CONVEX CXbatch System Manager's Guide

Third Edition

CONVEX COMPUTER CORPORATION

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

CONVEX
CXbatch System Manager's Guide for ConvexOS

Order No. DSW-182

Third Edition
January 1992

CONVEX

CXbatch System Manager's Guide for ConvexOS

Order No. DSW-182

Printed in the United States of America

Revision information for

CONVEX CXbatch System Manager's Guide for ConvexOS

Edition	Document No.	Description
Third	710-006730-004	Released with CONVEX CXbatch V2.1. Contains CXbatch concepts, description of status utilities output, and new qmgr command that copies open files when job is checkpointed.
Second	710-006730-003	Released with CONVEX CXbatch V2.0. Contains information about integration with CONVEX Share Scheduler and Checkpoint Restart, enhancements to existing functionality, new qmgr options, and enhanced accounting features.
First Edition, Rev1	710-006730-001	Released with CONVEX CXbatch V1.1, April 1990.
First	710-000040-201	Released with CONVEX CXbatch V10.0, February 1989. Initial release.

Contents

Figures

Tables

Using this guide

Purpose and audience

The *CONVEX CXbatch System Manager's Guide* describes basic CXbatch concepts, how to configure your CXbatch network, and how to maintain the queues and queue requests on a regular basis. This information is required by CXbatch managers and operators.

How to use this guide

If you have never used CXbatch, read Chapter 1 for basic CXbatch concepts before attempting to perform any of the tasks described in this book.

If you are configuring CXbatch for the first time, or if you want to make changes to the existing configuration, read Chapter 2.

To maintain queues and queue requests, read Chapters 3 and 4.

To generate accounting reports, read Chapter 5.

For an alphabetical listing of the commands available with qmgr and qmapmgr utilities, and how to use them, read Chapters 6 and 7.

Notational conventions

This section discusses notational conventions used in this book.

General conventions

The following conventions are used in this guide:

- *Italics*
  - Designate user-supplied variables in a command-line example
  - Indicate document titles
- `Constant-width font` designates input that must be typed exactly as it appears and output displayed on the terminal screen. This includes:
  - Command names and options
  - Directives, program statements, display examples, printout examples, and error messages returned.
- Horizontal ellipsis (...) shows repetition of the preceding item(s).
- 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 the *ConvexOS Programmer's Reference* appear in the form adb(1), where the name of the man page is followed by its section number enclosed in parentheses.

Command syntax

In order to use the commands in this document, you must understand the conventions used when describing command syntax. Consider this example:

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

1. `Constant-width font` indicates that you must type the characters exactly as they appear (uppercase and lowercase are identical). The letters that appear in uppercase indicate the minimum amount you must type to make the command unique. However, they do not have to be typed in uppercase.
2. *Italics* indicate a variable that must be supplied by the user. In this case, the user must supply the name of an *input_file*.
3. Square brackets [ ] indicate optional data. Horizontal ellipsis (...) shows repetition of the preceding item(s). In this case, the user can optionally specify more than one *input_file* on the command line.
4. Curly brackets { } indicate a choice. The choices available are shown inside the curly brackets and separated by the pipe (|) sign. In this case, the user can enter either `a` or `b`.
5. [*output_file*] indicates the user can optionally specify an output file name with the command.

Associated documents

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

For more information on the ConvexOS operating system, you can order the following books from CONVEX Computer Corporation:

- *Managing ConvexOS: Configuration Guide* (DSW-030). Contains information for configuring system resources and provides background information and concepts necessary to make configuration decisions.
- *CONVEX CXbatch User's Guide* (DSW-183). Describes how to use CXbatch to submit, track, and control jobs for batch execution in queues.
- *CONVEX COVUEbatch Guide* (DSW-151). Explains how to submit jobs from a VAX to a Convex system; how to display and remove jobs from a CONVEX batch queue; and how to initialize, start, and delete COVUEbatch queues.
- *CONVEX Share Scheduler System Manager's Guide* (DSW-265). Explains how to use Share utilities to configure, tune, and maintain Share.

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.

CXbatch processing

CONVEX CXbatch permits users to submit jobs to a queue for batch execution. A queue is a list of batch requests that are ready and waiting to execute. A batch request is one or more commands submitted by a user or a user program to a batch queue. These commands are usually executed after a certain time or event has passed and do not require further interaction with the user. A typical batch request is a program containing commands that perform large-scale, detailed computations on a static data file.

Users can submit batch jobs for execution on either the local machine or a remote machine configured with CXbatch or another version of Network Queueing System (NQS).

When a user submits a request to CXbatch, CXbatch assigns it to a queue and assigns it a priority. The request waits in the queue until it is selected for execution. Requests are selected according to their priority. Once selected, CXbatch executes it and routes any output to the specified recipient.

This chapter introduces CXbatch features and describes how CXbatch processes jobs submitted to the queues.

Queue types

There are two types of CXbatch queues:

- **Batch**—Batch queues are used only to execute batch requests. They hold requests for scheduled, perhaps delayed, processing by subsystems within CXbatch.
- **Pipe**—Pipe queues are routing queues that do not directly process requests but, instead, transmit requests to other queues for execution on either the same or a remote machine. Each pipe queue has a set of destination queues that are possible recipient queues for that pipe queue. A destination queue can be either a batch queue or another pipe queue.

The flow of a job through each queue type is explained in detail in the following sections.

Batch queue processing

A user submits a job request to CXbatch using the `qsub` command. If the request is submitted to a local batch queue, `qsub` sends the request to the nqsdaemon.

The nqsdaemon checks any qualifiers included with the `qsub` command against defined queue limits and attributes. The nqsdaemon also checks the ability of the queue to import files and determines whether the recipient queue is a pipe-only queue. Based on these checks, nqsdaemon accepts or rejects the request. If the request is accepted, nqsdaemon runs the job when its turn comes in the batch queue.

