* on your machine with *user-id* 1001, you may specify the user with *robin* or [1001]. In order for a user to have access to a queue, it is only necessary for one of the following to be true:

- Queue access is unlimited. (See the **Set Unrestricted_Access** description on page 6-53.)
- The user's login group has access. (See the **Add Groups** description on page 6-24.)
- The user's account has access.

Full NQS manager privileges are required to use this subcommand.

Disable Queue

DIsable Queue *queue*

queue	A batch or pipe queue.

Prevent any more requests from being placed in this queue.

NQS operator privileges are required to use this subcommand.

Enable Queue

ENable Queue *queue*

queue	A batch or pipe queue.

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

NQS operator privileges are required to use this subcommand.

Exit

EXit

Exit from the **qmgr** utility. The NQS manager program is also exited if an end-of-file is encountered on the standard input file. Thus, typing `<Ctrl-D>` will also cause the NQS manager program to exit.

Help

Help [*subcommand*]

subcommand	A **qmgr** utility subcommand.

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

Lock Local_Daemon

Lock Local_daemon

Lock the NQS local daemon into memory. See the **plock(2)** description in the OSF/1 documentation.

NQS operator privileges are required to use this subcommand.

NOTE

This subcommand has been disabled in this release of the Paragon system software.

Modify Request

MODify Request *request-id* [**Nice_limit =** *nice*] [**RTime_limit =** *Tlimit*]

nice	A valid nice value.
Tlimit	A CPU time limit.
request-id	A valid request identifier.

Modify parameters for the request specified by *request-id. Nice* is the initial nice value for the request. *Tlimit* is a per-request CPU time limit. For the syntax of these limits, see the LIMITS section below. This subcommand only applies to requests that are not already running.

NQS operator privileges are required to use this subcommand.

Move Queue

MOVe Queue *queue1 queue2*

queue1	A batch or pipe queue.
queue2	A batch or pipe queue.

Move all requests currently in *queue1* to *queue2*. Jobs in a RUNNING or WAITING state cannot be moved.

NQS operator privileges are required to use this subcommand.

Move Request

MOVe Request = (*request-id* [,*request-id* . . .]) *queue*

request-id	A valid request identifier.
queue	A batch or pipe queue.

Move requests identified by *request-id* to the queue named by *queue*.

NQS operator privileges are required to use this subcommand.

Purge Queue

Purge Queue *queue*

queue	A batch or pipe 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 subcommand.

Remove Queues

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

queue	A batch or pipe queue.
complex	A named batch complex.

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

Full NQS manager privileges are required to use this subcommand.

Set Complex Run_Limit

SEt COMplex Run_limit = *run-limit complex*

run_limit	A specified number of requests.
complex	A named batch 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 subcommand.

Set Debug

SEt DEBug *level*

level	A valid debug level number.

Set the debug level. The following values are valid:

0	No debug
1	Minimum debug
2	Full debug
3	Maximum debug

The debug messages are written to the NQS logfile.

Full NQS manager privileges are required to use this subcommand.

Set Default Batch_Request Queue

SEt DEFault Batch_request Queue *queue*

queue	A named batch queue.

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

Full NQS manager privileges are required to use this subcommand.

Set Default Destination_Retry Time

SEt DEFault DEStination_retry Time *retry_time*

retry_time	A number of hours.

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 subcommand.

Set Default Destination_Retry Wait

SEt DEFault DEStination_retry Wait *interval*

interval	A number of minutes.

Set the default number of minutes to wait before retrying a pipe queue destination that was unreachable at the time of the last attempt. When a pipe queue destination fails to accept a request because of a network failure, or remote server failure, then the request is not transferred, and the destination is disabled for the number of minutes as determined by the *interval* parameter. At the end of this time, the destination is re-enabled, and retried as appropriate. To prevent an infinite number of retries, the **set default destination_retry time** subcommand is used to prevent a pipe queue destination from being endlessly retried forever.

Full NQS manager privileges are required to use this subcommand.

Set Destination

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

destination	A valid destination for a pipe queue.
queue	A named pipe queue.

Associate one or more destination queues with a particular pipe queue. All previous destinations in the pipe queue destination set are discarded. All machine-names appearing in any queue destination names must be defined in the local system's network host table.The syntax of a pipe queue destination (specified by *destination*) must conform to one of the following four forms:

- local_queue_name
- *local_queue_name@local_machine_name*
- *remote_queue_name@remote_machine_name*
- *remote_queue_name@*[*remote_machine_mid*]

where the *local_machine_name* or *remote_machine_name* must be defined in the network host table of the local system. The destination syntax form:

- *remote_queue_name@*[*remote_machine_mid*]

requires the explicit specification of the destination machine by its machine-id, entered as an integer:

```
pipequeue@[123]
```

Full NQS manager privileges are required to use this subcommand.

Set EPL_Nprime

SEt EPL_Nprime *effective-priority-limit queue*

effective-priority-limit	An EPL value in the range 0 to10.
queue	A named batch queue.

Changes the effective priority limit (EPL) for the named batch queue's partitions during the non-prime time period. The queue named must already exist.

(NQS creates a partition for each request to run in. The effective priority limit (EPL) of a partition sets a maximum value for the allocator priority of the parallel applications in the partition. An application's allocator priority determines whether or not the application runs if the application overlaps with nodes that are in use by other applications. See the *Paragon*™ *User's Guide* for more information on EPLs.)

If the **set epl_nprime** and **set epl_prime** subcommands are issued while jobs are in the queue, requests in a QUEUED state will have the new EPL value when the request's partition is created. However, the EPL value for requests in a RUNNING state are unchanged. Once the request is executing, it retains the same prime and non-prime EPL values for its lifetime, even if its lifetime spans a prime/non-prime or non-prime/ prime switch.

The **qstat -bl** command and the **qmgr show long queue** subcommand show the prime and non-prime time EPLs for a queue.

Full NQS manager privileges are required to use this command.

Set EPL_Prime

SEt EPL_Prime *effective-priority-limit queue*

effective-priority-limit	An EPL value in the range 0 to10.
queue	A named batch queue.

Changes the effective priority limit (EPL) for the named batch queue's partitions during the prime time period. The queue named must already exist.

(NQS creates a partition for each request to run in. The effective priority limit (EPL) of a partition sets a maximum value for the allocator priority of the parallel applications in the partition. An application's allocator priority determines whether or not the application runs if the application overlaps with nodes that are in use by other applications. See the *Paragon™ User's Guide* for more information on EPLs.)

If the **set epl_nprime** and **set epl_prime** subcommands are issued while jobs are in the queue, requests in a QUEUED state will have the new EPL value when the request's partition is created. However, the EPL value for requests in a RUNNING state are unchanged. Once the request is executing, it retains the same prime and non-prime EPL values for its lifetime, even if its lifetime spans a prime/non-prime or non-prime/ prime switch.

