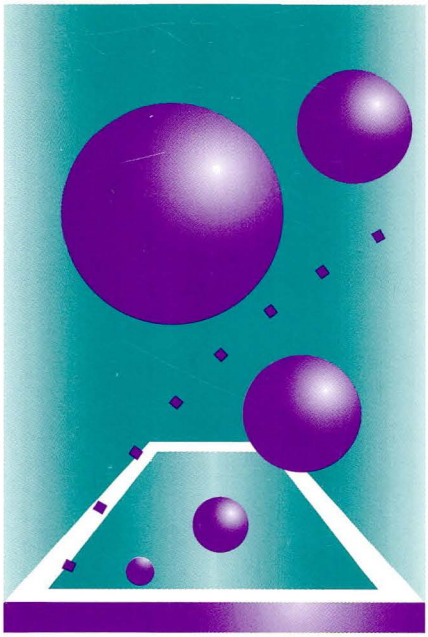


CONVEX

ConvexNQS+ User's Guide for Exemplar Systems

First Edition

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

ConvexNQS+ User's Guide for Exemplar Systems

Order No. DSW-861

First Edition
August 1994

CONVEX Press
Richardson, Texas
United States of America

ConvexNQS+
User's Guide
for Exemplar systems

—PRELIMINARY—

Order No. DSW-861

Printed in the United States of America

Revision information for

ConvexNQS+ User's Guide for Exemplar systems

Edition	Document No.	Description
First	770-007530-000	Released for ConvexNQS+ software V2.0 for use with CONVEX Exemplar systems, August 1994.

Contents

Figures

Tables

Using this book

Purpose and audience

The *ConvexNQS+ User's Guide for Exemplar Systems* includes information on the following topics:

- Basic ConvexNQS+ concepts
- How to submit requests
- How to manipulate requests

Organization

This guide addresses the needs of ConvexNQS+ users.

Specifically, this guide is organized into the following chapters and appendixes:

- Chapter 1 introduces general concepts necessary to using the software
- Chapter 2 describes how to display queue and request information
- Chapter 3 describes how to submit a request
- Chapter 4 describes how to manipulate a request
- Appendix A and Appendix B list transaction completion and request completion codes, respectively

Conventions

This section discusses notational conventions used in this book.

Command syntax

Consider this example:

```
COMMAND input_file [...] {a | b} [output_file]
  ①       ②      ③     ④       ⑤
```

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

General conventions

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

- *Italics*:
  - Designate user-supplied variables in a command-line example
  - Introduce new and important terms
  - Identify variables in mathematical equations
  - Indicate document titles
- `Constant-width font` designates:
  - System output in screens and examples
  - Command names and options
  - System calls
  - Data structures and types
  - Directives, program statements, display examples, printout examples, and error messages returned
- **`Bold, constant-width font`** designates user input in screens and examples, and must be typed exactly as it appears.
- Horizontal ellipsis (...) shows repetition of the preceding item(s).
- Vertical ellipsis shows that lines of code have been left out of an example.

- Words and abbreviations that indicate keyboard keys you press are identified in a distinctive bold type. For example, **RETURN** refers to the carriage return key. Words separated by a hyphen indicate two keys that you must press simultaneously. For example, **CTRL-X** indicates that you must press and hold down the **CTRL** key and then press the **X** key.
- The word "enter" in a phrase such as "enter `ls`" means that you type the command and then press **RETURN**.
- References to man pages appear in the form adb(1), where the name of the man page is followed by its section number enclosed in parentheses.

Note

A note highlights supplemental information.

Caution

A caution highlights procedures or information necessary to avoid damage to equipment, damage to software, loss of data, or invalid test results.

Associated documentation

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

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

- *ConvexNQS+ System Manager's Guide for Exemplar Systems* (DSW-860). This book describes how to configure a ConvexNQS+ network and how to maintain queues and requests on a regular basis.
- *CONVEX COVUEbatch Guide* (DSW-151). This book describes how to use and maintain the COVUEbatch batch processing software.
- *ConvexOS Extensions User's Guide* (DSW-053). This book describes ConvexOS utilities from the user's perspective.

Ordering documentation

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

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

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

Technical assistance

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

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

ConvexNQS+ 1

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

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

Queue processing

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

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

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

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

ConvexNQS+ queues

There are two types of ConvexNQS+ queues:

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

ConvexNQS+ pipe clients

There are three types of ConvexNQS+ pipe clients:

- pipeclient
- pipedemand
- pipeldav

Pipeclient routes a request to the first queue in its destination set that is able to accept the job.

Destinations may reject the request due to queue limit violations, lack of account authorization, and many other reasons.

Pipedemand routes a request to the first queue in its destination set that can immediately run the request.

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

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

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

Pipeldav sorts its destination set by load factor and routes a request to the destination with the lowest load factor that is able to accept the job.

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

To route requests based on load factor, pipeldav

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

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

  A ConvexNQS+ manager sets the weighting factor as an option of the pipeldav program; the weighting factor defaults to 1.0.
- Routes the request to the destination queue—with the lowest load factor—that is able to accept the job.

ConvexNQS+ configuration and control utilities

There are two ConvexNQS+ utilities for configuring and operating ConvexNQS+:

- `qmapmgr`
- `qmgr`

Both utilities are discussed in the following sections.

qmapmgr utility

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

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

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

qmgr utility

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

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

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

Levels of authorization

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

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

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

Batch queue accounting

SPP-UX accounting systems track the system resources used by an individual user or group by *process*. ConvexNQS+ batch accounting tracks the system resources used by an individual user or group by *request*.

Implementing ConvexNQS+ batch accounting involves identifying a log file to collect ConvexNQS+-specific accounting data, and then enabling batch accounting on a per-batch-queue basis.

CONVEX Share Scheduler

CONVEX Share Scheduler, an optional CONVEX product, is a per-user process scheduler that operates with the standard process scheduler. It equitably allocates machine resources among individual users and groups according to their allocation of shares.

Differences between ConvexNQS+ and NQS

In addition to the enhancements detailed in the "Platform-specific features" section, there are five major differences between ConvexNQS+ on Exemplar systems and other NQS systems.

Understanding the following differences is important if you combine several different systems at one site:

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

Table 1 lists packet numbers recognized only by ConvexNQS+.

Table 1 Packet numbers recognized only by ConvexNQS+

Packet Number	Type	Use
200	nqs	Set queue import attribute
201	nqs	Set queue description
203	nqs	Set queue accounting on or off
204	nqs	Set accounting log file
206	nqs	Queue request from remote qsub
207	nqs	Get sequence number
208	nqs	Hold request
209	nqs	Release request
210	nqs	Add queue alias
211	nqs	Delete queue alias
212	nqs	Ping with ack (used for debugging)
213	nqs	Ping without ack (used for debugging)
214	nqs	Set maximum request priority
218	nqs	Force request to run
219	nqs	Suspend request
220	nqs	Resume request
223	nqs	Force run request
224	nqs	Set global per-user-run-limit
225	nqs	Set queue per-user-run-limit
226	nqs	Restart request
228	nqs	Set copy-open-files on checkpoint
205	net	Get remote queue and request information

Displaying information 2

This chapter describes how to display information about queues and queue requests using `qstat`, `qwatch`, `qlimit`, `qps`, and `qjlist`, the ConvexNQS+ commands explained below.

- `qstat` displays the status of a specified ConvexNQS+ queue(s). The specified queue(s) can reside on a local or remote machine
- `qwatch` interactively displays the status of ConvexNQS+ queues that reside on the local machine
- `qlimit` displays the batch queue resource limits supported by the operating system on a specified machine(s). The specified machine(s) can be local or remote
- `qps` displays the status of ConvexNQS+ processes associated with batch queues that reside on the local machine
- `qjlist` displays the contents of a specified ConvexNQS+ request. The specified request can originate from a local or remote machine

This chapter describes how to use each of these commands.

Displaying queue status

You can display information about ConvexNQS+ queues and queue requests using the `qstat` command.

The command format is

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

where

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

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

`-l` Displays additional information about the queue requests.

`-m` Displays the date and time requests are scheduled to run.

`-u` *username*
Displays only those requests belonging to the specified *username*.

`-x` Displays additional information about the queue.

queuename is the name of the queue for which you want status information. If you omit *queuename*, ConvexNQS+ assumes all queues on the requested machine.

hostname is the name of the machine that receives the request. If you omit *hostname*, ConvexNQS+ assumes the local machine.

The following sections describe the output in more detail. For more information, refer to the `qstat`(1) man page.

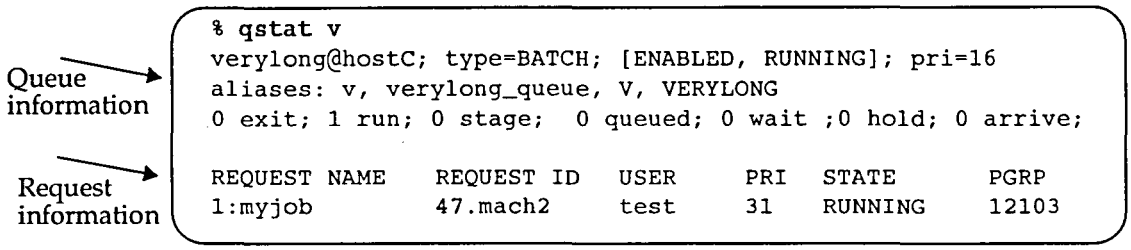
Displaying standard output

Use the `qstat` command with no specified options to display two types of information:

- Information about a queue(s)
- Information about each request in a queue(s)

Figure 1 illustrates the output for the `qstat` command with no specified options.

Figure 1 `qstat` sample standard output



The first line of the output describes information about this queue:

```
verylong@hostC; type=BATCH; [ENABLED, RUNNING]; pri=16
```

verylong@hostCName of this queue and the machine on which it is configured. This example shows the *verylong* queue on *hostC*.

type= Type of queue. This can be

`BATCH`	Runs ConvexNQS+ requests.
`PIPE`	Routes ConvexNQS+ requests to queues that can run them.

enabled Indicates if this queue can accept requests. This can be

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

running Indicates if this queue can run requests. This can be

`INACTIVE`	Requests in the queue can run; none are running.
`RUNNING`	Requests in the queue can run; some are running.
`STOPPING`	New requests sent to the queue cannot run, but requests that are currently running can complete.
`STOPPED`	Requests in the queue cannot run; none are running.

SHUTDOWN	ConvexNQS+ is not running on the machine.

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

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

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

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

The rest of the output displays status information for each request in this queue:

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

Request name
Name assigned to this request.

Request id	Unique identifier assigned to this request when it is submitted to the queue.
User	User submitting this request.
PRI	Intraqueue priority assigned to this request. This priority affects which request runs next in a queue. Priority can be any number from 0 to 63; 0 is the lowest priority and 63 the highest.

State — State of this request. This can be

ARRIVING	Request is arriving at the queue.
CHECKPOINTED	A failed attempt was made to restart the request after it was checkpointed.
DEPARTING	Request is departing from the queue but has not yet been received by the destination queue.
EXITING	Request has completed running and will exit from the system after ConvexNQS+ returns the required output files to their intended destinations.
HOLDING	A hold has been placed on the request, preventing it from entering any other state.
QUEUED	Request is queued and eligible for running or routing. This is the most common state.
ROUTING	Request has reached the head of a pipe queue and is being routed to another queue.
RUNNING	Request has reached its final destination batch queue and is running.
WAITING	Request is waiting a specified amount before running. This could be because it was submitted with a future run date and time, or because a pipe queue could not route the request and will try to route it later.
SUSPENDED	Request is suspended from running. It will remain suspended until manually resumed.

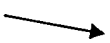
PGRP — Process group of the request, if available to the local ConvexNQS+ daemon. This information is displayed only for processes that are running.

Displaying additional request information

Use the -l option to display additional information about a request(s). Figure 2 illustrates the output for this option.

Figure 2 `qstat -l` option sample output

Additional request information



```
% qstat -l long
Queue for long jobs.
 long@mach2; type=BATCH; [ENABLED, RUNNING]; pri=32
aliases: l, long_queue, L, LONG
 0 exit; 1 run; 0 stage; 0 queued; 0 wait; 0 hold; 0 arrive;

Request 1: Name=STDIN Id=25.mach2
   Owner=test Priority=31 RUNNING Pgrp=4918
Created at Mon Jan 23 16:30:03 CST 1989
Mail = [NONE]
Mail address = test@mach2
Owner user name at originating machine = test
Import directory: Yes
Per-proc. core file size limit= UNLIMITED <DEFAULT>
Per-proc. data size limit= UNLIMITED <DEFAULT>
Per-proc. permanent file size limit= UNLIMITED <DEFAULT>
Per-proc. execution nice priority = 0 <DEFAULT>
Per-proc. stack size limit= UNLIMITED <DEFAULT>
Per-proc. CPU time limit= [600.0, 600.0]<DEFAULT>
Per-proc. working set limit= UNLIMITED <DEFAULT>
Standard-error access mode = SPOOL
Standard-error name = mach2:/mnt/test/STDIN.e272
Standard-output access mode = SPOOL
Standard-output name = mach2:/mnt/test/STDIN.o272
Shell = DEFAULT
Umask = 2
```

`Created at`
: Day, date, and time the request was submitted.