If the request is submitted to a remote batch queue, `qsub` sends the request to the remote netdaemon. The netdaemon spawns a netserver, which attempts to queue the request with the nqsdaemon. nqsdaemon checks to make sure that the user has an account on the remote machine and that the user name on the local machine matches the user name on the remote machine. If these conditions are true, it accepts the request.

The remote nqsdaemon checks any qualifiers included with the `qsub` command line against the defined queue limits and attributes, checks the ability of the queue to import files, and determines if the recipient queue is a pipe-only queue. Based on these checks, nqsdaemon accepts or rejects the request.

When nqsdaemon determines that a request should be run, the daemon spawns a shepherd process. This shepherd process sets up the environment for the batch request and runs the job. It sends mail to the user if it cannot execute the shell script submitted. The shepherd process:

- Waits for the job to complete

- Logs accounting information
- Undoes anything it set up to run the job (such as unmounting remote file systems mounted to import files)
- Returns output files to the user's directory

Pipe queue processing

A user submits a job request to CXbatch using the `qsub` command. If the request is submitted to a local pipe queue, `qsub` sends the request to the nqsdaemon.

The nqsdaemon checks the `qsub` qualifiers against defined queue limits and attributes and determines whether the recipient queue is a pipe-only queue. Based on these checks, nqsdaemon accepts or rejects the request. If the request is accepted, nqsdaemon routes the request when the request moves to the top of the queue.

If the request is submitted to a remote pipe queue, `qsub` sends the request to the remote netdaemon. The netdaemon spawns a netserver, which attempts to queue the request with the nqsdaemon.

nqsdaemon checks to make sure that the user identification (UID) matches the UID on the remote machine, and, if this is true, it accepts the request. The remote nqsdaemon then routes the request when it reaches the top of the queue.

When nqsdaemon determines that a job should be routed, it spawns the pipe client to handle the routing. The pipe client is a program that decides which destination queue will receive the request. The pipe client first selects a destination queue for the request. This selection is based on characteristics of the request and of each queue in the destination set defined for the pipe queue.

If the pipe client does not find a suitable destination, it returns an appropriate transaction code to nqsdaemon. If a destination is found, the pipe client contacts the netdaemon and sends the request to it. The netdaemon spawns the netserver, which in turn attempts to queue the request with the local nqsdaemon.

The two possible pipe clients are pipeclient and pipeldav (pipe load average). The system manager configures each pipe queue with a pipe client. pipeclient is the standard pipe client. It routes the request to the first queue in its destination list that is able to accept the job. pipeldav is the load-balancing pipe client. It routes the request to the queue with the lowest load factor that is able to accept the job. See the "Load balancing" section in this chapter for more details.

Load balancing

Pipe queues can be configured to route a job to a destination batch queue or another pipe queue by using the principle of load balancing. Load balancing affects only the initial placement of jobs, with a modified load average as the criterion for job placement. The modified load average is a combination of load average, processor speed, queue length, and a weighting factor. The formula for determining modified load average is

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

CXbatch batch queues can be fed by more than one pipe queue, whether or not the pipe queues run the load balancing pipe client. The load averaging algorithm artificially inflates the load average of batch hosts to take into account the number of jobs waiting for service in that queue.

Specifically, pipeldav needs to know how many jobs are waiting for execution in each queue. The load-balancing algorithm requires that the local subnet hosts run the rstatd software, so that load information for each batch queue host is available for the pipeldav queue host. Machines must run rstatd to participate in load balancing. rstatd is an optional NFS product; contact your CONVEX sales representative for additional information.

The pipeldav program first reads this load information for each host that it serves. It then queries the netdaemon on the host machines to obtain queue information about each destination queue on that host. The pipeldav program maintains load average information on a queue-by-queue basis. For each queue, it increments the load average by a weighting factor once for each job waiting in that queue. This weighting factor can be set by the system manager as an option of the pipeldav program; the weighting factor defaults to 1.0.

This load-balancing system places requests into queues quickly. It does not wait until a queue is empty. This way, the system does not spend unnecessary time polling the queues. Neither does it give all the jobs to a processor with a light load, because each job placed in a queue causes the load average to increase.

Configuration and control utilities

There are two CXbatch utilities that allow you to configure and operate CXbatch:

- **qmapmgr**—Used to configure the network database used by CXbatch to map machine name to machine identifier (MID). The MID is used only by CXbatch to identify machines.
- **qmgr**—Used to define and control CXbatch queues.

Both these utilities are discussed in the following sections. Detailed information on each of these functions can be found in the qmgr(8) man page, the on-line help facility within qmgr, and Chapter 6, "qmgr commands" in this manual.

The qmapmgr utility

qmapmgr is the CXbatch utility that builds and maintains the network database used by CXbatch. This database contains information on MIDs and machine names and is mandatory for operation of CXbatch. qmapmgr establishes a mapping between CXbatch and each machine in the CXbatch configuration. Without this mapping, CXbatch does not recognize remote destinations and queues and does not recognize local queues.

You can make additions, deletions, and changes to the database through qmapmgr after the database has been initially configured. Detailed information on qmapmgr can be found in the qmapmgr(8) man page and in Chapter 7, "qmapmgr commands" in this manual.

The qmgr utility

qmgr is the CXbatch utility that configures and controls CXbatch queues and requests. It controls CXbatch requests, queues, and the general CXbatch configuration on the local machine. Use the qmgr utility 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 CXbatch

Levels of authorization

qmgr is the CXbatch utility that controls CXbatch queues and requests. It allows users to control requests, track requests through the system, and with the right privileges, control CXbatch queues. The qmgr commands available to each user depend on their user type. CXbatch distinguishes three user types:

- General users
- CXbatch operators
- CXbatch managers

The following sections list and describe the commands available with each user type. The rest of this chapter provides details on the syntax and usage of these commands.