The **qstat -bl** command and the **qmgr show long queue** subcommand show the prime and non-prime time EPLs for a queue.

Full NQS manager privileges are required to use this command.

Set Global Batch_Limit

SEt Global Batch_limit = *limit*

limit	The number of concurrent batch requests.

Sets the maximum number of batch requests that can run concurrently on the local host.

NQS operator privileges are required to use this subcommand.

Set Hardulimit

SEt HARDUlimit = *nodes*

nodes	The number of nodes.

Specifies the total number of nodes any user can use at a time. This limit cannot be exceeded even if the NQS batch partition is not fully utilized and there are no other waiting requests. Compare to ''Set Softulimit''.

Full NQS manager privileges are required to use this subcommand.

Set Lifetime

SEt LIfetime *lifetime*

lifetime	A number of hours.

Set the pipe queue request lifetime in hours. This value determines the number of hours that any request can reside within a pipe queue, from the time at which the request was first placed within the queue. If any request resides within a pipe queue for a duration of time equal to or greater than this limit (for example, if the pipe queue is stopped), then the request is deleted, and mail is sent informing the owner of the request that the request has been deleted.

The *lifetime* parameter exists to prevent requests from filling up all available queue space because of network failures, or other causes that would prevent requests from being delivered to their respective destinations. This value should be typically configured to describe a fairly long time interval (such as 72 hours). Alternatively, the *lifetime* parameter can be assigned a value of `0`, which indicates that all requests residing within a pipe queue have an infinite lifetime.

Full NQS manager privileges are required to use this subcommand.

Set Log_File

SEt LOg_file *filename*

filename	A valid file name.

Specify the name of a new log file for NQS messages. If the specified file is successfully opened for append, then NQS writes a message on the old logfile stating that a switch is being made to the new logfile. Otherwise, an error message is written on the current log file and the current logfile is not changed.

The logfile contains ERROR, FATAL, INFO, WARNING, DEBUG and LOG messages, and each message includes the date and time it was written. Logfiles specified with this subcommand are not automatically backed up by NQS.

Full NQS manager privileges are required to use this subcommand.

Set Mail

SEt MAIl *userid*

userid	The user id of the user that is to receive NQS mail messages.

Specify the user id used to send NQS mail. This subcommand is used to set the account name that will appear in the From: portion of mail sent by NQS to notify users (when appropriate), of certain events concerning their NQS request(s).

The mail account name can contain any printable non-blank character except it cannot contain the @ character or start with a decimal digit (0–9), and it cannot contain any of the **qmgr** subcommand punctuation characters of ",", "=", " (", and ")".

Full NQS manager privileges are required to use this subcommand.

Set Managers

SEt MANagers *user*:{m | o} . . .

user	The name of a valid user that has an NQS account.

The list of authorized NQS managers is set to the specified manager(s). A manager specification consists of an account name specification, a colon, and 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**, the account is designated as an NQS manager account, capable of using all **qmgr** subcommands. If the account name specification is followed by **:o**, the account is designated as an NQS operator account, capable of using only NQS operator subcommands. The root account always has full privileges. Each of the following four examples demonstrates the use of the above four parse constructs for an NQS manager.

```
kingsbur                             ; Local account of "kingsbur"
[149]                               ; Local account (user-id = 149)
[149]@Hal9000                      ; Remote account (user-id = 149)
                                     ; on remote machine: "Hal9000"
[149]@[9000]                       ; Remote account (user-id = 149)
                                   ; on remote machine with machine
                                                       ; id = 9000.
```

Attempts to exclude the system root account from the NQS manager set as a fully privileged manager account are silently ignored. Also see the **Add Managers** description on page 6-25.

Full NQS manager privileges are required to use this subcommand.

Set Network Client

SEt NEtwork Client = (*client*)

client	The valid pathname to the client associated with a pipe queue. The text within the enclosing parentheses defines the command line that will be used to execute the client.

Specify the network client to be used. The *client* parameter should consist of the absolute path name of the client followed by any arguments required by the client. The network client stages out batch request output files and empties network queues.

Full NQS manager privileges are required to use this subcommand.

Set Network Daemon

SEt NEtwork Daemon = (*daemon*)

daemon	The valid pathname to the daemon associated with the network. The text within the enclosing parentheses defines the command line that will be used to execute the daemon.

Specify the network daemon to be used. The network daemon listens for messages from remote clients. The *daemon* parameter 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 subcommand.

Set Network Server

SEt NEtwork Server = (*server*)

server	The valid pathname to the server associated with the network. The text within the enclosing parentheses defines the command line that will be used to execute the server.

Specify the network server to be used. The network server processes submission requests from remote pipe clients. The *server* parameter 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 subcommand.

Set Nice_Value_Limit

SEt NIce_value_limit = *nice-value queue*

nice-value	An integer preceded by an optional negative sign.
queue	A named batch queue.

Set the 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. Requests with numerically higher execution priorities than the nice-value are rejected.

The nice-value assigned to a particular batch request is determined at the time that the request is queued. If the nice-value limit for the batch queue is later raised (for example, the new nice-value is numerically greater than the old nice-value), no requests in the queue will be affected. However, if the new nice-value limit is numerically greater than the nice-value requested by a previously queued request, then a warning diagnostic is displayed.

Unless a process of the request is running with an effective user-id of root, it will be unable to lower its nice value (raise its execution priority) from the value determined when the request was queued.

Nice-values in the range -20 to -1 actually specify a higher execution priority than the default of 0, while values in the range 1 to 19 specify a lower execution priority than the default value.

Full NQS manager privileges are required to use this subcommand.

Set Node_group

SEt NODE_Group = *node_group-value queue*

node_group-value	An integer in the range 0 to 63. If *node_group-value* is not specified, the queue is associated with node group 0.
queue	A batch or pipe queue.

Assigns a batch queue to the node group specified by *node_group-value*.

Full NQS manager privileges are required to use this subcommand.

Set No_Access

SEt NO_Access *queue*

queue	A batch or pipe queue.

Specify that no one will be allowed to place requests in *queue*. Requests in the queue will be allowed to remain there. In order to enforce access restrictions on a queue where access is currently unrestricted, start with this subcommand and then add groups or users as needed. Root is an exception; requests submitted by root are always allowed into a queue, even if root is not explicitly given access. If you wish to deny access to root, see the **Disable Queue** description on page 5-33.

In order for a user to have access to a queue, it is only necessary for one of the following to be true:

- Queue access is unlimited.
- The user's login group has access. (See the **Delete Groups** description on page 6-30.)
- The user's account has access. (See the **Delete Users** description on page 6-32.)