`Mail`
: Indicates if ConvexNQS+ notifies the user of any situation other than an inability to run the submitted shell script. This can be

  `BEGIN` — ConvexNQS+ notifies the user when this request starts running.

  `END` — ConvexNQS+ notifies the user when this request completes running.

`BEGIN, END`	ConvexNQS+ notifies the user when this request starts running and completes running.
`NONE`	ConvexNQS+ notifies user only if it cannot run the submitted shell script

`Mail address`
: Notification destination.

`Owner user name at originating machine`
: User submitting this request.

`Import directory`
: Indicates if mounting the submitter's current working directory is required to process this request.

The next seven lines display the per-process resource limits submitted with this request.

Note

If you use the `qstat -l` command to display information about a request in a pipe queue, the command displays all limits submitted with the request, regardless of enforceability.

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

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

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

`Per-process core file size limit`
: Maximum core file size for any process in the request.

`Per-process data size limit`
: Maximum data segment size for any process in the request.

`Per-process permanent file size limit`
: Maximum permanent file size for any process in the request.

`Per-process execution nice value`
: Minimum nice value for any process in the request.

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

`Per-process stack size limit`
: Maximum stack segment size for any process in the request.

`Per-process CPU time limit`
: Maximum total CPU time for any process in the request.

`Per-process working set limit`
: Maximum amount of physical memory for any process in the request.

`Standard-error access mode`
: Indicates that ConvexNQS+ writes the stderr file to spooling directories while this request runs and then copies it to the intended destination when the request is finished running.

`Standard-error name`
: Name of the file where ConvexNQS+ sends error output for this request.

`Standard-output access mode`
: Indicates that ConvexNQS+ writes the stdout file to spooling directories while this request runs and then copies it to the intended destination when the request is finished running.

`Standard-output name`
: Name of the file where ConvexNQS+ sends standard output for this request.

`Shell` Command interpreter used to interpret script commands for this request.

`Umask` Default umask for this request.

Displaying a future run time

Use the `-m` option to display the date and time a request(s) is scheduled to run. This information is available only for those requests submitted with future run time specifications. Figure 3 illustrates the output for this option.

Figure 3 `qstat -m` option sample output

```
% qstat -m long
Queue for long jobs.
long@mach2; type=BATCH; [ENABLED, INACTIVE]; pri=32
aliases: l, long_queue, L, LONG
0 exit;0 run; 0 stage; 0 queued;1 wait; 0 hold; 0arrive;

Request 1: Name=myjob Id=297.mach2
Owner=test Priority=31 WAITING Wed Jan 25 00:00:00 CST 1989
```


Displaying additional queue information

Use the -x option to display additional information about a queue(s). Figure 4 illustrates the output from this option.

Figure 4 `qstat -x` option sample output

Additional queue information →

```
% qstat -x long
Queue for long jobs.
long@mach2; type=BATCH; [ENABLED, RUNNING]; pri=32
aliases: l, long_queue, L, LONG
0 exit; 1 run; 0 stage; 0 queued; 0 wait; 0 hold; 0
arrive;
Run_limit = 3;
Accounting: Off
Activity ID offset: 0
Maximum request priority : 63
Cumulative system space time = 428.720000 seconds
Cumulative user space time = 202.560000 seconds
Unrestricted access
Import directory: Yes
Share policy fixed = long
Per-process core file size limit = UNLIMITED
Per-process data size limit = UNLIMITED
Per-process permanent file size limit = UNLIMITED
Per-process execution nice value = 0
Per-process stack size limit = UNLIMITED
Per-process CPU time limit = 600.0
Per-process working set limit = UNLIMITED
```

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

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

`Activity ID offset`
: Typically an integer from 1 to 9. ConvexNQS+ adds this number to an activity ID and uses the resulting job ID for accounting/billing purposes (applies to batch queues only).

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

`Cumulative system space time`—for batch queues
: Total amount of system time used to process requests in this batch queue since the queue was created.

`Cumulative system space time`—for pipe queues
: Total amount of system time used to route requests in this pipe queue since the queue was created.

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

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

`Unrestricted`	This queue can accept any request from any submitter.
`Restricted`	This queue can accept only those requests submitted by a specified group(s) or user(s).
`Pipeonly`	This queue can accept requests only from a pipe queue.

`Import directory`
: Indicates if this queue imports the current working directory for a request (mounts the directory on the

machine processing the request) before running the request (applies to batch queues only). This can be

`Yes`	This queue automatically imports the current working directory for any request running in the queue. A user can override this setting for individual requests using the -ni option of qsub.
`Available`	Lets a user specify importation of the current working directory for any request submitted to this queue using the -i option of qsub.
`No`	This queue does not import the current working directory for requests submitted to the queue. Furthermore, this queue rejects requests that require imported directories.

The rest of the output displays the per-process resource limits, if any, configured for this queue (applies to batch queues only).

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

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

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

`Per-process core file size limit`
: Maximum core file size for any process in a running request.

`Per-process data size limit`
: Maximum data segment size for any process in a running request.

`Per-process permanent file size limit`
: Maximum permanent file size for any process in a running request.

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

`Per-process stack size limit`
: Maximum stack segment size for any process in a running request.

`Per-process CPU time limit`
: Maximum total CPU time for any process in a running request.

`Per-process working set limit`
: Maximum amount of physical memory for any process in a running request.

Displaying queue status interactively

`qwatch` is designed for system managers who fine-tune ConvexNQS+ queue loads. Use it to interactively monitor a queue(s) without making repetitive calls to `qstat`.

While `qwatch` does not interfere with ConvexNQS+ daemons, it does use a small amount of CPU resource.

Do not use `qwatch` to check on the status of one request; use `qstat` instead.

The command format is

`qwatch [-i` *interval*`][-c` *count*`]`

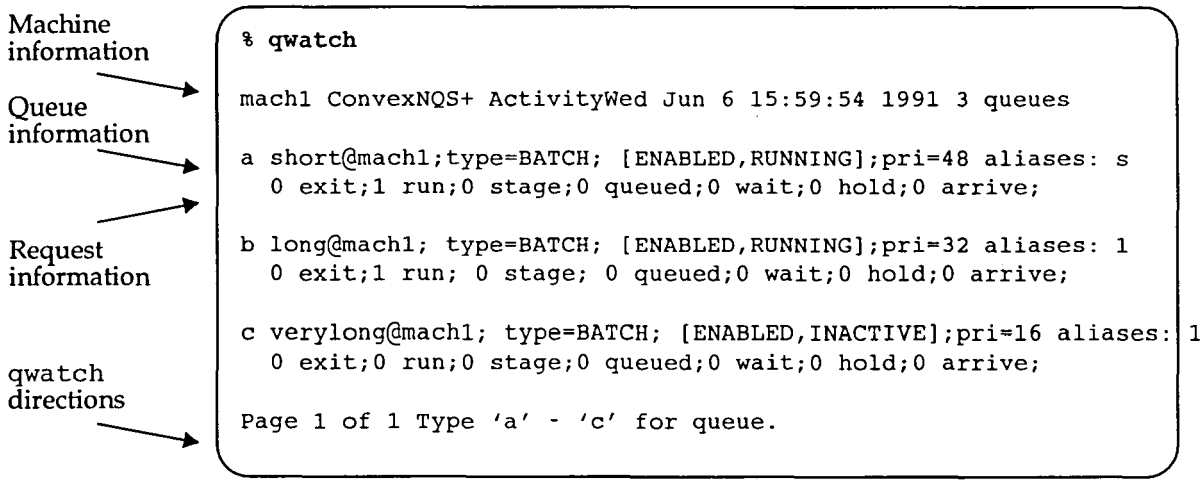
where

`-i` *interval* indicates the number of seconds between each screen update. The default is five seconds.

`-c` *count* indicates the number of screen updates. If this number is 0, ConvexNQS+ continues updating until you press CTRL-c. The default is 0.

Figure 5 illustrates queue status output.

Figure 5 `qwatch` queue status sample output



The first two lines of output describe information about the local machine:

```
mach1 ConvexNQS+ ActivityWed Jun 6 15:59:54 1991
3 queues
```

mach1... Name of the local machine, and the day, date, and time the status was taken.

3 queues Number of queues on the local machine.

The next two lines of output describe information about the first queue on the local machine:

```
a short@mach1;type=BATCH; [ENABLED,RUNNING];pri=48 aliases:
```

qwatch reference letter for this queue.

short@mach1
: Name of the queue and the machine on which it is configured.

type
: Type of queue. This can be

  BATCH — Runs ConvexNQS+ requests.

  PIPE — Routes ConvexNQS+ requests to queues that can run them.

enabled
: Indicates if this queue can accept requests. This can be

  ENABLED — ConvexNQS+ is running on the machine and the queue is accepting requests.

  DISABLED — ConvexNQS+ is running on the machine but the queue is not accepting requests.

  CLOSED — ConvexNQS+ is not running on the machine.

running
: Indicates if this queue can run requests. This can be

  INACTIVE — Requests in the queue can run; none are running.

  RUNNING — Requests in the queue can run; some are running.

  STOPPING — New requests sent to the queue cannot run, but requests that are currently running can complete.

  STOPPED — Requests in the queue cannot run; none are running.

  SHUTDOWN — ConvexNQS+ is not running on the machine.

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

aliases
: Alternate name(s) for this queue.

The next line of output tallies information about the requests in the first queue on the local machine:

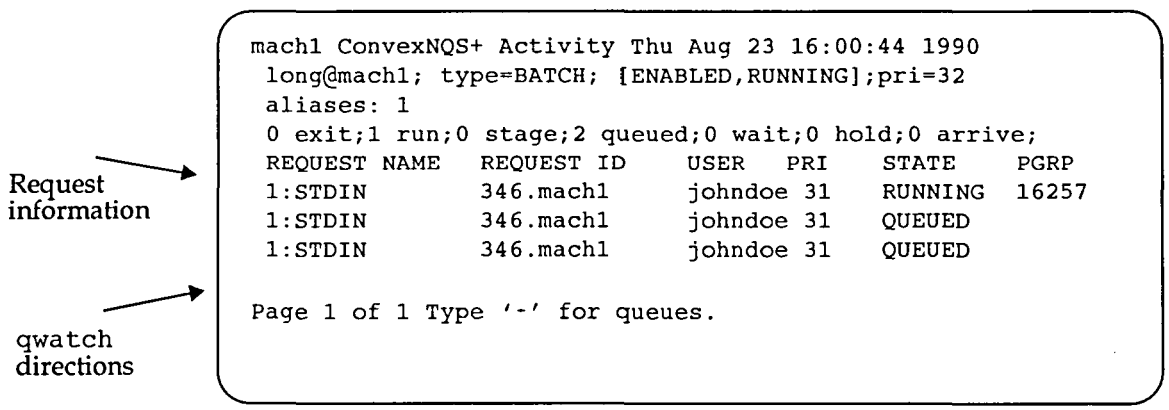
```
0 exit;1 run;0 stage;0 queued;0 wait;0 hold;0 arrive;
```

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

One `qwatch` page can display status information for up to five queues. To display status information for another five queues, type the appropriate page number. `qwatch` can display up to five pages, or 25 queues.

To display more information about each request in a queue, type the appropriate queue reference letter. Figure 6 illustrates request status output.

Figure 6 `qwatch` request status sample output



The information provided about each request is

```
REQUEST NAME    REQUEST ID   USER      PRI    STATE     PGRP
1:STDIN         346.mach1    johndoe    31    RUNNING   16257
```

REQUEST NAME
: Name assigned to this request.

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

USER
: User submitting this request.

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

STATE
: State of this request. This can be

  ARRIVING
  : Request is arriving at the queue.

  CHECKPOINTED
  : A failed attempt was made to restart this request after it was checkpointed.

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

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

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

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

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

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

  WAITING
  : Request is waiting a specified amount of time before running.