General users

General CXbatch users can track and control their own batch requests using the following qmgr commands:

`chkpnt request`	Checkpoint a request. The state of the batch request is saved into a set of checkpoint files stored in the checkpoint directory. The request continues to run.
`delete request`	Delete a request. The request can either be running or not running.
`exit`	Exit qmgr.
`help`	Get help on the commands available to them.
`hold request`	Place a request on hold, preventing its execution. The request must be in the queued state in order to place it on hold.
`modify request`	Modify the priority of a request. Users can only decrease the priority of their requests. CXbatch managers and operators can increase the priority of any user's request.
`move my_request`	Move a request from one queue to another. The request is not moved if any queue limits, access restrictions, or attributes prevent the request from being submitted to the queue.
`release request`	Release a request previously placed on hold, allowing the request to execute.

`resume request`	Resume execution of suspended request. A resumed request starts out in the queued state. Once it is about to enter the running state, it is restarted from its checkpointed state.
`show`	View the status of requests (all `show` commands). Refer to the qmgr(8) man page for a complete list of `show` commands.
`suspend request`	Temporarily suspend execution of a request. The request is checkpointed and execution is terminated. However, the request remains in the queue so it can be resumed later. If a request fails to checkpoint, it continues executing. Only checkpointable requests can be suspended.

CXbatch operators

Users with CXbatch operator privileges have access to commands that allow them to manipulate batch requests for other users, control queues, and set run limits. The CXbatch manager assigns operator privileges to a user in order for them to act as a CXbatch operator.

Managing queue requests

The commands listed under the "General users" section in this chapter are available to users with CXbatch operator privileges to track and control any user's batch requests. In addition, the following qmgr commands are available to CXbatch operators:

`move request`	Move a request from one queue to another queue. The request is moved regardless of any queue limit violations, access restrictions, or attribute violations.
`run request`	Force a request to begin executing immediately. If running the request exceeds the current run limit of the queue, the queue's run limit is increased by one until the request completes.

Managing queues

The following qmgr commands are available to CXbatch operators to manage queues:

`abort queue`	Abort all currently running requests in the queue.
`disable queue`	Prevent queue from accepting requests.
`enable queue`	Allow queue to accept requests.
`move queue`	Move all requests in one queue to another queue.
`purge queue`	Delete all queued requests from the queue.
`start queue`	Put queue into operation to allow it to execute requests.
`stop queue`	Prevent queue from executing any other requests; the currently executing request is allowed to complete.

Managing CXbatch

The following qmgr commands are available to CXbatch operators to manage CXbatch:

`set copy_open_files`	Preserve the state of open files within a process when the request is checkpointed.
`set share policy`	Specify where CPU usage is charged for jobs running in the queue.
`set per-user run limit`	Establish the per-user run limit on the queue.
`set run limit`	Establish the run limit of an existing queue.
`shutdown`	Checkpoint requests that are checkpointable and shut down CXbatch on the local machine. You must execute the qmgr utility shutdown command to shut down CXbatch before executing /etc/shutdown to shut down the system. Otherwise, requests are not checkpointed and therefore, not recoverable
`start Cxbatch`	Start CXbatch on the local machine.

CXbatch managers

Users with CXbatch manager privileges have access to all qmgr commands. Refer to the qmgr(8) man page for a complete list. By default, root is a CXbatch manager and can assign this privilege to other users.

Daemons used by CXbatch

A daemon is a process that provides a particular service when needed and is not connected to a terminal or a particular user. Daemons used by CXbatch and their related services are:

nqsdaemon	Handles all local transactions, including job submission and deletion, job scheduling, and system configuration.
netdaemon	Handles all remote transactions involving queues, including job submission and copying stdout and stderr files.
logdaemon	Is contacted by nqsdaemon and netdaemon when they need to print an error message. logdaemon sends a message to syslogd (if defined) and notifies the batch manager if the error is fatal.
netserver	Is executed by netdaemon to handle operations that require a substantial amount of time, such as queuing a request.
shepherd	A child of the nqsdaemon that watches over batch jobs; it is responsible for setting up the job environment and returning output files.
netclient	Transfers jobs to remote machines and transfers stderr and stdout to appropriate destinations.
initiator	Invokes Share Scheduler on CXbatch jobs.

Incompatibilities between NQS and CXbatch

CXbatch is based on NASA's Network Queueing System (NQS), but has many added enhancements, including checkpoint restart, load balancing, fair share scheduling for batch jobs, and batch accounting. There are six major differences between other NQS systems and CONVEX CXbatch. It is important that these differences are understood when debugging the CXbatch system or by sites that combine several different systems.

- `move my_request` does not exist in other NQS systems and can only be used within CXbatch.
- CXbatch can mount remote files using NFS; other systems cannot.
- CXbatch's `maximum_request_priority` command, if used when submitting a request between CXbatch and another NQS system, causes the priority of previously-submitted requests to be moved to a lower priority. It does not delete previously-submitted requests.
- Direct submission (for example, `qsub -q long@host`) between CONVEX machines and other machines is not possible.
- Several CXbatch `qsub` options are not applicable if the destination machine is a CONVEX machine. See the *CXbatch User's Guide* for details on these options.
- Several CXbatch packet numbers are not recognized by other NQS systems. Table 1 lists packet numbers recognized only by CXbatch.

Table 1 Packet numbers recognized by CXbatch

Packet Number	Type	Use
200	nqs	Set queue import attribute
201	nqs	Set queue description
202	nqs	Set queue a activity ID offset
203	nqs	Set queue accounting on or off
204	nqs	Set accounting log file
205	nqs	Set activity ID mask
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 with ack (used for debugging)
214	nqs	Set maximum request priority
215	nqs	Set checkpoint directory
216	nqs	Checkpoint a request
217	nqs	Set per-queue checkpoint resource
218	nqs	Force request to run
219	nqs	Suspend request
220	nqs	Resume request
221	nqs	Set share policy fixed
222	nqs	Set share policy user
223	nqs	Force run request
224	nqs	Set global per-user-run-limit
225	nqs	Set queue per-user-run-limit
205	net	Get remote queue and request information

Integration with CONVEX Share Scheduler

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

Shares are assigned by system managers. Refer to the *CONVEX Share Scheduler System Manager's Guide* for details on assigning shares to users.