Full NQS manager privileges are required to use this subcommand.

Set No_Default Batch_Request Queue

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 subcommand.

Set No_Network_Daemon

SEt NO_Network_daemon

Indicate that there is to be no network daemon.

Full NQS manager privileges are required to use this subcommand.

Set Per_Process Permfile_Limit

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

limit	A value indicating file size.
queue	A named batch 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. The default per-process permanent file size is unlimited. If a request already in the queue has asked for more than the new limit, then it will be given a grandfather clause. However, if the new permanent-file size limit value is less than the permanent-file size usage limit requested by a previously queued request, then a warning diagnostic is displayed.

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. The size of any permanent-file created by any process comprising the running batch request cannot exceed the per-process maximum permanent file size as determined when the request was queued. For the syntax of *limit*, see the "Limits" discussion on page 6-21.

Full NQS manager privileges are required to use this subcommand.

Set Per_Request CPU_Limit

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

limit	A value indicating time. For the syntax of *limit*, see the "Limits" discussion on page 6-21.
queue	A named batch queue.

Set a per-request time limit for a batch queue. The value of *limit* determines how long a request can run. For example, if *limit* is set to an hour, NQS attempts to limit all requests in the queue to a maximum running time of one hour. This value is unlimited by default.

The enforcement mechanism NQS uses to enforce the CPU limit depends on the setting of the *wallclock_limits* configuration parameter. When *wallclock_limits* is set to `0`, NQS multiplies the *limit* by the number of nodes allocated to the request (which is the number of nodes specified for the queue with the **set per_request ncpus** subcommand or the number specified by **qsub -lP**, whichever is less) to arrive at the total limit for the job in node-hours. This means that requests can run longer than expected if the request uses fewer nodes than the queue allows (this can only occur if the request uses **-sz** or sets *NX_DFLT_SIZE*). For example, if an application using **-sz 2** runs in a 4-node queue that has a CPU limit of one minute, it can run for up to two minutes. When *wallclock_limits* is set to `1`, the CPU limit is enforced as simple wall-clock time regardless of the number of nodes in the queue.

NQS will reject a batch request if it requests more time (using **qsub -lT**) than that specified by *limit*. NQS will kill a request once it exceeds the time specified by *limit*.

The time limit for a particular batch request at execution time is determined at the time that the request is queued. If the *limit* for the containing batch queue is later reduced, no requests in the queue are affected (NQS will issue a message indicating that the request has been grandfathered). If the request is later re-queued, a warning diagnostic is displayed.

Full NQS manager privileges are required to use this subcommand.

Set Per_Request NCPUS

SEt PER_Request Ncpus = *nodes queue*

nodes	A value indicating the number of nodes for this queue.
queue	A named batch queue.

Set a per-request number of CPUs for a batch queue. If a batch queue does not have a number of CPUs defined using the **set per_request ncpus** subcommand, it will default to a value of 1. A user may explicitly specify the number of CPUs to use with the **-lP** flag of the **qsub** command if the value specified is less than or equal to the number of nodes associated with the queue.

Full NQS manager privileges are required to use this subcommand.

Set Priority

SEt PRiority = *priority queue*

priority	An integer (0 to 63) corresponding to queue priority (63 = highest).
queue	A batch or pipe queue. The queue named as a parameter of the command must already exist.

Specify the inter-queue priority of a queue.

Full NQS manager privileges are required to use this subcommand.

Set Pipe_Client

SEt PIpe_client = (*client*) *queue*

client	The valid pathname to the client associated with a pipe queue.
queue	A named pipe queue. The text placed within the parentheses defines the fully qualified pathname of the client, and client arguments that are to be used as the command line for a pipe queue client invocation by NQS.

Associate a pipe client with a pipe queue. The *client* parameter should consist of the absolute path name to the program binary followed by any arguments required by the program. The named queue cannot refer to a batch or device queue.

Full NQS manager privileges are required to use this subcommand.

Set Queue_Request_Limit

SEt Queue_request_limit = (*queue-request-limit*) *queue*

queue-request-limit	The number of requests any one user can have at one time (in a RUNNING or QUEUED state) in the specified queue. The *queue-request-limit* can be the word **unlimited** (or simply **u**) or an integer in the range 1 to 32767. The default for *queue-request-limit* is UNLIMITED.
queue	A batch queue. The queue named must already exist.

Any attempt to submit additional requests above this limit will be rejected with a "permission denied" error.

Full NQS manager privileges are required to use this subcommand.

Set Run_Limit

SEt Run_limit = *run-limit queue*

run-limit	Maximum number of requests that can run at once.
queue	A batch or pipe queue. The queue named must already exist.

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 subcommand.

Set Shell_Strategy Fixed

SEt SHell_strategy FIxed = (*shell*)

shell	The valid pathname to a command interpreter.

Specify what shell should be used to execute all batch requests. The *shell* parameter must be the absolute path name of a command interpreter. This subcommand exists for NQS installations that use only one type of shell (e.g. Bourne shell, C-shell, Korn shell, etc.). A "fixed" shell strategy reduces the amount of virtual memory and number of processes consumed by each NQS request.

The configuration parameter *use_login* determines whether or not the specified shell is invoked as a login shell.

Full NQS manager privileges are required to use this subcommand.

Set Shell_Strategy Free

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. This will result in an extra shell being spawned for all batch request executions, just as would happen if the user were running their batch request script interactively.

Under this strategy, the user's shell is invoked to determine the shell to be used to execute the batch request. The user's shell is named within the user's entry in the password file (see the **passwd**(4) description in the OSF/1 documentation). The user's shell picks a shell to execute the batch request as if the batch request was a shell script.

The configuration parameter *use_login* determines whether or not the user's shell is invoked as a login shell.

Full NQS manager privileges are required to use this subcommand.

Set Shell_Strategy Login

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 shell is the shell named in the password file (see the **passwd**(4) description in the OSF/1 documentation). This means that the user's shell will be spawned to run all batch request scripts, without examining the batch shell script to determine which type of script it is.

The configuration parameter *use_login* determines whether or not the user's shell is invoked as a login shell.

Full NQS manager privileges are required to use this subcommand.

Set Softulimit

SEt SOFTUlimit = *nodes*

nodes	The number of nodes.

Specifies the total number of nodes any user can use at a time. This limit can be exceeded if the NQS batch partition is not fully utilized and there are no other waiting requests. Compare to ''Set Hardulimit''.

Full NQS manager privileges are required to use this subcommand.