	This could be because it was submitted with a future run date and time, or because a pipe queue could not route the request and will try to route it later.
SUSPENDED	Request is suspended from running. It will remain suspended until manually resumed.
PGRP	Process group of the request, if available to the local ConvexNQS+ daemon. This information is displayed only for processes that are running.

One `qwatch` page can display status information for up to 16 requests. To display status information for another 16 requests, type the appropriate page number. `qwatch` can display up to nine pages, or 144 requests.

To redisplay queue status output, type a dash (-).

Displaying enforceable resource limits

You can display the enforceable batch queue resource limits for a specified machine(s) using the `qlimit` command.

The command format is

`qlimit [`*hostname* ...`]`

where *hostname* is the desired machine. If you omit *hostname*, ConvexNQS+ assumes the local machine.

Figure 7 illustrates `qlimit` output.

Figure 7 `qlimit` sample output

```
% qlimit
Per-process permanent file size limit (-lf)
Nice value (-ln)
```

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

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

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

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

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

Displaying process status

You can display the status of ConvexNQS+ processes on the local machine using the `qps` command.

To display status for processes associated with a specified queue(s), the command format is

`qps [`*queuename ...*`]`

where *queuename* is the name of the specified queue(s). If you omit *queuename*, ConvexNQS+ display status information for all ConvexNQS+ processes associated with all local batch queues, including the daemon processes.

To display status for processes associated with a specified request, the command format is

`qps -r` *request-id* where *request-id* is the name of the specified request. Use `qstat` to find the *request_id*. Refer to "Displaying queue status" in this chapter or the qstat(1) man page for details on using the `qstat` command.

To determine if a particular process is running, the command format is

`qps -{p|q}` *process-id*

where

-p *process-id* Determines if the process specified by *process-id* is running. If it is, `qps` shows the associated queue and request in the following manner:

```
Process ID 508 is being executed by Request ID
3782 in the short queue.
```

If the specified process is not running, `qps` displays the following:

```
Process ID 508 is not being executed by
ConvexNQS+.
```

`qps` regards ConvexNQS+ shepherd processes as running under ConvexNQS+; it does not regard top-level daemons as running under ConvexNQS+.

`-q` *process-id* Silent version of the `-p` option. ConvexNQS+ prints nothing to the screen, but sets the exit status of `qps` appropriately.

Figure 8 illustrates the output for the qps command.

Figure 8 qps sample output

```
% qps
QUEUE       REQ          PID      STAT    TIME        COMMAND
<daemon>                 21052 S          0:00        ConvexNQS+ logdaemon
<daemon>                 21053 S          0:00        ConvexNQS+ nqsdaemon
<daemon>                 21054 S          0:00        ConvexNQS+ netdaemon
long        508.mach     112535 S         0:00        ConvexNQS+ shepherd
long        508.mach     112536 S R       0:00        /bin/make -k ...
long        508.mach     112539 S T       0:00        /bin/make -k ...
long        508.mach     112535 S D       0:00        tsch
long        508.mach     112535 S R       0:00        tsch
```

Note

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

QUEUE — Queue that contains the request related to the process.

REQ — Identification number assigned to the request initiating the process, and machine where the request originated.

PID — Unique identification number assigned to the process.

STAT — Status of the process. Table 2 shows possible values for processes on Exemplar systems machines.

Table 2 qps STAT values

STAT	Value on an Exemplar
I	Intermediate
R	Running
S	Sleeping
T	Stopped
W	Waiting
X	Growing
Z	Terminated

TIME — Amount of CPU time used so far.

COMMAND — Command that started the process.

For more information, refer to the qps(1) man page.

Displaying request contents

You can display the contents of a request shell script using the `qjlist` command.

To display a shell script, you must either

- Own the request
- Have superuser privileges
- Be designated as a batch operator or manager

The command format is

`qjlist` *request_id* [*@hostname*]

where

request_id is the number assigned to the request when it is submitted to ConvexNQS+. Use the `qstat` command to find the *request_id*. Refer to "Displaying queue status" in this chapter or the `qstat`(1) man page for details on using the `qstat` command.

hostname is the machine that receives the request. If you omit *hostname*, ConvexNQS+ assumes the local machine.

Figure 9 illustrates the contents of the 52.mach2 shell script using the `qjlist` command.

Figure 9 `qjlist` sample output

```
% qjlist 52.mach2
sleep 20
echo "It is time to go home"
CTRL-d
```

For more information, refer to the `qjlist`(1) man page.

Submitting requests

3

This chapter explains how to submit a request to a ConvexNQS+ queue using the `qsub` command. It describes the following topics:

- Three ways to submit batch requests
- `qsub` command format
- Controlling how, when, and where a request is submitted and executed
- Redirecting output and error information
- Specifying resource limits
- Requesting notification of request status
- Embedding `qsub` command options in a shell script

Three ways to submit batch requests

A batch request consists of one or more commands submitted by a user or a user program to ConvexNQS+. You can submit a batch request using:

- Interactive commands
- A script file
- A compiled program

Each submittal method is described in the following sections.

Using interactive commands

You can submit a request interactively using the following procedure:

Step 1 Issue the `qsub` command.

Step 2 Issue the commands you want to execute.

Step 3 Press **CTRL-d**.

Figure 10 illustrates an interactive session.

Figure 10 Submitting a request interactively

```
% qsub
sleep 20
echo "It is time to go home"
CTRL-d
```

Using a script file

You can submit a request using the following procedure:

Step 1 Create a script file that contains all the commands you want to execute.

Make sure the script file

- Contains commands the interpreting shell can recognize
- Is completely self-contained so that when the script exits, no background child processes remain

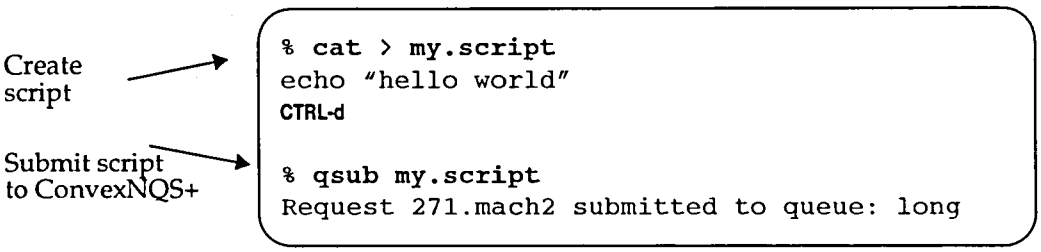
Note

If you send mail to a local user at the end of a script, have the request sleep for a few seconds just before exiting to let the delivery process complete. If the request exits before the mail delivery process completes, the shepherd process aborts the delivery and the mail disappears. This does not occur if you send mail to a remote machine or if you pass the -v (verbose) option to the mail program.

Step 2 Issue the `qsub` command with the script file name on the command line.

Figure 11 illustrates how to create a script file named *my.script* and submit it to ConvexNQS+.

Figure 11 Submitting a request using a script file



Using a compiled program

You can submit a request using a compiled program (binary code) by

- Including the name of the program in a script file

or

- Issuing the `qsub` command with the program name on the next command line

Figure 12 illustrates how to submit the compiled program *test* by entering the program name.

Figure 12 Submitting a request using a compiled program

```
% qsub
test
CTRL-d
```

Figure 13 illustrates how to submit the compiled program *test* by creating a script file and submitting that script file using the `qsub` command.

Figure 13 Submitting a compiled program using a script file

```
% cat > myjob
test
CTRL-d
% qsub myjob
```

qsub command options

Use the qsub command to submit a request to a queue. The format for the qsub command is

qsub *option* [*option* ...]

where *option* can be one or more values that let you:

- Control how, when, and where the request is run
- Redirect output files and error messages
- Specify resource limits
- Request notification of request status

ConvexNQS+ command format requires that you include a dash before each option. For example, to execute the qsub command with two options, enter:

qsub *-option1 -option2*

The following sections describe qsub command options.

Controlling requests

The qsub command includes options to control how, when, and where a request runs. Table 3 lists each option and its meaning.

Table 3 qsub options that control running requests

Option	Description
-a [*date*] [*time*]	Specifies a future run time
-b *bill_account*	Charges request to a specified billing account
-h	Places request on hold
-i	Imports current working directory
-l	Specifies interactive login shell
-ni	Prevents importation of current directory
-p *priority*	Sets intraqueue priority
-q *queue*	Submits request to a specified queue
-r *name*	Assigns name to request
-s *shell*	Specifies shell to interpret script files
-x	Exports value of environment variables
-z	Submits request silently

Specifying a future run time

Unless you specify otherwise, a request is eligible to run immediately after you issue the `qsub` command.

Use the `-a` option to specify a future run time.

The command format is

`qsub -a` [*date*] [*time*]

where

date is the date the request should run. You can specify the date several ways, including:

- Month, day and year in the following format: `"Oct 31, 1992"`
- Day, month and year in the following format: `31-Oct-1992`
- Day of the week
  * today
  * tomorrow

Notice the quotation marks around a date that contains embedded spaces.

If you omit the year, ConvexNQS+ assumes the current year. If you omit the entire date, ConvexNQS+ assumes the current month, day, and year.

You can abbreviate a month or day of the week to the first three (or more) letters. A period following the abbreviation is optional. Uppercase and lowercase letters are equivalent, so `Jun` is treated the same as `jun` or `JUN`.

time is the time the request should run. You can specify the time several ways, including:

- 24-hour clock (default) and time zone in the following format: `13:00-CST`
- 12-hour clock in the following formats: `1:00pm` or `1pm` or `"1:00 pm"`
  * noon
  * midnight

Notice the quotation marks around a time that contains embedded spaces.

If you omit the time zone, ConvexNQS+ assumes the local time zone, with daylight savings time included when appropriate.

12am refers to 0:00:00, 12n refers to noon, and 12pm refers to 24:00:00. The order of *date* and *time* is irrelevant. The following formats are equivalent:

```
qsub -a [date] [time]
  qsub -a [time] [date]
```

If you use both *date* and *time*, enclose the entire string in one set of quotation marks. For example:

```
"January 1, 1989, 12:31"
  "01-Jan-1988 13:00"
  "today 5pm"
```

For example, enter the following command to execute the script file *test* on July 4, 2026, at 12:31 Eastern Daylight Time:

```
qsub -a "July 4, 2026 12:31-EDT" test
```

Charging a request to a specified billing account

Unless you specify otherwise, ConvexNQS+ charges a request to a billing ID assigned by your system manager during initial system configuration.

Use the -b option to specify that a request be charged to a specified billing account.

The command format is

`qsub -b` *bill_account*

where *bill_account* is the desired account.

For example, enter the following command to charge the script file *my_job* to the *activity1* billing account:

```
qsub -b activity1 my_job
```

Placing a request on hold at submittal

Unless you specify otherwise, a request is eligible to run immediately after you issue the `qsub` command.

Use the -h option to place a request on hold at the time you submit it.

For example, enter the following command to place the script file *my_job* on hold when it is submitted:

```
qsub -h my_job
```

Controlling importation of the current directory

A request often requires access to all files in all subdirectories of your current working directory in order to run. Granting this access is called *importing* the current working directory.

Your system manager configures importing capability by queue during initial system configuration. Use the qstat command to determine if your destination batch queue permits importing (`Import directory=Yes` or `Available`) or does not permit importing (`Import directory = No`). See Chapter 2, "Displaying information," for details on using the qstat command.

If your destination batch is configured with `Import directory=Available`, use the -i option to import the current working directory.

For example, enter the following command to import the current working directory for the script file *my_job*:

```
qsub -i my_job
```

If the destination batch queue is on the same machine as your current working directory, ConvexNQS+ changes directories before it starts. If the destination batch queue is on another machine, ConvexNQS+ imports the directories and subdirectories with NFS by making temporary mount points in the /tmp directory on the originating machine. Be aware of any local automatic cleanup facilities of /tmp that might affect these NFS mount points.

ConvexNQS+ cannot import a current working directory that is already within an NFS file system. For example, if you are on a machine called *sleepy* in a file system imported from *happy*, and you submit a job with the -i option to a queue on *grumpy*, ConvexNQS+ cannot import the file system from *happy* to *grumpy*.

If your destination batch queue is configured with `Importing=Yes`, use the -ni option to prevent importing the current working directory.

For example, enter the following command to prevent importing the current working directory for the script file *my_job*:

```
qsub -ni my_job
```

If your destination batch queue is configured with `Import directory=No`, ConvexNQS+ expects to find any files required to run the request in your home directory.