COVUEbatch considerations

CONVEX COVUEbatch is an optional product that allows a user to submit batch jobs from VAX/VMS systems to remote CONVEX systems. A user submits a command procedure to COVUEbatch. COVUEbatch then uses COVUEnet (an optional CONVEX network management product) to transmit the job to CXbatch on the CONVEX system and to transmit the results of the batch job to the VMS user's home directory.

There are several items to consider if COVUEbatch is installed on your system:

- The `covue show queue` command does not display full CXbatch queue information, such as queue type (batch or pipe), queue limits, request limit, and attributes such as import, type of pipe queue, and accounting.
- If COVUEbatch is installed and running on only one machine in the CXbatch network and a user wants to submit a job to a remote queue, there must be a pipe queue on the local machine that has the remote queue as a destination. In this situation, the same performance degradation occurs as mentioned above. If COVUEbatch is installed and running on all machines that have CXbatch, however, remote queues can be accessed without use of pipe queues.
- The `covue show queue` command displays batch queues on the local machine and pipe queue destinations. The `covue delete/entry` command deletes only those entries from CXbatch queues on the local machine.
- Because COVUEbatch uses lowercase queue names, references to COVUEbatch queue names must also be lowercase.

Please refer to the *CONVEX COVUEbatch Guide* for further information.

Integration with Checkpoint Restart

Checkpoint Restart is a standard feature of ConvexOS. It permits the state of selected processes or process hierarchies to be saved to disk files and later to be restarted from the saved state. Checkpoint Restart is useful for application programs that must run for long lengths of time and that—once halted—cannot be started over from the beginning without wasting significant time and resources. Such applications can be saved to files (checkpointed) either by operator intervention or by CXbatch periodically checkpointing it. If the application is then halted, either by a system crash, by a scheduled shutdown, or by an operator, it can be restarted as it was when it was last checkpointed. If desired, the process can be restarted under the control of a debugger.

Users and operators can checkpoint and restart processes that are running under CXbatch by using Checkpoint Restart features built into CXbatch. See the *CXbatch User's Guide* for details on these features.

Configuring CXbatch 2

CXbatch is suitable for most single-machine system configurations as it is shipped and installed. Each site is different, however, and you may want to install CXbatch on multiple machines or configure it to handle site-specific situations. You should match your work loads to the available resources to create an optimal, efficient configuration.

This chapter lists and in some cases describes tasks you must perform to install and configure the CXbatch system. Information on how to complete these tasks is provided in later sections of this chapter, unless otherwise noted. The tasks and the order in which you must perform them are:

1. Install the base CXbatch system.
2. Assign manager and operator privileges.
3. Decide on types of queues for each machine.
4. Create scheduling groups within Share Scheduler if you have Share installed in your system. See *CONVEX Share Scheduler System Manager's Guide* for details on how to do this. If you do not have Share installed in your system, omit this step.
5. Create and configure batch queues.
6. Create and configure pipe queues.
7. Delete any unnecessary queues.
8. Configure CXbatch accounting.
9. Enable Checkpoint Restart.
10. Choose a shell strategy.
11. Specify where to log CXbatch errors.
12. Schedule the automated starting and stopping of queues.
13. Notify users of changes.

Viewing queue attributes

Throughout the procedures in this chapter, it is necessary to view the attributes of queues. You can view the attributes of a queue using the `show long queue` command. You must start the qmgr utility before you can use this command. To start qmgr, enter

```
qmgr
```

The format for the `show long queue` command is

```
show long queue queuename
```

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

Figure 1 Viewing queue attributes

```
# qmgr
Mgr: show long q s
s@hostC; type=BATCH; [ENABLED, INACTIVE]; pri=48
0 exit; 0 run; 0 stage; 0 queued; 0 wait; 0 hold; 0 arrive;
Run_limit = 2; Per-user run-limit = NONE
Accounting: Off
Activity ID offset: 1
Maximum request priority: 63
Cumulative system space time = 0.000000 seconds
Cumulative user space time = 0.000000 seconds
Unrestricted access
Import directory: Yes
Checkpoint: Not available
Copy open files on checkpoint: No
Share policy fixed = s
Per-process core file size limit = UNLIMITED
Per-process data size limit = UNLIMITED
Per-process permanent file size limit = UNLIMITED
Per-process execution nice value = 0 <DEFAULT>
Per-process stack size limit = UNLIMITED
Per-process CPU time limit = 36000.0 <DEFAULT>
Per-process working set limit = UNLIMITED
```

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

```
s@hostC; type=BATCH; [ENABLED, INACTIVE]; pri=48
   ①        ②          ③          ④        ⑤
```

① Name of the queue and what host it is configured on. In this case, the attributes shown apply to the *s* queue on *hostC*.

② Type of queue. This can be:

`BATCH`	Executes CXbatch requests.
`PIPE`	Routes CXbatch requests to queues that can execute them.

③ Indicates whether or not the queue can accept requests. This can be:

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

④ Indicates whether or not the queue can execute requests. This can be:

`INACTIVE`	Requests in the queue are permitted to run; none are running.
`RUNNING`	Requests in the queue are permitted to run; some are running.
`STOPPING`	New requests sent to the queue are not permitted to run, but requests that are currently running are allowed to complete.
`STOPPED`	Requests in the queue are not permitted to run; none are running.
`SHUTDOWN`	CXbatch is not running on the local machine.

⑤ Inter-queue priority. This priority affects which queue is looked at first for the next job to run. 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;
  ①      ②       ③        ④        ⑤       ⑥        ⑦
```

① Number of requests in this queue in a state of exiting.

② Number of requests in this queue in the state of running.

③ Number of requests in this queue in the staged state, which means the request has completed executing and is moving the stdout and stderr files to the appropriate destination directory.

④ Number of requests in this queue in a state of being queued.

⑤ Number of requests in this queue in a state of waiting.

⑥ Number of requests in this queue in a state of holding.

⑦ Number of requests in this queue in a state of arriving.

The next 11 lines show miscellaneous information about the queue.