Set Stack_Limit

SEt STack_limit = (*limit*) *queue*

limit	A value indicating size.
queue	A named batch 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. The default stack limit is unlimited. 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. If an attempt is made to queue a request which asks for a larger stack-segment size limit than the destination queue has defined as allowable, then the request is rejected. Stack-segment limits cannot be applied to pipe queues. Any attempt to set a stack-segment limit for a pipe queue will fail, with an appropriate error diagnostic being displayed.

The maximum stack-segment size limit assigned to a particular request is determined at the time that the request is queued. If the stack-segment size limit for the containing queue is later reduced, no requests in the queue will be affected. However, if the new stack-segment size limit value is less than the stack-segment size limit requested by a previously queued request, then a warning diagnostic is displayed. For the syntax of *limit,* see the "Limits" discussion on page 6-21.

Full NQS manager privileges are required to use this subcommand.

Set Unrestricted_Access

SEt UNrestricted_access *queue*

queue	A batch or pipe queue.

Specify that no requests will be turned away from *queue* on the grounds of queue access restrictions. In order to enforce access restrictions on a queue where access is currently unrestricted, see the **Set No_Access** description on page 6-46.

In order for a user to have access to a queue, it is only necessary for one of the following to be true:

- Queue access is unlimited.
- The user's login group has access. (See the **Add Groups** description on page 6-24.)
- The user's account has access. (See the **Add Users** description on page 6-26.)

Full NQS manager privileges are required to use this subcommand.

Show All

SHOw All

Display information about limits supported, managers, parameters, and queues.

Show Limits_Supported

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** subcommands 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** is invoked. If 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. For a more complete discussion of the limits supported by NQS, see the "Limits" discussion on page 6-21.

Show Long Queue

SHOw LOng Queue [*queue-name* [*user-name*]]

queue-name	The valid name of an NQS queue.
user-name	The valid name of an NQS user.

Display in long format the status of all NQS queues on this host.

If no queue name is specified, then status information is displayed for all NQS queues. If a queue name is specified, output will be limited to that queue. The specific status information for the named queue is displayed, including the ordering of requests within the queue. If a user name is specified, output is limited to the requests belonging to that user.

Show Managers

SHOw Managers

Display the list of authorized NQS managers.

Show Parameters

SHOw Parameters

Display the current general NQS parameters.

Show Queue

SHOw Queue [*queue-name* [*user-name*]]

queue-name	The valid name of an NQS queue.
user-name	The valid name of an NQS user.

Display the status (in short format) of all NQS queues on this host. If no queue name is specified, then status information is displayed for all NQS queues. If a queue name is specified, output will be limited to that queue. If a user name is specified, output is limited to the requests belonging to that user.

Shutdown

SHUtdown [*seconds*]

seconds	A number of seconds.

Shut down NQS on the local host. If *seconds* is not specified, then the shutdown is immediate if no jobs are running.

If jobs are running, a SIGTERM signal is sent to each running job. After the specified number of seconds of real time have elapsed, a SIGKILL signal is sent to all remaining processes for each request. 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 subcommand.

Start Queue

STArt Queue *queue*

queue	A batch or pipe queue.

Start the queue and begin selecting requests. If the queue is already started, then nothing happens.

NQS operator privileges are required to use this subcommand.

Stop Queue

STOp Queue *queue*

queue	A batch or pipe queue.

Stop the named 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. No requests in the named queue will be run until the queue has been explicitly started again by the **start queue** subcommand.

NQS operator privileges are required to use this subcommand.

Unlock Local_Daemon

Unlock Local_daemon

Remove a lock that has been keeping the NQS local daemon in memory. See the **plock(2)** description in the OSF/1 documentation for more information.

NQS operator privileges are required to use this subcommand.

See Also

passwd, **plock**, **qdel**, **qdev**, **qlimit**, **qpr**, **qstat**, **qsub**

qpr qpr

Submit a hardcopy print request to NQS.

NOTE

The Paragon system supports print operations to ensure a full implementation of NQS. However, print operations are not readily applicable to the Paragon system.

Synopsis

qpr [*options*] [*files*]

Description of Parameters

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 explicit **-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 prepended 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.*
files	The valid names of any files that are to be queued for printing.

qpr *(cont.)* **qpr** *(cont.)*

Discussion

The **qpr** command 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** and **qstat** to find out what happened to a request, and given as an argument to **qdel** 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.

Queue Access

NQS supports queue access restrictions. For each queue, 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 the **NQS queue manager program.** description on page 4-20). Requests submitted by root are an exception; they are always queued, even if root has not explicitly been given access.

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

See Also

mail, qdel, qdev, qlimit, qmgr, qstat, qsub

qstart

Causes NQS to read the configuration parameters in the *sched_param* file.

Synopsis

qstart

Discussion

The **qstart** command submits a "dummy" request to NQS, which causes NQS to read the configuration parameters in the *sched_param* file. NQS reads the *sched_param* file every 15 minutes or whenever an NQS request starts or finishes. The **qstart** command is used primarily to pick up the setting of the *nosched* configuration parameter, which can be used to prevent NQS from rescheduling a job that has crashed the Paragon system.

If an NQS job crashes the system, **qstart** can be used in the following sequence of events to prevent the job from restarting and crashing the system again:

1. Boot the system to single-user mode.
2. Edit the *sched_param* file and change *nosched* to `1`.
3. Continue booting to multi-user mode.
4. Remove the problem job from the queue using the **qdel** command.
5. Edit the *sched_param* file and change *nosched* to `0`.
6. Issue the **qstart** command to pick up the new value for *nosched*, which will allow NQS to schedule requests immediately. (If you don't issue **qstart**, NQS will begin scheduling requests within 15 minutes of changing *nosched* to `0`.)

See Also

qcmplx, **qdel**, **qdev**, **qlimit**, **qmgr**, **qpr**, **qsub**

qstat qstat

Displays the status of NQS requests and queues.

Synopsis

qstat [**-a** | **-U**] [**-b**] [**-d**] [**-f** | **-l**] [**-h** *hostname*] [**-n**] [**-p**] [**-s** *state*]
[**-t** *n_days queue*] [**-u** *user*] [*queue* ...] [*queue@host* ...] [**-v**]

Description of Parameters

-a	Displays the status of requests belonging to all users.
-b	Displays batch queues.
-d	Displays device queues.
-f	Shows queues or requests in a full format, that is, multiple lines for each request or queue. The default listing shows one line for each request or queue.
-h *host-name*	Displays requests for queues on the *host-name* host.
-l	Same as **-f**.
-n	The queue header and trailer are not displayed.
-p	Displays pipe queues.
-s *state*	Displays only those requests with the specified state. Allowable states are:

r	Running
h	Holding
q	Queued
t	Exiting

qstat *(cont.)* qstat *(cont.)*

-t *n_days queue*	(Currently unused.) Displays the tennis court schedule for a dedicated request up to 7 days in advance. The *n_day* is an integer from 0 to 6, where 0 is the current day. If a 0 is specified, the available time slot will be listed. If an integer *n* from 1 to 6 is specified, the available time slots for the day which is *n* days in advance of the current day will be listed.
-u *user*	Shows only those requests belonging to the specified *user*. By default, **qstat** shows requests belonging to the invoking user.
-U	Same as **-a**.
-v	Displays output in 132-column format, and adds a NODES field (number of nodes associated with a request) and a SUBMITTED AT field (date and time the request was submitted) to the standard display.
queue ...	The name of a queue for which the status is being requested. The local host is assumed if it is not indicated with the *@host* specifier.
queue@host ...	The name of a queue for which the status is being requested. The host named *host* is indicated.