Specifying an interactive login shell

Your system manager defines a shell strategy for each batch queue during initial system configuration. A shell strategy determines the command interpreter used to interpret request script commands if a request is submitted to a queue without an identified shell interpreter. The shell strategy can be

- **Fixed**—The system manager specifies the shell during initial system configuration.
- **Login**—ConvexNQS+ uses the user's default login shell (as defined in the /etc/password file) to interpret the script file commands
- **Free** —ConvexNQS+ uses the user's login shell (as defined in the /etc/password) file to initially interpret commands. ConvexNQS+ then supplies the name of the script file to the login shell as standard input. The user's login shell reads the first line of the script file. If the first line specifies a shell, ConvexNQS+ uses that shell to interpret the script file commands. Otherwise, ConvexNQS+ uses sh.

Use the `qlimit` command to determine the strategy for your destination batch queue. See Chapter 2, "Displaying information," for details on using the `qlimit` command.

Unless you specify otherwise, ConvexNQS+ runs jobs using a noninteractive shell. Use the -l option to specify an interactive login shell. The interactive login shell defaults to the login for the batch queue unless you specify an alternate shell using the -s option.

The command format is

`qsub -s` *shell-name*

where *shell-name* is the absolute path name of the desired shell on the executing machine.

If you use a C shell, ConvexNQS+ gets environment variables from /etc/login and the .login file in your home directory. If you use a Bourne or Korn shell, ConvexNQS+ gets environment variables from /etc/profile and the .profile file in your home directory.

To prevent stty, tset, and msgs from running during batch jobs, add the string `ENVIRONMENT=BATCH` to your .login, .profile, or .cshrc file so that shell scripts can test for batch request execution. Then place an `if` statement in the .login, .profile, or .cshrc file.

For example, if your login shell is a C shell, the following if statement in your .login file prevents stty, tset, and msgs from

running during batch jobs. C shell always reads your .cshrc file regardless of whether or not it is running as a login shell.

```
if (! $?ENVIRONMENT) then
stty erase ^H kill ^U
tset -Q
msgs -q
endif
```

If your login shell is a Bourne or Korn shell, the following if statement in your .profile file prevents stty, tset, and msgs from running during batch jobs.

```
if test "$ENVIRONMENT" != "BATCH"
then
stty erase ^H kill ^U
tset -Q
msgs -q
fi
```

As an alternative, you might use

```
if ($?ENVIRONMENT) exit
```

Setting request priority

When ConvexNQS+ adds a request to a queue, it compares the intraqueue priority of the request with the intraqueue priority of all existing requests. It then places the request in the queue ahead of existing requests with a lower priority and behind existing requests with a higher priority. 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.

Unless you specify otherwise, ConvexNQS+ assigns the default intraqueue priority to the request. Use the `-p` option to assign an intraqueue priority to a request. This priority determines the relative ordering of requests within a queue; it does not determine execution priority.

The command format is

`qsub -p` *priority*

where *priority* is the desired intraqueue priority.

The specified priority can be any number from 0 and 63; 0 is the lowest priority and 63 the highest. You cannot assign a priority to a request that is higher than the maximum request priority assigned to the queue. For example, enter the following command to set a priority of 48 on the script file *test*:

```
qsub -p 48 test
```

Submitting a request to a specified queue

Unless you specify otherwise, ConvexNQS+ determines which queue should receive a request by checking the value of the QSUB_QUEUE environment variable in the .login file of the user submitting the job. If this environment variable is not defined, ConvexNQS+ submits the request to the default batch queue defined by your system manager. If your system manager has not defined a default batch queue, ConvexNQS+ displays an error message and rejects the request.

Use the -q option to submit a request to a specified queue.

The command format is

`qsub -q` *queuename* [@*hostname*]

where

queuename is the name of the queue.

hostname is the name of the machine on which the queue resides. If you omit *hostname*, ConvexNQS+ assumes the local machine.

For example, enter the following command to submit the script file *test* to the short queue:

```
qsub -q short test
```

Naming a request

Unless you specify otherwise, ConvexNQS+ assigns a request the same name as the script file (less the path name) supplied on the command line. If there is no script file, ConvexNQS+ assigns the default name STDIN.

Use the -r option to assign a name to a request.

The command format is

`qsub -r` *request-name*

where *request-name* is the desired name. The name can be any length, but its display is truncated to fifteen characters. In the actual output file, *request-name* is truncated to seven letters, plus regular extensions. If *request-name* begins with a digit, ConvexNQS+ prefixes the name with the letter R.

For example, enter the following command sequence to assign the name *myjob* to the interactive batch session:

```
qsub -r myjob
echo "hello world"
CTRL-d
```

Exporting environment variables

Unless you specify otherwise, a request does not inherit the submitting user's environment.

Use the-x option to export all user environment variables with the request except the following: HOME, SHELL, PATH, USER, LOGNAME, MAIL, TZ. These exception environment variables are reserved, but are prefixed with QSUB_ (as in QSUB_HOME) and passed to the job.

Submitting a request silently

Unless you specify otherwise, ConvexNQS+ displays the *request_id* assigned to a request at the submitting user's terminal upon successful submittal.

Use the -z option to prevent display of this message.

This option has no effect on error messages; they are always displayed.

Redirecting output files and error messages

Unless you specify otherwise, ConvexNQS+ sends request:

- Output to a file named STDIN.o*seq#* on the machine that originated the request, where *seq#* is the sequence number assigned to the request in the queue
- Errors to a file named STDIN.e*seq#* on the machine that originated the request, where *seq#* is the sequence number assigned to the request in the queue

When you submit a request from a machine running an automounting daemon, you must explicitly specify output and error files with full path names.

If you are near your quota limit on your home file system when you submit a job, you will lose the information normally stored in STDIN.o*seq#*. If the request runs on the same machine that originated the request, ConvexNQS+ notifies you that the output could not be saved. If the request runs on a remote machine, ConvexNQS+ does not send this notification.

The qsub command includes options to control output and error destinations. Table 4 lists each option and its meaning.

Table 4 qsub options that redirect output and error messages

Option	Description
-e	Redirects error messages
-eo	Redirects error messages to standard output file
-ke	Creates standard error output file on executing machine
-ko	Creates standard output file on executing machine
-o	Redirects standard output
-y	Appends accounting information to standard output file

The following sections explain each option in detail.

Redirecting error messages

Use the -e option to redirect error messages to a specified file instead of the default file.

The command format is

qsub -e [*hostname*:][/][*pathname*/]*filename*

where

hostname is the machine on which the specified error file resides.

If you omit *hostname*, ConvexNQS+ sends error messages to the machine that originated the request.

If you omit *hostname* and also use the -ke option, ConvexNQS+ keeps the error messages on the machine that executes the request.

If you omit *hostname* and the / preceding the path name, ConvexNQS+ sends the error messages to the specified file in the current working directory, provided that -ke is not used. Otherwise, ConvexNQS+ interprets any portion of the path name included on the command line in relation to the user's home directory on the standard error destination machine.

pathname is the path name for the specified error file.

filename is the name of the specified error file.

For example, enter the following command to send error messages for the *test* script file to the file *qsub07.redirected_err* on the *mach2* machine:

```
qsub -e mach2:qsub07.redirected_err test
```

You cannot use the -e option with the -eo option.

Redirecting error messages to standard output file

Use the -eo option to redirect error messages to the standard output file (STDIN.o*seq#*) instead of the standard error file (STDIN.*eseq#*).

For example, enter the following command to redirect error messages for the script file *test* to the standard output file:

```
qsub -eo test
```

You cannot use the -eo option with the -e or the -ke options.

Creating error output file on the executing machine

Use the -ke option to create the standard error output file on the machine executing the request instead of on the machine originating the request.

For example, if you submit the script file *test* from the *mach2* machine but ConvexNQS+ executes *test* on the *liberty* machine, enter the following command to create the standard error output file for the script file on the *liberty* machine:

```
qsub -ke test
```

Use the -ke option with the -e option to specify the path or file name. If you do not specify a path or file name, ConvexNQS+ places the file in your home directory. Do not specify a hostname with the -e option, or ConvexNQS+ ignores -ke option.

You cannot use the -ke option with the -eo option.

Creating standard output file on the executing machine

Use the -ko option to create the standard output file on the machine executing the request instead of on the machine originating the request.

For example, if you submit the script file *test* from the *mach2* machine but ConvexNQS+ executes *test* on the *liberty* machine, enter the following command to create the standard output file for the script file on the *liberty* machine:

```
qsub -ko test
```

Use the -ko option with the -e option to specify the path or file name. If you do not specify a path or file name, ConvexNQS+ places the file in your home directory. Do not specify a hostname with the -e option, or ConvexNQS+ ignores -ko option.

The -ko option is also meaningless if the executing machine imports the current directory.

Redirecting standard output

Use the -o option to redirect standard output to a specified file instead of the default file.

The command format is

`qsub -o` [*hostname:*] [[/] [*pathname/*] *filename*

where

hostname is the machine on which the specified output file resides.

If you omit *hostname*, ConvexNQS+ sends the output to the machine that originated the request.

If you omit *hostname* and also use the -ko option, ConvexNQS+ keeps the output on the machine that executes the request.

If you omit *hostname* and the / preceding the path name, ConvexNQS+ sends the output to the specified file in the current working directory, provided that -ko is not used. Otherwise, ConvexNQS+ interprets any portion of the path name included on the command line in relation to the user's home directory on the standard output destination machine.

pathname is the path name for the specified output file.

filename is the name of the specified output file.

For example, enter the following command to send standard output for the *test* script file to the file *myjob.redirect_out* on the local machine:

```
qsub -o myjob.redirect_out test
```

Appending accounting information to standard output file

Use the -y option to append accounting information to your request's standard output file. Accounting must be enabled for the queue in which the request runs to use this option.

For more information on accounting, see the *ConvexNQS+ System Manager's Guide for Exemplar Systems*.

Specifying resource limits

The `qsub` command includes options that let you limit the:

- Resources consumed by each process within a request (per-process limits)
- Resources consumed by all processes together within a request (per-request limits)

The command format is

`qsub` *option size-limit* [, *warning-limit*]

where

option indicates a resource limit. See Table 5 for per-process and per-request limit options.

size-limit is the desired maximum limit.

warning-limit is the limit at which ConvexNQS+ delivers a warning signal to the executing process. If you omit *warning-limit*, ConvexNQS+ does not deliver a warning signal.

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

If you submit a request without limit specifications to a queue, ConvexNQS+ uses the enforceable limits set for the queue as defaults.

Not all operating systems support all limits. If you submit a request specifying a limit to a machine whose operating system cannot enforce the limit, ConvexNQS+ ignores the limit.

Table 5 lists resource limit options on the SPP-UX operating system.

Table 5 Preprocess and per-request limits

Option	Resource	Limit
`-lc`	Core file size	A number specified in bytes
`-ld`	Data segment size	A number specified in bytes
`-lf`	Permanent file size	A number specified in bytes
`-lm`	Memory	A number specified in bytes
`-ln`	Nice value	A number between -64 and +64

Table 5 Preprocess and per-request limits **(continued)**

Option	Resource	Limit
`-ls`	Stack segment size	A number specified in bytes
`-lt`	CPU time	hours:minutes:seconds:milliseconds
`-lv`	Temporary file	A number specified in bytes
`-lw`	Working set size	A number specified in bytes

Requesting notification of request status

Unless you specify otherwise, ConvexNQS+ sends mail to the user submitting the request only if it cannot run the submitted shell script.

The `qsub` command includes options that let you request that mail be sent when a request starts running or completes running.

Table 6 lists the options that affect notification.

Table 6 `qsub` options that affect notification

Option	Description
`-mb`	Requests notification of when request starts running
`-me`	Requests notification of when request completes running
`-mu`	Requests notification be sent to a specified user
`-t`	Signals a process when request completes running

The following sections explain these options.

Requesting notification of when request starts running

Use the -mb option to request that ConvexNQS+ send you mail on the originating machine when the request starts running.

For example, enter the following command to send mail to the user submitting the request named *myjob* when the job starts running:

```
qsub -mb myjob
```

Figure 14 illustrates the mail received as a result of using the `-mb` option when submitting a request.

Figure 14 Mail received when request starts running

```
Request name: myjob
Request owner: johndoe
Mail sent at: Fri Aug 24 10:43:07 CDT 1994
```

Requesting notification of when request completes running

Use the -me option to request that ConvexNQS+ send you mail on the originating machine when the request completes running.

For example, enter the following command to send mail to the user submitting the request named *myjob* when the job completes running:

```
qsub -me myjob
```

Mail sent as a result of using the -me option contains a summary of CPU time used by the request. Figure 15 illustrates an excerpt of a mail message received as a result of using the -me option.