```
      ①                ②
   Run_limit = 2; Per-user run-limit = NONE
③ Accounting: Off
④ Activity ID offset: 1
⑤ Maximum request priority: 63
⑥ Cumulative system space time = 0.000000 seconds
⑦ Cumulative user space time = 0.000000 seconds
⑧ Unrestricted access
⑨ Import directory: Yes
⑩ Checkpoint: Not available
❶ Copy open files on checkpoint: No
❷ Share policy fixed = s
```

① `Run_limit` limits the number of requests that can run in a queue at any one time. When the limit is exceeded, requests are queued until the number of jobs running is less than the limit (applicable to batch queues only).

② `Per-user run limit` limits the number of requests a user is allowed to have running in a queue at any one time. Per-user run limits apply after per-queue run limits (applicable to batch queues only).

③ `Accounting` indicates whether or not batch accounting is activated (applicable to batch queues only).

④ `Activity ID offset` is the number added by CXbatch to a job's activity identification number before the request is executed. The activity ID offset is typically an integer from 1 to 9, with 0 reserved for jobs not submitted to a batch queue. The activity ID is used for accounting purposes (applicable to batch queues only).

⑤ `Maximum request priority` is the maximum priority at which a request can be submitted to the queue.

⑥ `Cumulative system space time` for batch queues is the total amount of system time used by batch jobs since the queue was created. For pipe queues, the status indicates the total time used to route jobs.

⑦ `Cumulative user space time total` is the amount of user time used by completed batch jobs since the queue was created.

⑧ `Unrestricted access` specifies the access restrictions placed on the queue. A request submitted by the superuser is an exception to these limitations; superuser requests are always queued. Access restrictions can be:

`Unrestricted` Queue can receive any request from any submitter.

`Restricted` Queue can receive only those requests submitted by a specified group(s) or user(s).

`Pipeonly` Queue accepts requests only from a pipe queue.

⑨ `Import directory` describes whether or not the current working directory for a request is imported (mounted on the machine processing the request) before the request is executed (applicable to batch queues only). This can be:

`Yes` Queue automatically imports the current working directory for any request executing in the queue. The user can override this setting for individual requests using the `-ni` option of `qsub`.

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

`No` Queue does not allow the current working directory to be imported for requests submitted to the queue. Requests requiring imported directories are rejected when submitted.

⑩ `Checkpointable` specifies whether or not jobs are automatically checkpointed in this queue in the event CXbatch shuts down (applicable to batch queues only). This can be:

`Yes` Requests running in the queue are automatically checkpointed when CXbatch shuts down. The user can override this setting for individual requests using the `-nc` option of `qsub`. Requests submitted to this queue can also be manually checkpointed at any time by the owner of the request, a CXbatch operator, or a CXbatch manager, unless they were submitted with the `-nc` option. Requests submitted with the `-nc` option are not automatically checkpointed and cannot be manually checkpointed.

`Available` Requests submitted with the `-c` option to qsub are automatically checkpointed when CXbatch shuts down. Requests not submitted with the `-c` option are not automatically checkpointed when CXbatch shuts down. Requests can be manually checkpointed at any time by the owner of the request, a CXbatch operator, or a CXbatch

manager, unless they were submitted with the `-nc` option. Requests submitted with the `-nc` option are not automatically checkpointed and cannot be manually checkpointed.

`No` Requests are not automatically checkpointed if CXbatch shuts down and cannot be manually checkpointed.

❶ `Copy open files` defines whether or not CXbatch preserves the state of open files within a process when the request is checkpointed (applicable to batch queues only).

❷ `Share policy` describes the queue share policy (applicable to batch queues only). This can be:

`User` Charges CPU usage to the user submitting the request.

`Fixed` Charges CPU usage to a specified account.

The rest of the output displays the per-process resource limits (if any) configured for the queue.

```
① Per-process core file size limit = UNLIMITED
② Per-process data size limit = UNLIMITED
③ Per-process permanent file size limit = UNLIMITED
④ Per-process execution nice value = 0 <DEFAULT>
⑤ Per-process stack size limit = UNLIMITED
⑥ Per-process CPU time limit = 36000.0 <DEFAULT>
⑦ Per-process working set limit = UNLIMITED
```

Requests submitted to the queue must comply with these limits. Requests submitted through a pipe queue are forwarded with any defined limits for the request when sent to a batch queue.

The limit used by any process of a running request cannot exceed the maximum in force when the request was queued. Limits are assigned to a request when the request is queued. If a limit for the queue is changed after the request is queued, the queued request is not affected. However, if the previously queued request exceeds the new limit, a warning message is displayed.

If a request is submitted to a queue without limit specifications, CXbatch uses the resource limits set for the queue as default limits. When a request is submitted to a queue with limits specified, CXbatch checks request limits defined to ensure they do not exceed the queue limits.

① `Per-process core file size limit` is the maximum size a core file for a process can be. If a process attempts to create a core file exceeding this maximum size, the core file is not written.

② `Per-process data size limit` is the maximum size the data segment for a process can be.

③ `Per-process permanent file size limit` is the maximum size a file created by a process can be. If a process attempts to write to a file larger than this maximum, CXbatch sends a signal to the process. If nothing is defined in the process to handle that signal, the process is killed.

④ `Per-process execution nice value` is the maximum nice value a process can have. 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.

⑤ `Per-process stack size limit` is the maximum size a stack segment for a process can be.

⑥ `Per-process CPU time limit` it the maximum total CPU time that a process can use. If this limit is exceeded, CXbatch sends a signal to the process. If this signal is not caught or ignored, the process exits.

⑦ `Per-process working set limit` is the maximum amount of physical memory that a process can use. If this limit is reached, the job is targeted for paging.

Installing the base CXbatch system

Perform the following steps to install your CXbatch base system.

Step 1: Determine which machines will run CXbatch.

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

Step 3: Log in as the superuser.

Step 4: Assign machine identification numbers (MIDs) to all the machines that run CXbatch. To do this, enter the following command at the system prompt