Discussion

The **qstat** command displays the status of NQS requests and queues. In the absence of **-a**, **-u**, or **-U**, **qstat** shows only requests and queues belonging to the user who executes it.

If no options are specified, NQS displays the current state of each NQS request on the local host. Otherwise, the type of information displayed is determined by the invocation flag used.

qstat displays information about each selected queue. The flags can be used to display particular types of information.

If information about the queues is requested with the **-b**, **-d**, or **-p** flags, 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.

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.

One of the words AVAILABLE, STOPPED, DISABLED, UNAVAIL, or NQS DOWN will appear in the queue status field to indicate the respective queue states of:

AVAILABLE	Enabled and started.
STOPPED	Enabled and stopped.
DISABLED	Disabled and started.
UNAVAIL	Disabled and stopped.

Requests can only be submitted to the queue if the queue is enabled, and the local NQS daemon is present.

If the NQS daemon for the local host upon which the queue resides is not running, the status displays NQS DOWN.

Caveats

The **-s** flag currently gives incorrect results.

See Also

qcmplx, **qdel**, **qdev**, **qlimit**, **qmgr**, **qpr**, **qsub**

qsub qsub

Submit an NQS batch request.

Synopsis

qsub [*flags*] [*script-file*]

Short Description of Parameters

The **qsub** flags are listed briefly below. An asterisk indicates that the flag is not supported on the Paragon system, but can be used when submitting NQS jobs from the Paragon system to other systems that do support the flag. A verbose description of all flags follows the Discussion.

-c	establish per-request account
-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
-lP	establish per-request number of nodes
-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
-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
-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

The *script-file* is an optional batch file that contains embedded default flags that set default characteristics for the batch request.

Discussion

The **qsub** command submits a batch request to the Network Queueing System (NQS). The Paragon system can queue up to 500 requests at any given time.

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.

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 : (colon), then go to step 7.
3. If the first non-whitespace character of the current line is not a # (pound) character, then go to step 8.
4. If the second non-whitespace character in the current line is *not* the @ (at) character, or the character immediately following the second non-whitespace character in the current line is not a $ (dollar sign) character, or if the second non-whitespace character is not a Q character followed immediately by the string "SUB", then go to step 7.
5. If no - is present as the character immediately following the "@$" sequence or the "QSUB" sequence, then go to 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. Go to 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:
#
#  @$-a "11:30pm EDT" -lt "21:10, 20:00"
#                 # Run request after 11:30 EDT by default,
#                 # and set a maximum per-process CPU time
#                 # limit of 21 minutes and ten seconds.
#                 # Send a warning signal when any process
#                 # of the running batch request consumes
#                 # more than 20 minutes of CPU time.
#  QSUB-lT 1:45:00
#                 # Set a maximum per-request CPU time limit
#                 # of one hour, and 45 minutes.  (The
#                 # implementation of CPU time limits is
#                 # completely dependent upon the NQS
#                 # implementation at the execution
#                 # machine.)
#  QSUB-mb -me
#                 # Send mail at beginning and end of
#                 # request execution.
#  @$-q batch1 # Queue request to queue: batch1 by
#                 # default.
#  @$             # No more embedded flags.
#
make all
```

Queue Types

NQS supports both *batch* queues and *pipe* queues. A batch queue holds requests for scheduled or delayed processing. A pipe queue is a queue that can pass queued requests on to batch queues or other pipe queues.

Pipe queues are used to send NQS requests to other pipe queues or to batch queues. In general, pipe queues act as the mechanism that NQS uses to transport batch requests to queues on other remote machines. Pipe queues can also 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 request to pass through many pipe queues on its way to its ultimate destination, which must eventually be a batch queue.

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 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.

If a pipe queue server cannot handle a request, the request is deleted and the user is notified by mail (see **mail**(1)).

Queue Access

NQS supports queue access restrictions. For each queue, 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. Requests submitted by root are an exception; they are always queued, even if root has not explicitly been given access.

Use **qstat** 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 queue. 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.

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

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

White space can appear anywhere between the principal tokens, with the exception that no white space 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. 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.

If a batch request specifies a limit that cannot be enforced by the underlying operating system executing the job, 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.

For each remaining finite limit that can be supported by the underlying operating system that is not a CPU time limit, or 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 operating systems reserve coefficients of the form: 2^N-1 as synonymous with infinity, meaning no limit is to be applied. For such 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 configured for a particular batch queue using the **qmgr** 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** 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** commands that alter the limits of the containing batch queue.

Batch Request Sequence

The following events take place in the following order when an NQS batch request is spawned:

1. The process that will become the head of the process group for all processes comprising the batch request is created by NQS.

2. Resource limits are enforced.

3. 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.

4. The real and effective user-id of the process is set to the real user-id of the batch request owner.

5. 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.

6. If 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).

7. 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.

8. The environment variable ENVIRONMENT is set to BATCH. This allows shell scripts and the user's *.profile*, *.cshrc*, or *.kshrc* and *.login* files to test for batch request execution when appropriate, and not set terminal characteristics since a batch request is not connected to an input terminal. For example, you could put something similar to the following example in your *.cshrc* file:

```
if ($?ENVIRONMENT == 0) then
     # ENVIRONMENT is not set, must not be running under NQS
     setenv ENVIRONMENT INTERACTIVE
endif

if ("$ENVIRONMENT" != "BATCH") then
     # interactive shell
     setenv HOME /tmp
else
     # NQS shell
     date
endif
```

9. 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.*

10. 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.

11. 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.

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). If the shell strategy is free, NQS executes 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.

If the *use_login* configuration parameter is set to `1`, 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* files are read. If the Korn shell is chosen, then both the *.profile* and *.kshrc* files are read.)

Long Description of Parameters

-a *date-time*

Currently unused. Do not run the batch request before the specified date and/or time. If a *date-time* specification contains any 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. 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 time zone 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 Wednesday.

Some valid date-time examples are:

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

The **-a** switch is also used to make a reservation for a dedicated job if the queue to which the batch request is to be submitted is designated "dedicated." A queue is designated "dedicated" if the queue priority is greater than or equal to the *tsched_pri* value defined in the */usr/spool/nqs/conf/sched_param* file. In addition, the starting time specified by the -a switch and the time limit (specified with the **-lT** switch) must be specified in 30 minute increments for dedicated job reservation because dedicated times are allocated at 30 minute intervals.

-c *account*

An integer accounting ID or a character string representing the MACS accounting group name may be specified for *account*. If this option is not specified, the account ID in the *ACCOUNT* environment variable is used. If the *ACCOUNT* environment variable is not set, the default specified in */etc/nx/nx_dflt_accts* file is used. Accounting IDs and group names are defined in the */etc/nxaccount* file.

This flag is unique to the Paragon system, and as such, is not supported by other NQS implementations. For this reason, if a batch job is submitted from a remote system (including another Paragon system), this flag has no effect. In these cases, the account ID should be specified through the *ACCOUNT* environment variable and then exported with the **-x** option.

-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.*

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 is 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.

qsub *(cont.)* qsub *(cont.)*

-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** flag is 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 corefile 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 corefile 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 operating system.

See the "Limits" discussion on page 6-68 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.*

(This flag is not supported on the Paragon system, but can be used when submitting NQS jobs from the Paragon system to other systems that do support the flag.)

-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 may be terminated by a signal chosen by the underlying operating system. For OSF/1, memory allocation calls will fail and no signal will be sent.

The ability to specify an optional warning limit exists for those operating systems that support per-process data-segment warning size limits. When a warning limit is exceeded, a signal as determined by the underlying operating system is delivered to the offending process (OSF/1 does not support a warning limit).

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 NQS 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 "Limits" discussion on page 6-68 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.*

(This flag is not supported on the Paragon system, but can be used when submitting NQS jobs from the Paragon system to other systems that do support the flag.)

-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 operating system. For OSF/1, this is SIGXFSZ.

The ability to specify an optional warning limit exists for those operating systems that support per-process warning permanent-file size limits. When a warning limit is exceeded, a signal, as determined by the underlying operating system, is delivered to the offending process. For OSF/1, this is SIGXFSZ.

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 NQS 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.

For all NQS implementations that do not support a distinction between permanent, and temporary files at the kernel level (such as OSF/1), this limit is interpreted as a per-process file size limit, with the word *permanent* removed from the definition.

See the "Limits" discussion on page 6-68 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.*

(This flag is not supported on the Paragon system, but can be used when submitting NQS jobs from the Paragon system to other systems that do support the flag.)

qsub *(cont.)*

-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. 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 operating system implementation.

The ability to specify an optional warning limit exists for those 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 operating system 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 NQS 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.

For all operating systems 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.

(This flag is not supported on the Paragon system, but can be used when submitting NQS jobs from the Paragon system to other systems that do support the flag.)

See the "Limits" discussion on page 6-68 for more information on the implementation of batch request limits, and for a description of the precise syntax of *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. 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 operating system.

The ability to specify an optional warning limit exists for those operating systems that support per-process warning memory size limits. When a warning limit is exceeded, a signal, as determined by the operating system, 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 NQS 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 "Limits" discussion on page 6-68 for more information on the implementation of batch request limits, and for a description of the precise syntax of *per-process memory size limit.*

(This flag is not supported on the Paragon system, but can be used when submitting NQS jobs from the Paragon system to other systems that do support the flag.)

-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. 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 operating system.

The ability to specify an optional warning limit exists for those 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 operating system.

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 operating systems 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 "Limits" discussion on page 6-68 for more information on the implementation of batch request limits, and for a description of the precise syntax of *per-request memory space limit.*

(This flag is not supported on the Paragon system, but can be used when submitting NQS jobs from the Paragon system to other systems that do support the flag.)

-ln *per-request nice value limit*

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

Most NQS 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.

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 operating systems 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 NQS 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.

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 "Limits" discussion on page 6-68 for more information).

-lP *node-number*

Set a per-request limit on the number of nodes used. This option is used to set a node limit for each parallel application within an NQS job that is less than or equal to the limit associated with a batch queue. In the absence of this flag, parallel applications of an NQS job will run in the number of nodes specified as an attribute of the NQS batch queue.

This flag is unique to the Paragon system, and as such, is not supported by other NQS implementations. For this reason, if a batch job is submitted from a remote system (including another Paragon system), this flag has no effect. In these cases, a node limit can be specified through the *NCPUS* environment variable and then exported with the **-x** option.

-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 operating system.

The ability to specify an optional warning limit exists for those operating systems that support per-process warning stack-segment size limits. When a warning limit is exceeded, a signal as determined by the underlying operating system 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 operating systems 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 "Limits" discussion on page 6-68 for more information on the implementation of batch request limits, and for a description of the precise syntax of *per-process stack-segment size limit.*

(This flag is not supported on the Paragon system, but can be used when submitting NQS jobs from the Paragon system to other systems that do support the flag.)

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

Set a per-process CPU time limit 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 node hour limit for the request, then that process is terminated by a signal chosen by the underlying operating system. For OSF/1, this is SIGXCPU.

The ability to specify an optional warning limit exists for those operating systems that support per-process CPU warning time limits. When a warning limit is exceeded, a signal, as determined by the underlying operating system is delivered to the offending process. For OSF/1, this is SIGXCPU.

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 NQS 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 "Limits" discussion on page 6-68 for more information on the implementation of batch request limits, and for a description of the precise syntax of *per-process CPU time limit.*

(This flag is not supported on the Paragon system, but can be used when submitting NQS jobs from the Paragon system to other systems that do support the flag.)

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

Sets a per-request maximum node hour limit and an optional warning limit for a batch request. If the node hour limit is exceed, all of the processes in the request are sent a SIGTERM signal, and then a SIGKILL signal after the number of seconds specified by the *grace_time* configuration parameter in the *sched_param* file (10 seconds by default). If the optional warning limit is specified, the user is notified before the processes are terminated.

If the node hour limit (and optional warning limit) is comprised of two or more tokens separated by whitespace characters, they 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.

qsub *(cont.)* qsub *(cont.)*

While the Paragon system supports this limit, not all NQS 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 "Limits" discussion on page 6-68 for more information on the implementation of batch request limits, and for a description of the precise syntax of *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. 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 operating system.

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

See the "Limits" discussion on page 6-68 for more information on the implementation of batch request limits, and for a description of the precise syntax of *per-process temporary-file size limit.*

(This flag is not supported on the Paragon system, but can be used when submitting NQS jobs from the Paragon system to other systems that do support the flag.)

qsub *(cont.)* qsub *(cont.)*

-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. 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 operating system implementation.

The ability to specify an optional warning limit exists for those 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 operating system 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.

See the "Limits" discussion on page 6-68 for more information on the implementation of batch request limits, and for a description of the precise syntax of *temporary-file space limit.*

(This flag is not supported on the Paragon system, but can be used when submitting NQS jobs from the Paragon system to other systems that do support the flag.)

-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 operating systems 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 "Limits" discussion on page 6-68 for more information on the implementation of batch request limits, and for a description of the precise syntax of *per-process working set size limit.*

(This flag is not supported on the Paragon system, but can be used when submitting NQS jobs from the Paragon system to other systems that do support the flag.)

-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 **-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 shut down via an operator command to the **qmgr** 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 60, but this value can be overridden by the operator), all remaining processes comprising all remaining running NQS requests are killed by the SIGKILL signal.

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. 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" discussion on page 6-92 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 of the named [*machine*:][[/]*path/*]*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.

-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 queue specified by the environment variable *QSUB_QUEUE* is used. 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 error message is displayed to this effect.

qsub *(cont.)*   qsub *(cont.)*

-r *request-name* Assign the specified *request-name* to the request. In the absence of an explicit **-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 written to 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.

-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.

-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 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.
- A shell strategy of login means that the user's login shell is used to execute the batch request.

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** command.

qsub *(cont.)*

-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.

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** 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** 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 operating systems that provide a mechanism of locking a process, and all of its subsequent children in a particular process-group. For such operating systems, 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** subcommand issued by an operator using the **qmgr** program, or because it is necessary to shut down 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 shell, Korn shell, or C-shell is displayed. This is merely the value returned by the shell through the **exit** (2) system call.

qsub *(cont.)* qsub *(cont.)*

If the **qsub** command returns the error:

```
Explicit request quota limits exceed maximums at local host.
The request could not be queued or delivered because the
following explicit request quota limit(s) exceeded the
corresponding limits of the target queue:
  Per-request CPU quota limit;
```

This error message indicates that the request has either exceeded the queue's CPU time limit (set by the **qmgr** subcommand **set per_request cpu_limit**) or the queue's number-of-CPUs limit (set by the **qmgr** subcommand **set per_request ncpus**). These two errors result in the same error message because they are represented internally by a single error code; this is necessary in order to maintain compatibility with other versions of NQS.

If the qsub command returns the error:

```
Permission denied
for a queue.
```

Check the setting of the **qmgr set queue_request_limit** subcommand.

See Also

mail, qdel, qdev, qlimit, qmgr, qpr, qstat, kill, setpgrp(), signal()

que_update

Reads the R1.3 queue database files and writes new files in a format compatible with R1.4.

Synopsis

/usr/lib/nqs/que_update

Discussion

The **que_update** command reads the pre-R1.4 queue database files and creates new database files with the same queue information in a format compatible with R1.4. The new files and directories are created with extension *.new*; the old files and directories are left in place.

The NQS setup script will run **que_update** if necessary (with user approval).

NOTE

que_update is normally called by the **nqs_setup** or **nqs_queue_upgrade** script. If **que_update** is run by hand, the old queue database files must be backed up by hand first and the new files installed by hand afterwards. (See the Files section for a list of the files and directories affected.)

Files

/usr/spool/nqs/private/root/queues	Old queue database
*/usr/spool/nqs/private/root/database_qa/**	Old queue access files
*/usr/spool/nqs/private/root/database_qo/**	Old queue order files
/usr/spool/nqs/private/root/queues.new	New queue database
*/usr/spool/nqs/private/root/database_qa.new/**	New queue access files
*/usr/spool/nqs/private/root/database_qo.new/**	New queue order files

See Also

nqs_queue_upgrade, qcmplx, qdel, qdev, qlimit, qmgr, qpr, qsub, qstart

nqs_queue_upgrade nqs_queue_upgrade

Upgrades the R1.3 NQS queue database files "in place" to a new format compatible with R1.4.

Synopsis

/usr/lib/nqs/setup/nqs_queue_upgrade

Discussion

The **nqs_queue_upgrade** script is a simplified front end to the **que_update** command. It backs up your pre-R1.4 NQS queue database files, runs **que_update**, then installs the updated files if **que_update** was successful. In effect, it replaces your pre-R1.4 queue database files "in place" with new database files containing the same queue information in a format compatible with R1.4. The old files and directories are saved with extension *.bak*. The NQS setup script will run **nqs_queue_upgrade** if necessary (with user approval).

CAUTION

You must stop NQS (/sbin/init.d/nqs stop) before running **nqs_queue_upgrade**. If **nqs_queue_upgrade** is run while NQS is running, the queue database files could become corrupted.

Files

/usr/lib/nqs/que_update	Command that performs the actual update
/usr/spool/nqs/private/root/queues	Queue database file
*/usr/spool/nqs/private/root/database_qa/**	Queue access files
*/usr/spool/nqs/private/root/database_qo/**	Queue order files
/usr/spool/nqs/private/root/queues.new	New queue database (temporary)
*/usr/spool/nqs/private/root/database_qa.new/**	New queue access files (temporary)
*/usr/spool/nqs/private/root/database_qo.new/**	New queue order files (temporary)
/usr/spool/nqs/private/root/queues.bak	Old queue database
*/usr/spool/nqs/private/root/database_qa.bak/**	Old queue access files
*/usr/spool/nqs/private/root/database_qo.bak/**	Old queue order files

See Also

que_update

sched_param sched_param

NQS configuration file.

Synopsis

/usr/spool/nqs/conf/sched_param

Description

NQS is configured by the configuration file */usr/spool/nqs/conf/sched_param* that contains tunable parameters. NQS managers can change these parameters to tailor NQS to their environment.

All configuration parameters take effect immediately after NQS startup. Some are also scanned each time NQS initiates a scheduling event (whenever someone submits a batch job, a batch job finishes, or every 15 minutes if there is no NQS activity). This allows the NQS manager to change some parameters without taking NQS down. If a parameter appears more than once in the configuration file, the last occurrence takes precedence. The following list describes the NQS parameters. An asterisk (*) indicates NQS must be restarted for any new parameter values to take effect.

node_set*
: A *node set* is a rectangular grouping of nodes. Node sets are used to create batch partitions of varying sizes and to group together nodes of similar attributes (groups of 16MB and 32MB nodes, for example). Node sets also play a role in switching between prime and non-prime time. There is no default. You must define at least one node set.

node_group*
: A *node group* is a collection of one or more node sets. Each NQS batch queue is associated with a single node group using the **qmgr set node_group** subcommand. There is no default. You must define at least one node group.

prime_list*/nprime_list*
: The *prime_list* defines which node sets in a given node group are to be used during prime time. The *nprime_list* defines which node sets are used during non-prime time. A node set can be assigned to both the *prime_list* and *nprime_list* of a node group, meaning it will be used during both prime and non-prime time. There is no default. You must supply values for each if your site uses a prime/non-prime time scheme.

cur_open_name*/open_p_name*/open_np_name*
: Define the partition names of the open prime and open non-prime partitions, and are used when creating a prime time/non-prime time scheduling scheme. If you specify one of these parameters, you must specify all three. There is no default. You must supply a value if your site uses a prime/non-prime time scheme.

sched_param *(cont.)*

cur_open_epl*/ncur_open_epl*
: Define the effective priority level (EPL) for the open partition. *cur_open_epl* defines the EPL for the active open partition (named by *cur_open_name*). *ncur_open_epl* defines the EPL for the inactive open partition (named by *open_np_name* during prime time, and *open_p_name* during non-prime time). The EPL values must be between `0` and `10`, inclusive. The default EPL for *cur_open_epl* is `3`; The default EPL for *ncur_open_epl* is `1`.