Figure 15 Mail received when request completes running

```
Request name: STDIN
Request owner: johndoe
Mail sent at: Fri Aug 24 10:47:36 CDT 1990
Request exited normally.
_Exit() value was: 0.
 %
```

Requesting notification be sent to a specified user

Use the -mu option to request that ConvexNQS+ send mail to a particular user or machine instead of to yourself. The command format is

qsub -mu *username* [@*hostname*]

where

username is the user to receive the mail.

hostname is the machine where *username* is located.

For example, enter the following command to send mail to user *smith* on the *mach2* machine when the request starts running and completes running:

```
qsub -me -mb -mu smith @mach2
```

Figure 16 illustrates the resulting mail received by the user *smith*.

Figure 16 Mail received by specified user

```
>From daemon Tue Jan 24 15:39:55 1990
Received: by mach2 (5.51/4.7)
 id AA18211; Tue, 24 Jan90 15:39:53 CST
Date: Tue, 24 Jan90 15:39:53 CST
From: root (Superuser)
Subject: ConvexNQS+ request: 2172.mach2 beginning.
Apparently-To: smith
Status: R

Request name: STDIN
Request owner: johndoe
Mail sent at: Tue Jan 24 15:39:52 CST 1990

>From daemon Tue Jan 24 15:40:03 1990
Received: by mach2 (5.51/4.7)
id AA18224; Tue, 24 Jan90 15:40:01 CST
Date: Tue, 24 Jan90 15:40:01 CST
From: root (Superuser)
Subject: ConvexNQS+ request: 2172.mach2 ended.
Apparently-To: smith
Status: R

Request name: STDIN
Request owner: johndoe
Mail sent at: Tue Jan 24 15:40:01 CST 1990
Request exited normally.
_Exit() value was: 1.
```

Signalling processes when request completes running

Use the -t option to request that ConvexNQS+ signal a particular process when the request completes running. The command format is

`qsub -t` *process-id*

where *process-id* is the process to be signalled.

ConvexNQS+ sends one of the following signals, depending on how the request completes:

- **SIGTERM** —Request completed normally.
- **SIGUSR1**—Request aborted while running.
- **SIGUSR2**—Request was deleted before it ran.

Embedded options

You can also include `qsub` options in the first comment block of a script file.

If the next nonblank characters are @$, ConvexNQS+ treats the line as an embedded option. Indicate the end of the option with either the end of the line or unquoted # or $! characters. Indicate the end of the comment block with any character other than # or $! as the first character on a line.

Options embedded in a script file set the default characteristics for the request. Command line options override embedded values in the script file.

For example, when you enter the command:

```
qsub -a 10:00am EDT testjob
```

and embed the following option in the script file *testjob*:

```
-a "11:30pm EDT"
```

the request runs at 10:00 a.m. EDT because the value on the command line takes precedence over the embedded option.

Figure 17 illustrates use of embedded options in a script file.

Figure 17 Embedded options in a script file

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

Controlling requests

4

There are several commands available for controlling your requests, including commands for

- Deleting a request
- Delaying a queued request
- Moving a request
- Change a request's priority

This chapter describes how to perform each of these tasks.

Deleting specific requests

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

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

Each command is described in the following sections.

Using the delete request command

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

```
qmgr
```

The command format is

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

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

You can specify more than one request.

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

ConvexNQS+ removes deleted requests and discards them.

Using the qdel command

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

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

where

option can be one of the following:

`-u` *username*	Lets you delete requests other than your own, where *username* is the name of the user who owns the request. By default, only the user who submitted the request can delete it from a queue. This option lets the superuser, or ConvexNQS+ manager or operator delete someone else's request from a queue.
`-k`	Sends a SIGKILL (-9) signal to a running request. When the request exits, ConvexNQS+ deletes it.
sig	Sends a specified signal to a running request where *sig* can either be the signal number or signal name in the /usr/include/signal.h file.

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

You can specify more than one request.

You can specify a running or non-running request.

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

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

For example, if you have operator privileges, enter the following command to delete the request identified as *291* on the local machine submitted by user *smith*.

```
qdel -u smith 291
```

Delaying queued requests

Use the `hold request` and `release request` commands to delay the running of a queued request. If a ConvexNQS+ manager or operator places the request on hold, only a ConvexNQS+ manager or operator can release the hold.

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

```
qmgr
```

Each command is described in the following sections.

Placing a queued request on hold

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

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

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

You can specify more than one request. You must specify a request in the queued state.

Releasing the hold on a queued request

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

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

where *request_id* is the number assigned to the request when it is submitted to ConvexNQS+. Use the `qstat` command to find the *request_id*. You can specify more than one request. You must specify a request in the holding state.

Moving specific non-running requests to another queue

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

The `move my_request` command is a `qmgr` utility command, which means you must start `qmgr` before you can use this command. To start `qmgr`, enter **`qmgr`**.

The command format is

`move my_request` *request_id* [*request_id ...*] *queuename*

where

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

You can specify more than one request.

You must specify a nonrunning request.

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

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

Changing the priority of non-running requests

Use the `modify request` command to change the priority of a non-running request.

The `modify request` is a `qmgr` utility command, which means you must start `qmgr` before you can use this command. To start `qmgr`, enter **`qmgr`**.

The command format is

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

where

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

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

You can specify more than one request.

You must specify a non-running request.

Transaction completion messages

A

This appendix contains information on the following topics, and

- Lists common codes that may be returned by any part of the ConvexNQS+ system
- Discusses the meaning of the codes
- Describes actions you should take in response to the codes

These codes include error and success codes. The code format is

TCM*machine_ code, message_text*

where

machine	indicates the machine on which the request was running when the error occurred. This can be L for local or P for peer (remote).
code	is the mnemonic code for the error message.
message_text	is the text explaining the reason for the error.

The codes in this appendix are listed alphabetically by the first letter in *code*. Each is followed by explanatory text detailing

- The cause of the error
- The result
- In some cases, the action you should take

Some explanatory text may say *Quota information bits are present*. This indicates that the error involves a violation of a quota limit, and ConvexNQS+ displays another error message with additional information about the limit violation. These additional messages are listed at the end of this appendix.

TCML_ACCESSDEN **Access denied at local host.**

The transaction cannot be performed because:

- A queue denied access to a user invoking the `qsub` command.
- A request was submitted directly to a queue that can only receive requests from other pipe queues.
- A user submitted a request to a queue that does not have the user or the user's group in its access list and does not have unrestricted access.

The transaction failed; do not retry.

TCMP_ACCESSDEN **Access denied at transaction peer.**

The transaction cannot be performed because:

- A queue denied access to the owner of a request.
- A request was submitted to *queue@host* where *nqsdaemon@host* was started with the `-r` flag.
- A request was submitted directly to a *queue@host* (`qsub -q` *queue@host* remotely) that can only receive requests from other pipe queues.
- A user submitted a request to a queue that does not have the user or the user's group in its access list and does not have unrestricted access.

The transaction failed; do not retry.

TCML_ALREADACC **Already has access at local host.**

The transaction involves the addition of a user or group ID to a queue access set, and the user or group ID is already present in the queue access set.

The transaction failed.

TCML_ALREADEXI **Already exists at local host.**

The transaction involves the addition of a set element, and the element is already present in the set.

The transaction failed.

TCML_BADCDTFIL

The transaction involves an operation-upon-request control or data file which are corrupt at the local machine.

The transaction failed; do not retry.

TCMP_BADCDTFIL **Corrupt request control/data file(s) detected at transaction peer. Seek staff support.**

The transaction involves an operation on request control file(s) or data file(s), and the request control or data file(s) is corrupt at the peer machine.

The transaction failed; do not retry.

TCMP_CLIMIDUNKN **Client machine id unknown at transaction peer. Seek staff support.**

The transaction cannot be performed because:

- A machine ID cannot be determined on the remote machine for the client's network address.
- A user submitted a request from *hostA* to *hostB* when qmapmgr on *hostB* does not have an machine ID entry for *hostA*.

TCML_COMPLETE **Transaction complete at local host.**

The transaction completed successfully.

This is a generic transaction completion code and can be used as the final result code from the local machine for any transaction when a more specific code does not exist.

The transaction succeeded.

TCMP_COMPLETE **Transaction complete at transaction peer.**

The transaction completed successfully.

This is a generic transaction completion code and can be used as the final result code from the local machine for any transaction when a more specific code does not exist.

The transaction succeeded.

TCMP_CONNBROKEN **Network connection lost. Retry later.**

The transaction cannot complete because the network connection established for the transaction was severed and the machine went down.

The transaction failed; retry later.

TCMP_CONNTIMOUT **Network connection timeout. Retry later.**

The transaction cannot be performed because of a timeout waiting for the transaction peer to respond, causing the machine to go down.

The transaction failed; retry later.

TCMP_CONTINUE **Subtransaction complete at transaction peer.**

The transaction began successfully, or the previous subtransaction of the transaction completed successfully.

The server (in the client/server) is ready to perform the next subtransaction, as directed by the client.

The transaction continues.

TCML_CPUALRESVD

The transaction involves the allocation of a local CPU to a local batch queue, the CPU is already allocated to another local batch queue.

The transaction failed.

TCML_DEVACTIVE

The transaction specified the deletion of an active device.

The transaction failed.

TCML_DEVENABLE

The transaction specified the deletion of an enabled device.

The transaction failed.

TCML_EACCESS **File access denied at local host.**

The transaction involves a file operation that cannot be performed because the protections set on the local machine deny access to the associated user.

The transaction failed; do not retry.

TCMP_EACCESS **File access denied at transaction peer.**

The transaction involves a file operation that cannot be performed because the protections set on the peer machine deny access to the associated user.

The transaction failed; do not retry.

TCML_EFBIG **File size limit exceeded at local host.**

The transaction involves a file operation that increases the size of the file beyond the maximum supported at the local machine.

The transaction failed; do not retry.

TCMP_EFBIG **File size limit exceeded at transaction peer.**

The transaction involves a file operation that increases the size of the file beyond the maximum supported at the peer machine.

The transaction failed; do not retry.

TCML_EISDIR **Operation on directory is prohibited at local host.**

The transaction involves an illegal operation on a directory based on the protections set on the local machine.

The transaction failed; do not retry.

TCMP_EISDIR **Operation on directory is prohibited at transaction peer.**

The transaction involves an illegal operation on a directory based on the protections set on the peer machine.

The transaction failed; do not retry.

TCML_ELOOP **Symbolic link translation limit exceeded at local host.**

The transaction involves the translation of symbolic links that exceed the limit on the local machine.

The transaction failed; do not retry.

TCMP_ELOOP **Symbolic link translation limit exceeded at transaction peer.**

The transaction involves the translation of symbolic links that exceed the limit on the peer machine.

The transaction failed; do not retry.

TCML_ENFILE **Insufficient file descriptors at local host. Retry later.**

The transaction cannot be tried or completed because of a file descriptor shortage on the local machine.

The transaction failed; retry later.

TCMP_ENFILE **Insufficient file descriptors at transaction peer. Retry later.**

The transaction cannot be tried or completed because of a file descriptor shortage on the peer machine.

The transaction failed; retry later.

TCML_ENOBUFS **Insufficient buffer space at local host. Retry later.**

The transaction cannot be tried because there are not enough buffers available on the local machine to establish the network connection required for the transaction.

The transaction failed; retry later.

TCMP_ENOBUFS **Insufficient buffer space at transaction peer. Retry later.**

The transaction cannot be tried because there are not enough buffers available on the peer machine to establish the network connection required for the transaction.

The transaction failed; retry later.

TCML_ENOENT **Component in file path does not exist at local host.**

The transaction involves a file operation in which some element of the file path name does not exist at the local machine.

The transaction failed; do not retry.

TCMP_ENOENT **Component in file path does not exist at transaction peer.**

The transaction involves a file operation in which some element of the file path name does not exist at the peer machine.

The transaction failed; do not retry.

TCML_ENOMEM **Insufficient memory at local host. Retry later.**

The transaction cannot be performed because there is insufficient memory or swap space at the local machine.

The transaction failed; retry later.

TCMP_ENOMEM **Insufficient memory at transaction peer. Retry later.**

The transaction cannot be performed because there is insufficient memory or swap space at the peer machine.

The transaction failed; retry later.

TCML_ENOSPC **File system resource shortage at local host. Retry later.**

The transaction cannot be completed or performed because of a file system resource shortage on the local machine.

The transaction failed; retry later.

TCMP_ENOSPC **File system resource shortage at transaction peer. Retry later.**

The transaction cannot be completed or performed because of a file system resource shortage on the peer machine.

The transaction failed; retry later.

TCML_ENOTDIR **Invalid file path at local host.**

The transaction involves a file operation in which an element in the file path name is not a directory at the local machine.