```
qmapmgr
```

The qmapmgr prompt appears:

```
Mapmgr:
```

Step 5: Create the `nmap` database on the local host machine by entering

```
create
```

Step 6: Add the MID for each host to the nmap database using the `add mid` command. You must enter a separate command for each machine that will run CXbatch. The format for this command is

`add mid` *n machine_name*

where

n is a unique number identifying the machine. This should be the first name that appears in the /etc/hosts file line. This should not be an alias.

machine_name is the name of the machine that will run CXbatch.

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

Figure 2 Creating nmap database

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

Step 7: Exit qmapmgr by entering

```
exit
```

Step 8: Take a snapshot of the system using the `qsnapshot -m` command. This command saves the current CXbatch networking configuration as a series of qmapmgr commands. The information is saved to the screen by default and can be output to a file to be used to restore the CXbatch configuration. For example, to save the new configuration to a file named batch_nconfig, enter

```
qsnapshot -m > batch_nconfig
```

Figure 3 illustrates output from `qsnapshot` with the `-m` option

Figure 3 Saving the configuration database to a file

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

Step 9: Repeat Step 3 through Step 8 on each machine that will run CXbatch. The same MIDs must be entered at each machine. That is, *hostA* in the sample configuration will have MID of 1 on all machines.

Assigning managers and operators

After installing the base CXbatch system, set up CXbatch managers and operators for each machine. A CXbatch manager can change any CXbatch characteristic on the local machine. A CXbatch operator can execute operator commands, which are a subset of all the commands available with qmgr. See Chapter 1, "CXbatch processing," for a complete list of operator commands.

You can specify managers and operators by using either of the following two methods:

- Using the `add manager` command in qmgr to give users access to all commands defined by CXbatch as operator or manager commands.
- Using the `op` utility in ConvexOS to specify access to some or no commands.

Using add manager command

Perform the following steps to add managers and operators to the CXbatch system using the `add manager` command.

Step 1: Log in as the superuser.

Step 2: Start the qmgr utility by entering

```
qmgr
```

Step 3: The qmgr prompt appears:

```
Mgr:
```

Step 4: Add CXbatch managers using the `add manager` command. The format is

```
add manager user_name:x
```

where

user_name is the name of the user who is to have manager or operator access.

x specifies manager or operator access. Enter an `m` to grant manager access; enter an `o` to grant operator access.

Figure 4 illustrates use of this command to grant manager privileges to users *batchman* and operator privileges to user *batchop*.

Figure 4 Granting manager and operator privileges

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

Step 5: Repeat these steps on each host configured with CXbatch.

Using the op utility

If you want to provide access to specific commands using the `op` utility, specify in the /etc/op.access file which users are allowed access to which commands. If the /etc/op.access file does not already exist, you must create it. Refer to *Managing ConvexOS: Configuration Guide* or the op.access(5) man page for detailed information on using the `op` utility and setting up the op.access file.

For example, in your configuration you want a user named *batchman* to be able to enable, disable, and abort queues, delete requests, and shut down the batch system. You also want a user named *batchop* to enable, disable, and abort queues and delete requests. To provide this capability, enter the following lines in the /etc/op.access file:

```
enableq /usr/convex/qmgr enable queue *1;
  users=batchman, batchop
disableq /usr/convex/qmgr disable queue *1;
  users=batchman, batchop
abortq /usr/convex/qmgr abort queue *1;
  users=batchman, batchop
deletereq /usr/convex/qmgr delete request *1;
  users=batchman, batchop
shutdownbatch /usr/convex/qmgr shutdown *1;
  users=batchman
```

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

Determining additional queues

The next step in installing and configuring the CXbatch system is to determine which queues are needed to satisfy requirements of your CXbatch users. Do this for each host configured with CXbatch.

There are two types of CXbatch queues:

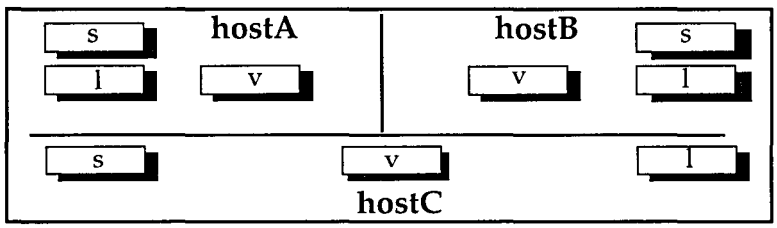
- **Batch**—Batch queues are used only to execute CXbatch batch requests. They hold requests for scheduled, perhaps delayed, processing by subsystems within CXbatch.
- **Pipe**—Pipe queues are routing queues that do not directly process requests but, instead, transmit requests to other queues on either the same or a remote machine.

When you install the base CXbatch system, three default batch queues are added named:

- **short** —Referred to as the *s* queue. It is used for short jobs that do not require much computer resource usage.
- **long**—Referred to as the *l* queue. It is used for long jobs that require moderate resource usage.
- **verylong**—Referred to as the *v* queue. It is used for long-running jobs that require an extensive amount of resource usage.

Assume a sample configuration with three host machines called *hostA*, *hostB*, and *hostC*. Each host is running ConvexOS and NFS connected to each other by Ethernet. Figure 5 illustrates this configuration with the default queues configured.

Figure 5 Default queues



You can add additional queues to all or some of your host machines, depending on needs at your site. Let's assume in the sample configuration that two pipe queues are added to *hostA* and *hostB*: one called *best* and one called *c*.

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

hostA is a CONVEX C120 used primarily by the development group. This is a general-purpose, multiuser machine. Long-running jobs should not be run on this machine because long-running jobs reduce interactive response time. The goal is to maintain reasonable interactive response time because this machine is used in software development efforts.

hostB is a C120 used primarily by the design group. This is a general-purpose, multiuser machine. Long-running jobs should not be run on this machine because long-running jobs reduce interactive response time. The goal is to maintain reasonable interactive response time because this machine is used in software design efforts.

hostC is a C210 used by both groups. This machine is designated for long-running jobs. The response time on this machine is not as important as response time on the other machines because jobs on this machine are run primarily in batch mode.

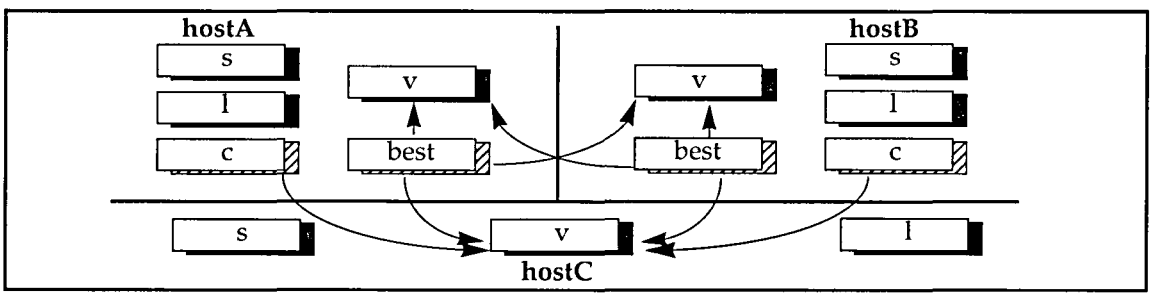
In the sample configuration, users who belong to the development group can submit long-running jobs to the *v* queue on either *hostA*, *hostB*, or *hostC* via the *best* pipe queue on *host*A. Users who belong to the design group can submit long-running jobs to the *v* queue on either *hostA*, *hostB* or *hostC* via the *best* pipe queue on *hostB*.

Small jobs can be submitted to the *s* and *l* queues on the local machine, or the *v* queue on *hostC* via the *c* pipe queue, regardless of group affiliation.

This configuration prevents one group from monopolizing the other group's resources, primarily CPU time. While the sample configuration used in this manual is not designed to cover every situation you may encounter, it covers the main issues you must consider when establishing your system.

Figure 6 illustrates this configuration. In Figure 6, squares with solid backgrounds represent the initial queues shipped with the system. Squares with striped backgrounds represent new queues that serve as pipe (routing) queues. Arrows show possible destinations of the pipe queues.

Figure 6 Sample configuration



Adding batch queues

Once you have installed CXbatch on the desired hosts, you may want to add batch queues to all or some of the hosts configured with CXbatch. Perform the following steps to do this. You must be a batch manager to perform these steps.

Step 1: If you have Share Scheduler installed on your system, be sure that you have added the queues as scheduling groups before proceeding. See the first page of this chapter for details on the correct sequence of tasks you should perform to configure the CXbatch system. See *CONVEX Share Scheduler System Manager's Guide* for details on how to add batch queues as scheduling groups.

Step 2: Decide on the additional batch queues you want on each host. Make your decisions based on such things as CPU time required, machine load, and machine use.

Step 3: Log in as a CXbatch manager on the host machine for which you are adding batch queues.

Step 4: Start the qmgr utility by entering

```
qmgr
```

The qmgr prompt appears:

```
Mgr:
```

Step 5: Create additional batch queues using the `create batch_queue` command. The format for the `create batch_queue` command is

```
create batch_queue queuename priority=priority
[share_policy user|fixed=user] [pipeonly]
[import_dir=option] [run_limit=run-limit]
```

where

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

`priority` is the inter-queue priority for the queue. This priority affects which queue is looked at first for the next job to run. *priority* can be any number from 0 to 63; 0 is the lowest priority and 63 the highest.

`share_policy` specifies where CPU usage charges are charged. If Share Scheduler is installed on your system (whether or not it is activated), you must include a Share policy setting or the command will not be processed. This can be:

fixed=*user* Charges CPU usage from this queue to the user specified as *user*. *user* can be either the user name or UID. Because resources are not charged to their own accounts, this gives users incentive to use batch queues for processing jobs.

user Charges CPU usage from this queue to the user submitting the job.

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

`import_dir` Describes whether or not the current working directory for a request is imported (mounted on the machine processing the request) before the request is executed. This can be:

`Yes` The queue automatically imports the current working directory for any request executing in the queue. The user can override this setting using the `-ni` option of `qsub`.

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

`No` The queue does not allow the current working directory to be imported for requests. Requests submitted requiring imported directories are rejected.

If you specify `Yes` or `Available` for `import_dir`, CXbatch imports directories with NFS by making temporary mount points in /tmp of the originating machine. Be aware of local automatic clean-up facilities of /tmp that might modify these NFS mount points.

`run_limit` is the maximum number of requests that can run in the queue at any given time. For example, when four requests are submitted to a queue whose run limit has been set to two, two of the requests begin running and two are queued. If you do not specify a run limit, the value defaults to one.

For example, to add a new batch queue named *b* with an inter-queue priority of 48 and a share policy of user, enter