prime_start/prime_end
: Define the beginning and ending times for prime time and non-prime time. Each parameter requires a list of seven input values separated by spaces, one for each day of the week, with the first element specifying the value for Sunday. Fractional hours are specified in decimal increments. The default *prime_start* value is `08` (8 a.m.) and the *prime_end* value is `18` (6 p.m.).

prime_script/nprime_script
: Specify the full pathname of an executable file (typically a shell script) which is run at the beginning of prime time (prime_script) or non-prime time (nprime_script). The script is run after all other processing has been performed for the prime/non-prime switchover. The default value is blank, meaning no scripts are run.

do_wall
: Defines whether a broadcast message will be sent to all logged in users during a prime/non-prime time switch. If *do_wall* is set to `0`, no message is sent. If *do_wall* is set to `1`, a message is sent. The default is `0`.This parameter is ignored if *cur_open_name*, *open_p_name*, and *open_np_name* are not defined.

macs_flag
: Allows MACS to distinguish between NQS batch jobs and interactive jobs. If MACS is running and *macs_flag* is set to `0`, all jobs recorded in the MACS reports appear as interactive jobs. If MACS is running and *macs_flag* is set to `1`, NQS jobs are differentiated from interactive jobs. If MACS is not running and *macs_flag* is set to `1`, all NQS jobs will fail. The default is `0`.

age_factor
: A floating point value used to calculate the dynamic priority of a job. The default is `0.1`.

block_pri
: Makes it easier for large jobs to get scheduled. When a job reaches the top of the queue and if there is no room in the base layer to schedule it, and if the static inter-queue priority for this job is great than or equal to the *block_pri* value, the scheduling of jobs behind it is blocked until the top job can be scheduled.

: When this feature is disabled with `-1`, NQS will tend to run smaller jobs instead of larger jobs if they fit the node availability better. The default is `-1`.

sched_param *(cont.)* sched_param *(cont.)*

chk_runlimit	Defines whether NQS *run limits* are checked before starting a job. The run limit determines the maximum number of jobs that are allowed to run globally, in a queue, or in a complex. If set to `1`, the queue and complex run limits are enforced; if set to `0`, the run limits are not enforced. The default is `0`.
grace_time	The grace period, in seconds, after which a batch request is killed if it exceeds its run time limit. The default is `10` seconds
time_zone*	Defines the time zone used by NQS. The string format follows the Paragon OSF/1 system convention. For example, Pacific time is indicated as `PST8PDT`. The default is the value set for the *TZ* environment variable.
np_overrun	Determines if NQS will allow non-prime time jobs to run over into prime time on nodes designated as non-prime time only. If *np_overrun* is set to `0`, NQS will allow non-prime time jobs to run over into prime time. If *np_overrun* is set to `1`, NQS will not schedule jobs if the requested time plus the current time will overrun into prime time. The default is `0`.
tsched_pri	Currently unused. The default is `20`.
timesh_pri	Currently unused. The default is `10`.
timeshare	Currently unused. The default is `0` (leave default).
rollin_quan	The rollin quantum is the time, in seconds, a batch job can execute before being rolled out into a time-sharing scheme. This value must be greater than the value set by MIN_RQ_ALLOWED= in the */etc/nx/allocator.config* file. The default is `600` seconds. This default is incompatible with the allocator default rollin quantum (one hour) set in the *allocator.config* file. This setting should be set to a value greater than that set for the allocator.
nosched	Prevents NQS from scheduling requests. If *nosched* is set to `1`, NQS will not schedule any jobs. If set to `0` or omitted, NQS schedules jobs normally. The *nosched* parameter is typically used to prevent NQS from rescheduling a job that has crashed the system. The default is `0`.
use_login	Determines how the user's shell is spawned. If *use_login* is set to `1`, NQS spawns the user's shell as a login shell, causing the shell to read its *.profile* or *.cshrc* and *.login* files as it starts up. If *use_login* is set to `0` or omitted, the user's shell is spawned as a non-login shell. The default is `0`. The **qmgr** subcommand **set shell_strategy** determines which shell is actually spawned.

sched_param *(cont.)* sched_param *(cont.)*

wallclock_limits

Determines if the CPU time limit for a queue or request is based on node-hours or wall-clock hours. If set to `1`, the CPU limit is enforced as a wall-clock limit; if set to `0`, the CPU limit is enforced as a node-hour limit.This setting affects all queues. The default value is `0`.

The *wallclock_limits* configuration parameter affects the CPU limits set with the **qmgr set per_request cpu_limit** subcommand and the CPU limits specified by the user with the **qsub -lT** flag.

In a node-hour configuration (*wallclock_limits* set to `0`), if the request uses fewer nodes than those reserved by the queue, the node-hours for that queue are redistributed among the nodes actually used. This means that the request could run longer than the per-request time limit for the queue. For example, an application can use fewer nodes than those reserved for the queue if the application uses the *NX_DFLT_SIZE* environment variable and specifies fewer nodes, or is invoked with the **-sz** switch specifying fewer nodes.

node_base	In the calculation to determine dynamic priority, used to determine the node priority of a job.
node_weight	In the calculation to determine dynamic priority, determines how much weight is given to the node priority when calculating the priority of a job.
time_base	In the calculation to determine dynamic priority, used to determine the time priority of a job.
time_weight	In the calculation to determine dynamic priority, determines how much weight is given to the time priority when calculating the priority of a job.

NOTE

If your site uses gang scheduling, the *wallclock_limits* setting should be 0. This will specify CPU limits based on node-hours, which will allow jobs that were rolled out for a time to finish running.

See Also

qstart

Index