Or the qmgr set checkpoint_directory command specified a path that is not a directory.

The transaction failed; do not retry.

TCMP_ENOTDIR **Invalid file path at transaction peer.**

The transaction involves a file operation in which an element in the file path name is not a directory at the peer machine.

The transaction failed; do not retry.

TCML_ENXIO

The transaction involves some operation on a special file (i.e. SPP-UX subdevice,) and the subdevice does not exist, or some other error condition exists with the subdevice at the local machine. The error can include, but is not limited to, the following:

- Tape drive not on line
- Disk pack not loaded in drive
- Beyond the limits of the device

The transaction failed, retry later.

TCMP_ENXIO

The transaction involves some operation on a special file (i.e. SPP-UX subdevice), and the subdevice does not exist, or some other error condition exists with the subdevice at the peer machine.The error can include, but is not limited to, the following:

- Tape drive not on line
- Disk pack not loaded in drive
- Beyond the limits of the device

The transaction failed, retry later.

TCML_EPERM **Permission denied at local host.**

The transaction involves an operation that cannot be performed because the protections set on the local machine deny permission to the associated user.

The transaction failed; do not retry.

TCMP_EPERM **Permission denied at transaction peer.**

The transaction involves an operation that cannot be performed because the protections set on the peer machine deny permission to the associated user.

The transaction failed; do not retry.

TCML_EPIPE **Fifo error at local host. Retry later.**

A transaction tried to write on a pipe or socket that has no process attached to read the data. This usually happens when the read process exits prematurely or is killed.

The transaction failed; retry later.

TCMP_EPIPE **Fifo error at transaction peer. Retry later.**

A transaction tried to write on a pipe or socket that has no process attached to read the data. This usually happens when the read process exits prematurely or is killed.

The transaction failed; retry later.

TCML_EROFS **Attempt to modify a read-only file system at local host.**

The transaction involves a file operation that alters a portion of a read-only file system at the local machine.

The transaction failed; do not retry.

TCMP_EROFS **Attempt to modify a read-only file system at transaction peer.**

The transaction involves a file operation that alters a portion of a read-only file system at the peer machine.

The transaction failed; do not retry.

TCML_ERRORRETRY **Recoverable transaction failure at local host. Retry later.**

The transaction cannot complete because of an error condition at the local machine that may not occur in the future.

This is a generic transaction completion code and can be used as the final result code from the local machine for any transaction when a more specific code does not exist. Explanatory text is displayed with this message.

The transaction failed; retry later.

TCMP_ERRORRETRY **Recoverable transaction failure at transaction peer. Retry later.**

The transaction cannot complete because of an error condition at the peer machine that may not occur in the future.

This is a generic transaction completion code and can be used as the final result code from the local machine for any transaction when a more specific code does not exist. Explanatory text is displayed with this message.

The transaction failed; retry later.

TCML_ETIMEDOUT **Connect(2) timeout at local host. Retry later.**

A request for connection on the local machine failed because the connected party did not properly respond after a period of time.

The transaction failed; retry later.

TCML_ETXTBSY **Text file operation denied at local host. Retry later.**

The transaction cannot complete because the involved file operation deals with a busy text file. The operation happened in circumstances under which such action is prohibited at the local machine.

The transaction failed; retry later.

TCMP_ETXTBSY **Text file operation denied at transaction peer. Retry later.**

The transaction cannot complete because the involved file operation deals with a busy text file in circumstances under which such action is prohibited at the peer machine.

The transaction failed; retry later.

TCML_FATALABORT **Non-recoverable transaction failure at local host.**

The transaction cannot complete because of an error condition that will not go away in the foreseeable future at the local machine.

This is a generic transaction completion code and can be used as the final result code from the local machine for any transaction when a more specific code does not exist. Explanatory text is displayed with this message.

The transaction failed; do not retry.

TCMP_FATALABORT **Non-recoverable transaction failure at transaction peer.**

The transaction cannot complete because of an error condition that will not go away in the foreseeable future at the peer machine.

This is a generic transaction completion code and can be used as the final result code from the local machine for any transaction when a more specific code does not exist. Explanatory text is displayed with this message.

The transaction failed; do not retry.

TCML_FEATNOSUP **Feature is not supported at local host.**

The transaction involves a a ConvexNQS+ feature that is not supported on the local machine.

The transaction failed; do not retry.

TCML_FIXBYNQS **Limit set to enforceable value at local host other than originally requested.**

The transaction involves a quota limit value for a batch queue that cannot be enforced by the local machine. The quota limit has instead been adjusted to fall within the limit enforceable at the local machine.

The transaction succeeds.

TCML_GRANFATHER **Transaction successful at local host but one or more requests were given a grandfather clause.**

The transaction modified a batch queue quota limit to be less than the limit set for one or more previously-queued batch request limits.

The transaction succeeds.

TCML_INSHUTDOWN **ConvexNQS+ is shutting down at local host.**

The transaction tested the internal state of ConvexNQS+ before continuing, and the local ConvexNQS+ daemon on the local machine is in the process of shutting down.

TCML_INSQUESPA **Insufficient space to queue request at local host. Retry later.**

The transaction cannot be performed because:

- There is insufficient queue space on the local machine.
- The local nqsdaemon either cannot allocate memory for a new request or cannot allocate a new transaction id.

The transaction failed; retry later.

TCMP_INSQUESPA **Insufficient space to queue request at transaction peer.Retry later.**

The transaction cannot complete because of insufficient queue space on the peer machine.

The transaction failed; retry later.

TCML_INSUFFMEM **Insufficient memory at local host.**

The transaction cannot complete because there is insufficient memory on the local machine.

The transaction failed.

TCML_INSUFFPRV **Insufficient privilege at local host.**

The transaction cannot be performed because:

- ConvexNQS+ does not grant the associated user the appropriate privileges.
- A qmgr command that requires operator or manager privileges was tried by a user without required privileges.
- A user who was not a manager tried to raise the priority of one of his/her requests.

The transaction failed.

TCML_INTERNERR **Internal ConvexNQS+ error at local host. Seek staff support.**

The transaction cannot complete because of an internal error at the local machine.

The transaction failed; do not retry.

TCMP_INTERNERR **Internal ConvexNQS+ error at transaction peer. Seek staff support.**

The transaction cannot complete because of an internal error at the peer machine.

The transaction failed; do not retry.

TCML_LOGFILERR **Cannot open or create new logfile at local host.**

The transaction involves the creation or opening of a new ConvexNQS+ log file, and the create or open operation failed.

The transaction failed.

TCMP_MAXNETCONN **Maximum network connection limit reached at transaction peer. Retry later.**

The transaction cannot begin because there are too many ongoing network transactions at the peer machine.

The transaction failed; retry later.

TCML_MAXQDESTS **Maximum destination set cardinality reached at local host.**

The transaction involves the addition of a new queue destination for a pipe queue, and that addition exceeds the maximum number of queues allowed within a destination set for a single pipe queue. The maximum number of queue destinations in a destination set is 100.

The transaction failed.

TCMP_MIDCONFLCT **Machine id conflict between client and peer at transaction peer. Seek staff support.**

The transaction cannot begin because the machine IDs of the peer (server) machine and the local (client) machine do not match.

The transaction failed. Mark the remote site as failed, and do not retry.

TCML_NETDBERR **Local network database error at local host. Seek staff support.**

This completion code is returned when an error or an inconsistency is detected while a local client process accesses the network database on the client machine.

The transaction failed; do not retry.

TCMP_NETDBERR **Local network database error at transaction peer. Seek staff support.**

This completion code is returned when an error or an inconsistency is detected while a server process accesses the network database on the server peer machine.

The transaction failed; do not retry.

TCML_NETNOTSUPP **Networking not supported by implementation at local host.**

The transaction cannot begin because the local host implementation of ConvexNQS+ does not support the networking implementation required to perform the transaction.

The transaction failed; do not retry.

TCMP_NETPASSWD **Network password verification failure at transaction peer. Seek staff support.**

The transaction cannot begin because the client failed to supply the proper ConvexNQS+ network server password to the remote server peer machine.

The transaction failed; do not retry.

TCML_NOACCAUTH **No account authorization at local host.**

The transaction cannot be performed because there is no account database authorization at the local machine for the user associated with the transaction.

The transaction failed; do not retry.

TCMP_NOACCAUTH **No account authorization at transaction peer.**

The transaction cannot be performed because there is no account database authorization at the peer machine for the user associated with the transaction.

The transaction failed; do not retry.

TCML_NOACCNOW **Does not have access now at local host.**

The transaction involves the deletion of a user or group ID from a queue access set, and the specified user or group ID is not already present in the set.

The transaction failed.

TCML_NOESTABLSH **Unable to make connection with ConvexNQS+ daemon at local host. Retry later.**

The transaction cannot be performed because the connection between the client transaction process and the

local ConvexNQS+ daemon at the local machine cannot be established. The nqs daemons/net daemons are not running.

The transaction failed; retry later.

TCMP_NOESTABLSH **Unable to make connection with ConvexNQS+ daemon at transaction peer. Retry later.**

The transaction cannot be performed because the connection between the transaction server and peer ConvexNQS+ daemon at the peer machine cannot be established.

The transaction failed; retry later.

TCML_NOLOCALDAE **ConvexNQS+ local daemon is not present at local host. Retry later.**

The transaction cannot be performed because the ConvexNQS+ daemon at the local machine is not running.

The transaction failed; retry later.

TCMP_NOLOCALDAE **ConvexNQS+ local daemon is not present at transaction peer. Retry later.**

The transaction cannot be performed because the ConvexNQS+ daemon at the peer machine is not running.

The transaction failed; retry later.

TCML_NOMOREPROC **Insufficient processes to perform transaction at local host. Retry later.**

The transaction cannot begin because there are not enough processes available on the local host to perform the transaction.

The transaction failed; retry later.

TCMP_NOMOREPROC **Insufficient processes to perform transaction at transaction peer. Retry later.**

The transaction cannot begin because there are not enough processes available on the remote peer machine to perform the transaction.

The transaction failed; retry later.

TCMP_NONETDAE **ConvexNQS+ net daemon is not present at transaction peer. Retry later.**

The transaction cannot be performed because the ConvexNQS+ netdaemon at the peer machine is not running.

The transaction failed; retry later.

TCMP_NONSECPORT **Non-secure network port verification error at transaction peer. Seek staff support.**

The transaction cannot begin because the client process connected to the remote ConvexNQS+ network daemon process using a nonsecure port.

The transaction failed. Mark the client server as failed, and retry the transaction after the client server program is repaired.

TCML_NOPORTAVAI **No network port available at local host. Retry later.**

The transaction cannot begin because there is no available network port to perform the network operations required by the transaction at the local machine.

The transaction failed; retry later.

TCML_NOSUCHACC **No such account at local host.**

The transaction involves a nonexistent account on the local machine. The account must exist for the transaction to complete.

Alternatively, an invalid account name was given to the `qmgr` add manager or add users command. When using an account name, the account must currently exist in /etc/passwd on the local machine. If the [uid] syntax is used, the account need not currently exist.

The transaction failed.

TCML_NOSUCHCPU

The transaction involves a reference to a nonexistent CPU at the local machine.

The transaction failed.

TCML_NOSUCHALI **No such queue alias at local host.**

The transaction involves an operation that references a nonexistent queue alias at the local machine.

The transaction failed; do not retry.

TCML_NOSUCHDES **No such destination at local host.**

The transaction involves a nonexistent queue destination for a pipe queue at the local machine.

The transaction failed.

TCML_NOSUCHDEV

The transaction involves a reference to a nonexistent device for a local device queue.

The transaction failed.

TCML_NOSUCHFORM

The transaction refers to nonexistent NQS device forms at the local machine.

The transaction failed; do not retry.

TCMP_NOSUCHFORM

The transaction refers to nonexistent NQS device forms at the peer machine.

The transaction failed; do not retry.

TCML_NOSUCHGRP **No such group at local host.**

The transaction involves a nonexistent group at the local machine.

Alternatively, an invalid group name was given to the qmgr add groups command. When using a group name, the group must currently exist in the /etc/group. If the GID syntax is used, the GID need not currently exist.

The transaction failed.

TCML_NOSUCHMAC **No such machine.**

The transaction involves a machine that is not known at the local host.

The transaction failed.

TCML_NOSUCHMAN **No such manager/operator at local host.**

The transaction involves a nonexistent ConvexNQS+ manager or operator account.

Or an attempt was made to delete a ConvexNQS+ manager using the qmgr delete manager command when the specified account does not currently have manager privileges.

The transaction failed.

