Chapter 2

Configuring and Managing NQS

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

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

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

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

2.1 Starting Up NQS

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

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

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

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

2.2 Configuring NQS — The qmgr Utility

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

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

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

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

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

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

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

2.3 Who Can Issue qmgr Commands

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

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

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

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

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

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

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

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

2.4 Configuring Batch Queues — An Overview

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

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

After each command, the system responds:

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

Let's look in more detail at the commands.

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

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

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

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

```
enable queue cmq1
```

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

```
exit
```

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

2.5 Creating, Enabling, and Starting a Batch Queue

There are three steps to making a batch queue operational:

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

2.5.1 The create batch_queue Command

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

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

For example,

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

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

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

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

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

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

2.5.2 The enable queue Command

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

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

2.5.3 The start queue and set restriction_window Commands

There are two commands that start a queue:

- Issue the **start queue** command (minimum abbreviation: **sta q**) to start the queue immediately and make requests eligible to run; use this command when the queue is to operate continuously.

- Use the **`set restriction_window`** command (minimum abbreviation: **`se rest`**) to set the times or dates when the queue is to start and stop. The queue starts automatically when the specified starting time arrives, and stops when the specified stopping time arrives. Its arguments are described below.

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

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

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

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

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

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

  For example,

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

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

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

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

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

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

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

Deleting a Restriction Window

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

```
Mgr: delete restriction_window cmq1
```

deletes the restriction window currently in effect for queue **`cmq1`**. You don't have to issue this command before creating a new restriction window; a new window automatically supersedes the window previously in effect. Operator privileges are required to issue this command.

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

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

2.6 Stopping, Disabling, and Deleting a Queue

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

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

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

In addition, an NQS operator can do the following:

- Issue the **`purge queue`** command (minimum abbreviation: **`pu q`**) to purge all requests from a specified batch queue. Requests that are currently running are allowed to complete; all other requests are lost and can't be recovered.
- Issue the **`abort queue`** command (minimum abbreviation: **`ab q`**) to abort all requests that are currently running in the specified queue. The command sends a SIGTERM signal to each running process. After a specified number of seconds (the default is 60), the command sends a SIGKILL signal to any remaining processes that are still running. For example,

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

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

2.7 Who Can Use a Queue?

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

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

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

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

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

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

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

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

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

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

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

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

```
Mgr: set unrestricted_access cmq1
```

2.8 Setting Limits

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

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

- **Core file size —** By default, the maximum per-process corefile size is unlimited. Use the **set corefile_limit** command (minimum abbreviation: **se cor**) to specify a limit.
- **CPU time limit —** By default, the maximum amount of front-end CPU time a process can use is 36000 seconds. Use the **set per_process cpu_limit** command (minimum abbreviation: **se per_p c**) to specify a different limit.
- **Data size —** By default, there is no maximum per-process data segment size. Use the **set data_limit** command (minimum abbreviation: **se da**) to specify a limit.
- **File size —** By default, the maximum per-process permanent file size on the partition manager is 1 megabyte. Use the **set per_process permfile_limit** command (minimum abbreviation: **se per_p p**) to specify a different limit for processes running in a specified batch queue.
- **Nice value —** By default, the nice value for a process is 0. Use the **set nice_value_limit** command (minimum abbreviation: **se ni**) to specify a different default. Nice values are integers in the range –20 to 20; a lower number specifies a higher execution priority.
- **Stack size —** By default, the maximum per-process stack segment size is 6 megabytes. Use the **set stack_limit** command (minimum abbreviation **se st**) to specify a different limit.

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

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

NQS manager privileges are required to issue these commands.

2.9 Creating Queue Complexes

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

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

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

For example,

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

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

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

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

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

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

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

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

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

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

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

2.10 Pipe Queues — An Overview

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

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

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

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

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

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

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

1. If NQS has not been installed on the computer on which you want to create the pipe queue, you must first install it.
2. As part of installing NQS, you must run the **`nmapmgr`** utility on the computer on which the pipe queue is to run, and on all computers with which the pipe queue is to communicate. This utility creates a database of mappings between computers. See Section 2.12.6.
3. You then use the **`qmgr`** utility as you do for batch queues, to create, enable, and start the pipe queue. See Section 2.11.

2.11 Creating, Configuring, and Managing Pipe Queues

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

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

2.11.1 An Example

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

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

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

2.11.2 Creating a Pipe Queue: The create pipe_queue Command

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

The following arguments are required:

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

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

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

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

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

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

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

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

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

The following arguments are optional:

- **pipeonly** (minimum abbreviation: **pi**). This argument specifies that this pipe queue is to accept only requests coming from other pipe queues; users will not be allowed to submit requests directly to this queue.

- **run_limit** (minimum abbreviation: **r**). The run limit sets a ceiling on the number of requests allowed to run in the pipe queue at any given time. If you omit this argument, the default run limit is 1.

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

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

2.11.3 Configuring and Managing Pipe Queues

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

- To specify who can use a queue, you can issue the **set no_access**, **set unrestricted_access**, **add users**, and **add groups** commands, as discussed in Section 2.7.
- To enable the queue, use the **enable queue** command discussed in Section 2.5.
- To start the queue, use the **start queue** command, as discussed in Section 2.5. Note that you can't use the **set restriction_window** command to start a pipe queue.
- Use the various commands listed in Section 2.6 for stopping, disabling, and deleting a pipe queue, and for purging and aborting requests in a pipe queue.

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

2.12 Managing NQS

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

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

2.12.1 Shutting Down NQS

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

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

```
shutdown 120
```

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

An NQS operator can issue this command.

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

2.12.2 Creating a Default Batch Queue

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

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

makes **cmq1** the default batch queue.

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

2.12.3 Determining What Shell Is to Be Used

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

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

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

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

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

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

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

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

2.12.4 Setting the Default Intraqueue Priority

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

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

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

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

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

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

2.12.5 Logging Error Output

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

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

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

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

0	No debug
1	Minimum debug
2	Maximum debug

By default, the debugging level is set to 0.

2.12.6 Running the nmapmgr Utility

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

The Machine ID

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

```
% nmapmgr
```

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

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

nmapmgr responds with the machine ID of the computer.

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

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

Adding an Entry to the nmapmgr Database

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

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

```
# nmapmgr
```

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

```
NMAPMGR>: create
```

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

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

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

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

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

```
NMAPMGR>: exit
#
```

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

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

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

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

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

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

2.13 Obtaining Information

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

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

Table 3. qmgr show commands.

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

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

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

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

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

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

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

2.13.1 Obtaining Accounting Information

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

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

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

/*
 *      NQS account file record header.
 */
struct  nqsacct {
        int     type;                   /* type of record */
```

```
        int     length;                 /* length of record (bytes) */
        int     jobid;                  /* UNICOS job id */
};

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

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

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

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

Note these points in interpreting the records:

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