TCML_NOSUCHMAP

The transaction refers to a nonexistent queue or device mapping.

The transaction fails.

TCML_NOSUCHQUE **No such queue at local host.**

The transaction involves an operation that references a nonexistent queue at the local machine.

The transaction failed; do not retry.

TCMP_NOSUCHQUE **No such queue at transaction peer.**

The transaction involves an operation that references a nonexistent queue at the peer machine.

The transaction failed; do not retry.

TCML_NOSUCHQUO **No such quota supported at local host.**

The transaction involves setting a batch queue quota limit that cannot be enforced at the local machine.

The transaction failed.

TCML_NOSUCHREQ **No such request at local host.**

The transaction involves a request that does not exist at the local machine.

The transaction failed; do not retry.

TCMP_NOSUCHREQ **No such request at transaction peer.**

The transaction involves a request that does not exist at the peer machine.

The transaction failed; do not retry.

TCML_NOSUCHSIG **No such signal at local host.**

The transaction involves a signal that does not exist at the local machine.

The transaction failed; do not retry.

TCMP_NOSUCHSIG **No such signal at transaction peer.**

The transaction involves a signal that does not exist at the peer machine.

The transaction failed; do not retry.

TCML_NOTREQOWN **Not request owner at local host.**

The transaction involves an operation on a request when the user associated with the transaction is not the request owner at the local machine.

Alternatively, a user without operator or manager privileges tried to perform one of the following `qmgr` commands on a request not owned by this user:

- `modify request priority`
- `move request`
- `chkpnt request`
- `delete request`

The transaction failed; do not retry.

TCMP_NOTREQOWN **Not request owner at transaction peer.**

The transaction involves an operation on a request when the user associated with the transaction is not the request owner at the peer machine.

The transaction failed; do not retry.

TCML_PATHLEN **Resolved output filename for batch request at local host exceeds the maximum supported path length.**

The transaction cannot complete because a local transaction process reserved a queue slot on the local machine for a new batch request, and the resolved standard output or standard error path name of the request exceeds the maximum request path length supported by the ConvexNQS+ implementation at the local machine.

The transaction failed; do not retry; delete the associated request.

TCMP_PATHLEN **Resolved output filename for batch request at transaction peer exceeds the maximum supported path length.**

The transaction cannot complete because a remote transaction process reserved a queue slot on the remote machine for a new batch request, and the resolved standard output or standard error path name of the request exceeds the maximum request path length supported by the ConvexNQS+ implementation at the remote machine.

The transaction failed; do not retry; delete the associated request.

TCML_PEERARRIVE **Request arriving at local host.**

The transaction involves deleting a request that is in transit between two machines in the network, and the request is presently in the arriving state (to be delivered or is being delivered from a remote machine).

The transaction failed, and the coordinator process attempting the transaction must now retry the delete operation using the networked form of the transaction.

TCML_PEERDEPART **Request departing at local host.**

The transaction involves deleting a request that is in transit between two machines in the network, and the request is presently in the departing state (to be delivered or is being delivered to a remote machine).

The transaction failed, and the coordinator process attempting the transaction must now retry the delete operation using the networked form of the transaction.

TCML_PROTOFAIL **ConvexNQS+ protocol failure at local host. Seek staff support.**

The transaction cannot complete because of a ConvexNQS+ message protocol failure on the local machine.

The transaction failed; do not retry.

TCMP_PROTOFAIL **ConvexNQS+ protocol failure at transaction peer. Seek staff support.**

The transaction cannot complete because of a ConvexNQS+ message protocol failure between:

- A local client transaction process and a remote server transaction process
- The remote server process and the network daemon at the remote machine

The transaction failed; do not retry.

TCML_QUEDISABL **Queue is disabled at local host. Retry later.**

The transaction involves an operation that cannot complete because the referenced queue is disabled at the local machine.

The transaction failed; retry later.

TCMP_QUEDISABL **Queue is disabled at transaction peer. Retry later.**

The transaction involves an operation that cannot complete because the referenced queue is disabled at the peer machine.

The transaction failed; retry later.

TCML_QUEENABLE **Queue is enabled at local host.**

The transaction involves deleting a local queue that is not disabled.

The transaction failed.

TCML_QUEHASREQ **Queue has request(s) at local host.**

The transaction involves deleting a local queue that is not empty.

The transaction failed.

TCML_QUOTALIMIT **Explicit request quota limits exceed maximums at local host.**

The transaction involves queuing a batch request or changing a batch request limit that exceeds one or more of the corresponding quota limits for that batch queue at the local machine.

Quota information bits are present.

The transaction failed; do not retry.

TCMP_QUOTALIMIT **Explicit request quota limits exceed maximums at transaction peer.**

The transaction involves queuing a batch request or changing a batch request limit that exceeds one or more of the corresponding quota limits for that batch queue at the peer machine.

Quota information bits are present.

The transaction failed; do not retry.

TCML_REQCOLLIDE **Attempt to queue already existing request at local host.**

The transaction involves queuing a request when the request ID already exists at the local machine.

The transaction failed; do not retry; delete the associated request.

TCMP_REQCOLLIDE **Attempt to queue already existing request at transaction peer.**

The transaction involves queuing a request when the request ID already exists at the peer machine.

The transaction failed; do not retry; delete the associated request.

TCML_REQDELETE **Request deleted at local host.**

The transaction involves deleting a specific request at the local machine, and the request was deleted successfully.

The transaction succeeded.

TCMP_REQDELETE **Request deleted at transaction peer.**

The transaction involves deleting a specific request at the peer machine, and the request was deleted successfully.

The transaction succeeded.

TCML_REQMOVSTA **Request is not in an allowable state at local host.**

The transaction involves moving a request from one queue to another when the request is not queued, holding, or waiting.

The transaction failed; do not retry.

TCML_REQNOTHOLD **Request is not being held at local host.**

The transaction involves releasing a held request, but the target request is not in the holding state.

The transaction failed; do not retry.

TCML_REQNOTQUE **Request is not in the queued state at local host.**

The transaction involves placing a queued request on hold, but the target request is not in the queued state.

The transaction failed; do not retry.

TCML_REQOPHOLD **Request is being held by operator at local host.**

The transaction involves releasing a held request, but the request is being held by an operator.

The transaction failed; do not retry.

TCML_REQRUNNING **Request is running at local host.**

The transaction involves deleting a specific request at the local machine, but the request is already running, and the transaction did not specify a signal. The request is not deleted and continues running.

The transaction failed; do not retry.

TCMP_REQRUNNING **Request is running at transaction peer.**

The transaction involves deleting a specific request at the peer machine, but the request is already running, and the transaction did not specify a signal. The request is not deleted and continues running.

The transaction failed; do not retry.

TCML_REQSIGNAL **Request was running and has been signalled at local host.**

The transaction involves deleting or signalling a specific request at the local machine.

The transaction succeeded.

TCMP_REQSIGNAL **Request was running and has been signalled at transaction peer.**

The transaction involves deleting or signalling a specific request at the peer machine.

The transaction succeeded.

TCML_ROOTINDEL **Root cannot be deleted at local host.**

The transaction involves deleting root as a ConvexNQS+ manager at the local machine.

The transaction failed.

TCMP_RRFUNKNMID **Request refers to unknown machines at transaction peer.**

The transaction involves queuing a request at a queue destination when the request refers to machine IDs not known to the transaction peer machine.

The transaction failed; do not retry.

TCML_SELMIDUNKN **Local machine id unknown at local host. Seek staff support.**

The transaction cannot be performed because the local client transaction process at the local machine cannot determine the machine ID of the local machine.

The transaction failed; do not retry. If the transaction is on behalf of a previously-queued ConvexNQS+ request, stop the containing queue and requeue the request.

TCMP_SELMIDUNKN **Local machine id unknown at transaction peer. Seek staff support.**

The transaction cannot be performed because the local server transaction process at the peer machine cannot determine the machine ID of the peer machine.

The transaction failed. Mark the remote site as failed, and do not retry.

TCML_SELREFDES **Attempt to create self-referential destination at local host.**

The transaction involves adding a pipe queue destination that creates a self-referential pipe queue destination set. This means that the destination set for a pipe queue contains the name of the pipe queue.

The transaction failed; do not retry.

TCML_SHUTERROR **Shutdown error at local host.**

The transaction involves shutting down the local ConvexNQS+ daemon at the local host, and an error occurred in the shutdown process.

The transaction failed.

TCML_SUBMITTED **Request successfully submitted at local host.**

The transaction involves queuing a request at the local machine, and the queuing transaction was successful.

Quota information bits are present.

The transaction succeeds.

TCMP_SUBMITTED **Request successfully submitted at transaction peer.**

The transaction involvsd queuing a request at the peer machine, and the queuing transaction was successful.

Quota information bits are present.

The transaction succeeds.

TCML_TOOMANDEV

The transaction involves the creation of a new device that exceeds the maximum number of devices supported by the local NQS implementation.

The transaction failed.

TCML_UNAFAILURE **Unanticipated transaction failure at local host.**

An unanticipated and totally unexpected error condition occurred when ConvexNQS+ tried to process the transaction at the local machine.

This is a generic transaction completion code and can be used as the code from the local machine for any transaction when a more specific code does not exist.

The transaction failed; do not retry.

TCMP_UNAFAILURE **Unanticipated transaction failure at transaction peer.**

An unanticipated and totally unexpected error condition occurred when ConvexNQS+ tried to process the transaction at the peer machine.

This is a generic transaction completion code and can be used as the final result code from the local machine for any transaction when a more specific code does not exist.

The transaction failed; do not retry.

TCML_UNRESTR **Access is currently unrestricted.**

The transaction involves setting a queue access to unrestricted, and the queue access is already unrestricted.

The transaction failed.

TCML_WROQUETYP **Wrong queue type for transaction at local host.**

The transaction involves

- Queuing a request, but the target queue on the local machine is the wrong type.
- Trying an operation that only relates to a pipe queue on a batch queue.

The transaction failed; do not retry.

TCMP_WROQUETYP **Wrong queue type for transaction at transaction peer.**

The transaction involves

- Queuing a request, but the target queue on the peer machine is the wrong type.
- Trying an operation that only relates to a pipe queue on a batch queue.

The transaction failed; do not retry.

Limit violation flags

The following are the quota limit violation information flags:

Table 7 Limit violation flags

Flag name	Violation
TCI_PP_CFLEXC	Per-process core-file size limit
TCI_PP_CTLEXC	Per-process CPU time limit
TCI_PP_DSLEXC	Per-process data-segment limit
TCI_PP_MSLEXC	Per-process memory size limit
TCI_PP_NELEXC	Per-process nice execution priority limit
TCI_PP_PFLEXC	Per-process permanent file size limit
TCI_PP_SSLEXC	Per-process stack-segment limit
TCI_PP_TFLEXC	Per-process temporary file size limit
TCI_PP_WSLEXC	Per-process working set quota limit
TCI_PR_CTLEXC	Per-request CPU time limit
TCI_PR_MSLEXC	Per-request memory space limit
TCI_PR_NCPEXC	Per-request CPU quota limit
TCI_PR_PFLEXC	Per-request permanent file space limit
TCI_PR_TFLEXC	Per-request temporary file space limit
TCI_NOIMPORT	Import resource not available

Request completion messages

B

This appendix

- Lists common codes that may be returned by a request
- Discusses the meaning of the codes
- Describes actions you should take in response to the codes

These codes include error and success codes. The code format is

RCM_*code, message_text*

where

code is the mnemonic code for the error message.

message_text is the text explaining the reason for the error.

The codes in this appendix are listed alphabetically by the first letter in *code*. Each is followed by explanatory text detailing:

- The cause of the error
- The result
- In some cases, the action you should take

Some explanatory text may say *Quota information bits are present*. This indicates that the error involved a violation of a quota limit, and ConvexNQS+ displays another error message with additional information about the limit violation. These additional messages are listed at the end of this appendix.

RCM_2MANYENVARS **Too many environment variables to run batch request. Request not executed. Request deleted.**

The request cannot be performed because there are too many environment variables to run the request.

No output files are returned.

The request is deleted.

RCM_2MANYSVARGS **Too many server arguments on server execve(). Request requeued.**

The request cannot be performed because there are too many server arguments in server execve().

No output files are returned.

The request is requeued, and the queue is stopped.

RCM_ABORTED **Request aborted via a signal. Request deleted.**

The request cannot be performed because a shell process terminated as a result of a signal unrelated to ConvexNQS+ shutting down (unless something unpredictable happens, such as the request received SIGTERM signals and then killed itself with some signal besides SIGTERM or SIGKILL).

Output files (if any) are queued for return.

The request is deleted.

RCM_BADCDTFIL **Corrupted request control and/or data files.**

Request not executed.

Request files placed in ConvexNQS+ failed directory.

The request involves an operation on request control file(s) or data file(s), and the request control or data file(s) is corrupt.

No output files are returned.

The request is placed in the failed directory on the local machine for later analysis.

RCM_BADSRVARG **Bad argument passed to request server. Request requeued.**

The request cannot be performed because of a bad argument or environment variable.

No output files are returned.

The request is requeued, and the queue is stopped.

RCM_DELIVERED **Request has been successfully delivered to destination.**

The pipe queue successfully delivered the request to its destination.

No output files are returned.

The request remains on the destination queue.

RCM_DELIVEREXP **Request delivery time expired. Request deleted.**

The request cannot be performed because the pipe queue server has not delivered it to its destination.

This message is converted from RCM_RETRYLATER when the delivery time expires.

No output files are returned.

The request is deleted.

RCM_DELIVERFAI **Request could not be delivered. Request deleted.**

The request cannot be performed because the pipe queue server has not delivered it to its destination.

This message indicates a disastrous situation in which the request can never be delivered. The information field of this return code must contain the TCM_ code that caused the failure.

No output files are returned.

The request is deleted.

RCM_DELIVERRETX **Request was not delivered. Retry limit exceeded. Request deleted.**

The request cannot be performed because the pipe queue server has not delivered it to its destination, and the retry limit has been reached.

The information field of this return code must contain the TCM_ code that caused the failure.

No output files are returned.

The request is deleted.

RCM_ENFILERUN **Unable to successfully start request because of a file descriptor shortage.**

Request requeued.

The request cannot be performed because of a file descriptor shortage on the local machine.

No output files are returned.

The request is requeued to run later. All ConvexNQS+ queues on the local machine are stopped to conserve any remaining available file descriptors.

RCM_ENOSPCRUN **Unable to successfully start request because of a file system resource shortage. Request requeued.**

The request cannot be performed because of a file system resource shortage on the local machine.

No output files are returned.

The request is requeued to run later. All ConvexNQS+ queues on the local machine are stopped so as not to place additional file system space demands on the local machine.

RCM_EXECUTING **Request executing.**

This code is used in the internal message protocol between the server and shepherd processes when spawning a ConvexNQS+ batch request.

This request completion code must never be returned.

RCM_EXITED **Request exited normally.**

The batch request server exited normally.

Output files are queued for return.

Local information concerning the request is deleted.

RCM_IMPORTFAI **NFS import of directory failed. Request deleted.**

The request cannot be performed because *nqsimport* cannot perform the import.

Output files are queued for return.

The request is deleted.

RCM_INSUFFMEM **Insufficient memory to start request. Request requeued.**

The request cannot be performed because of insufficient memory or swap space to spawn a server.

No output files are returned.

The request is requeued, and all queues are stopped.

RCM_INTERRUPTED **Server transaction processing aborted for ConvexNQS+ shutdown. Request requeued.**

The request cannot be performed because a network queue or pipe queue server stopped transaction processing when ConvexNQS+ shut down.

No output files are returned.

The request is requeued.

RCM_MIDUNKNOWN **Machine-id of request owner is no longer recognized by the execution machine. Requestnot executed. Request deleted.**

The request cannot be performed because a local batch shell process, local network queue server, or local pipe server cannot identify the machine from which the request originated.

When the request was originally accepted by the local system, the owner machine ID was known to the local system; however, since that time, the local host tables have changed, and the owner MID of the request is no longer recognized.

Output files are queued for return.

The request is deleted.

RCM_NETREQDEL **Request could not be successfully delivered.**

Previously routed request expired or was deleted at destination. Request deleted.

The request cannot be performed because the pipe queue server has not delivered it to its destination.

No output files are returned.

The request is deleted.

RCM_NOACCAUTH **No account authorization on execution machine for mapped request owner user-id. Request not executed. Request deleted.**

The request cannot be performed because the local machine has no password entry for the request owner user ID.

This can occur if the local password database is changed.

Output files are queued for return.

The request is deleted.

RCM_NOMOREPROC **Insufficient processes available to spawn request. Request requeued.**

The request cannot be performed because of a process shortage on the local machine.

No output files are returned.

The request is requeued to run at a later time. All ConvexNQS+ queues on the local machine are stopped to conserve remaining available processes.

RCM_NONSECPORT **Server bind() error. Request requeued.**

The request cannot be performed because the pipe queue or network queue server connected to the transaction server on a nonsecure port. The queue server is flawed.

No output files are returned.

The request is requeued, and the containing queue is stopped.

RCM_NORESTART **Interrupted request prohibits restart on ConvexNQS+ rebuild. Request deleted.**

The request cannot be performed because it was running at the time of a ConvexNQS+ shutdown or system crash, but is not defined as restartable.

This completion code is returned directly from the local ConvexNQS+ daemon and can only occur when ConvexNQS+ reboots.

Output files are queued for return.

The request is deleted.

RCM_NOSVRETCODE **Server for request did not return a completion code. Request failed.**

Request files placed in ConvexNQS+ failed directory.

The request cannot be performed because a network queue server or pipe queue server failed to report a completion code when it exited.

No output files are returned.

The request is placed in the failed directory, and the appropriate device or queue is stopped.

RCM_PATHLEN **Resolved stdout or stderr pathname of batch request at destination exceeds the maximum supported length. Request deleted.**

The request cannot be performed because the standard output or standard error path name, when resolved by the selected pipe queue destination, exceeds the maximum request path length supported by the ConvexNQS+ implementation at the receiving machine.

No output files are returned.

The request is deleted.

RCM_PIPREQDEL **Request deleted.**

The request cannot be performed because routing was interrupted by a delete request at the local machine.

No output files are returned.

The request is deleted.

RCM_REBUILDFAI **ConvexNQS+ rebuild failure. Request could not be requeued. Request deleted.**

The request in queue when ConvexNQS+ shut down cannot be performed because an internal ConvexNQS+ error is preventing it from requeuing.

This completion code is returned directly from the local ConvexNQS+ daemon and can only occur when ConvexNQS+ reboots.

No output files are returned.

The request is placed in the ConvexNQS+ failed directory.

RCM_REQCOLLIDE **Request collided with another previously existing request**

with the same request-id. The newer request has been deleted. Seek staff support.

The request cannot be performed because the request ID already exists on the destination machine.

No output files are returned.

The request is deleted.

RCM_RETRYLATER **Request transaction failed. Retry scheduled. Request requeued.**

The request cannot be performed and will be retried later.

No output files are returned.

RCM_ROUTED **Request successfully routed for delivery to destination.**

The request was successfully routed and queued at one of the remote destinations for the request, as selected by the pipe queue.

Quota information bits are present.

No output files are returned.

RCM_ROUTEDLOC **Request sent to local queue destination.**

The request was successfully routed and queued at one of the local destinations for the request, as selected by the pipe queue.

Quota information bits are present.

No output files are returned.

RCM_ROUTEEXP **Request routing time expired. Request deleted.**

The request cannot be performed because it has not been routed to a destination.

This message is converted from RCM_RETRYLATER when the routing time expires.

No output files are returned.

The request is deleted.

RCM_ROUTEFAI **Request could not be routed. Request deleted.**

The request cannot be routed because no destination will accept it.

Information bits that define the set of failure conditions are present.

No output files are returned.

The request is deleted.

RCM_ROUTERETX **Request was not routed. Retry limit exceeded. Request deleted.**

The request cannot be routed to any destination, and the retry limit for this type of transaction has been reached.

Failed pipe routing information bits that define the set of failure conditions are present.

No output files are returned.

The request is deleted.

RCM_SEREXEFAI **Request server execve() failed. Request requeued.**

The request cannot be performed because the server execve() operation failed.

No output files are returned.

The request is requeued, and the queue is stopped.

RCM_SERVESIGERR **Server killed by unanticipated signal. Request requeued.**

The request cannot be performed because a network or pipe server was killed by a signal that should not have killed it.

No output files are returned.

The request is requeued, and the device or queue is stopped as appropriate.

RCM_SHEXEF2BIG **Too many environment variables to run request. Request deleted.**

The request cannot be performed because one or both of the argv[] or envp[] argument sets to the batch request shell exceeds the maximum size supported by the underlying UNIX implementation.

Output files are queued for return.

The request is deleted.

RCM_SHUTDNABORT **Request aborted for ConvexNQS+ shutdown.**

The request was defined as unrestartable, and so the request has been deleted.

The request cannot be performed because a server or shell process exited as a result of a signal sent during ConvexNQS+ shutdown, and the request is not restartable.

Output files are queued for return.

The request is deleted.

RCM_SHUTDNREQUE **Request aborted for ConvexNQS+ shutdown.**

The request has been requeued for later restart.

The request cannot be performed because a server or shell process exited as a result of a signal sent during ConvexNQS+ shutdown, but the request is restartable.

When ConvexNQS+ reboots, the request is requeued for continued execution.

RCM_SSHEXEFAI **Request shell execve() failed. Request requeued.**

The request cannot be performed because an attempt to execute the shell chosen by the system to interpret the batch request shell script failed.

Output files are queued for return.

The request is requeued, and the queue is stopped.

RCM_STAGEOUT Output file successfully returned to destination.

The output file successfully returned to its intended destination.

The output file is deleted.

RCM_STAGEOUTBAK **Output file could not be returned to primary destination.**

Output file successfully returned to backup destination in user home directory on the execution machine.

The output file cannot return to its intended destination because circumstances prevented retry attempts. Instead, the file successfully returned to its backup destination.

The information field of this return code must contain the TCM_ code that caused the failure at the primary destination.

The output file is deleted.

The request completes.

RCM_STAGEOUTFAI **Output file could not be returned to primary or backup destination.**

The output file cannot return to its intended or backup destination because of error conditions that prevented retry attempts.

The information field of this return code must contain the TCM_ code that caused the failure at the primary destination.

The output file must be deleted.

RCM_UNABLETOEXE **Unable to execute request. Request deleted.**

The request cannot not be performed for reason(s) identified in the information bit vector of the completion code.

Quota information bits are defined.

Output files are queued for return.

The request is deleted.

RCM_UNAFAILURE **Request failed.**

Request files placed in ConvexNQS+ failed directory.

The request cannot be performed because an unanticipated error condition occurred while ConvexNQS+ tried to run the request or perform a transaction operation for the request.

If ConvexNQS+ is returning an output file to the submitter when this error occurs, the associated output file is deleted, and the output file loss is recorded for the request.

Otherwise, the request is placed in the failed directory, and any associated output files are deleted.

RCM_USHBRKPNT **Unsupported shell breakpoint encountered.**

Request deleted.

The request cannot be performed because a shell chosen by the user to execute the request stopped at a breakpoint.

Output files are queued for return.

The request is deleted.

RCM_USHEXEFAI **Request shell execve() failed. Request deleted.**

The request cannot be performed because an attempt to run the shell chosen by the system to interpret the request shell script failed.

Output files are queued for return.

The request is deleted.

Table 8 lists request completion flags.

Table 8 Request completion flags

Flag name	Description
RCI_ACCESSDEN	Access denied
RCI_CLIMIDUNKN	Client machine ID is unknown at transaction peer
RCI_EFBIG	File size limit exceeded
RCI_FATALABORT	Nonrecoverable transaction failure
RCI_MIDCONFLICT	Machine ID conflict between client and destination
RCI_NETNOTSUPP	Networking not supported at ConvexNQS+ site.
RCI_NETPASSWD	Network password verification error
RCI_NOSUCHQUE	No such queue
RCI_PEERINTERR	ConvexNQS+ internal error at transaction peer
RCI_PEERMIDUNKN	Local machine ID is unknown at transaction peer
RCI_PEERNETDB	Network database error at transaction peer
RCI_PEERNOACATH	No account authorization at transaction peer
RCI_PROTOFAIL	ConvexNQS+ protocol failure
RCI_QUOTALIMIT	Explicit request quota limits exceed maximums
RCI_RRFUNKNMID	Request refers to machine IDs unknown at transaction peer
RCI_WROQUETYP	Wrong queue type for request
RCI_UNAFAILURE	Unanticipated transaction failure

Index

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

Notes

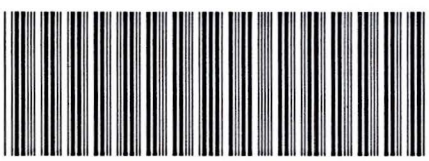
Notes

Notes

Notes

ORDER NUMBER
DSW-861

DOCUMENT NUMBER
770-007530-000



CONVEX PRESS