```
create batch_queue b pr=48 share_policy user
```

Step 6: Decide on and set the per-user run limit for the queues you just created and for default queues. The per-user run limit controls the number of requests a user can have running in a queue at any one time. Use the `set per_user run_limit` command. The format is

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

where

run-limit is the number of requests that a user can run from a queue at any one time. To turn off per-user run limit, set this value to 0. Per-user limits are applied after the per-queue limits are applied. Table 2 shows how per-user run limits work.

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

Table 2 Per-user run limit

	s queue	Comments
Attributes	run limit = 4 per-user = 2	In this table, with queue attributes set as shown, the requests with stars by them begin running, while other requests are queued.
Requests	johndoe * johndoe * Root * johndoe janedoe*	

Step 7: Decide on and set the global per-user run limit for the queues you just created and for default queues. The global per-user run limit controls the number of requests a user can have running in all queues at any one time. Use the `set global per_user run_limit` command. The format is

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

where

run-limit is the number of requests that a user can have running in all queues at any one time. To turn off the global per-user run limit, set this value to 0. Building on the previous table, we can see how the global run limit works in Table 3.

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

Table 3 Global run limits

	s queue	l queue	Global
Attributes	run limit = 4 per-user = 2	run limit = 4 per-user = 2	global run limit = 3
Requests	johndoe * johndoe * johndoe Root * Root * janedoe janedoe	Root * janedoe * janedoe * janedoe Root johndoe *	

Comments
In this table, with queue attributes set as shown, the requests with the stars by them begin running, while other requests are queued.

Step 8: Decide on and set the maximum CPU time limit for a process for the queues you just created and default queues. This limit controls the amount of CPU time each process can use. When a request is submitted to a queue, CXbatch checks the request limits to ensure the CPU time limit specified for the request does not exceed the maximum CPU time limit. If it does exceed the maximum limit, the request is rejected. Use the `set per_process cpu_limit` command. The format is

```
set per_process cpu_limit = (limit) queuename
```

where

limit is the maximum time a process can run in *hours:minutes:seconds.millisecond*. Milliseconds are ignored by CONVEX machines.

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

Step 9: Decide on and set the maximum working set size for the queues you just created and for default queues. This limit controls the maximum size of a working set created by a process. When a request is submitted to a queue, CXbatch checks the request limits to ensure the working set size limit specified for the request does not exceed the maximum working set size limit. If it does exceed the maximum limit, the request is rejected. Use the `set working_set_limit` command. The format is

```
set working_set_limit = (limit) queuename
```

where

limit is the maximum size in bytes for a working set created by any process in a request in *limit* [*units*] format. *limit* can be any integer up to 8 digits. You can specify that no limit is to be applied by entering `unlimited`.

units can be any one of the following:

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

If you omit *units*, bytes is assumed.

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

Step 10: Decide on whether or not the queue has unrestricted access. If unrestricted, any user or group can submit batch requests to the queue. If restricted, only users and groups specified in the next two steps have access to the queue. Use the `set no_access` command if you want to restrict access to a queue. The format is

```
set no_access queuename
```

where *queuename* is the name of the queue for which you wish to restrict access.

Step 11: If you set the access restriction parameter to no access in Step 10, specify the users who will have access to the queue. Use the `add user` command. The format is

```
add users = user queuename
```

where

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

user can be either the user name or UID. If you specify a UID, you must enclose it in square brackets [ ].

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

Step 12: If you set the access restriction parameter to no access in Step 10, specify the groups who will have access to the queue. Use the `add group` command. The format is

```
add group = group queuename
```

where

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

group can be either the group name or GID. If you specify a GID, you must enclose it in square brackets [ ].

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

Step 13: Check that the attributes were correctly set using the `show long queue` command. The format is

```
show long queue queuename
```

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

```
show long queue b
```

Figure 7 illustrates the output for this command.

Figure 7 Viewing queue attributes

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

Step 14: Fix any attributes using the `set` commands. Refer to Chapter 6, "qmgr commands" for a complete list of these commands and their formats.

Step 15: Repeat Step 5 through Step 14 for each queue on this host.

Step 16: Exit qmgr by entering

```
exit
```

Step 17: If you configured any queues to allow files to be imported from the current working directory, edit the /etc/exports file on each host to include entries for all file systems eligible for remote mounting.

For example, if you want to specify that the /usr and /mnt file systems on *hostA* can be exported to *hostB* and *hostC*, include the following lines in the /etc/exports file on *hostA*:

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

If you are using a name server, you must specify the fully qualified hostname. See the exports(5) man page for more information on the format of this file.

Step 18: After editing the exports file, initialize the list of exportable file systems using the `exportfs` command. From the shell prompt enter

```
exportfs -a
```

Step 19: Edit the /etc/hosts file on the local machine to include any remote host names you are exporting from. If you are using TCP/IP protocol, Figure 8 illustrates an example /etc/hosts file entry on *hostB*.

Figure 8 Example /etc/hosts file entry with TCP/IP protocol

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

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

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

where

internet_address	is the official internet address for this host.
official_name	is the official name for this host, as specified with the `hostname` program.
aliases	an unofficial name or list of unofficial names for this host.
#comment	any comment you want about this host.

If you are not using TCP/IP protocol, you must include an entry in this file for the local host. Figure 9 illustrates an example /etc/hosts file entry on *hostA*.

Figure 9 Example /etc/hosts file entry without TCP/IP protocol

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

Step 20: If only certain users are to have access to file systems on a remote machine, skip to Step 21. If all users are to have access to file systems on a remote machine, edit the /etc/hosts.equiv file on the remote machine to include the names of the hosts where file systems will be imported to. This makes it possible for all users to log into the machine without further password validation as long as the user has an account on that machine. Figure 10 illustrates an example /etc/hosts.equiv file on *hostC*. Each line in this file represents one entry. Each entry represents one remote host.

Figure 10 Example /etc/hosts.equiv file

```
hostA
hostB
```

Step 21: If only certain users are to have access to a remote machine, instruct those users to edit their .rhosts file on the remote machine to include the host names of the hosts where file systems will be imported to. The .rhosts file has the same format at the /etc/hosts.equiv file described in Step 20. Refer to rhost(5) man page for more details.

Step 22: During processing, several directories in usr/spool/nqs hold job output and transaction scripts before job output is sent to the submitter's output file. If /usr/spool/nqs becomes full, jobs will fail. Consider creating a separate disk partition for /usr/spool/nqs. See *Managing ConvexOS/Secure: Configuration Guide* for details on setting up partitions.

Step 23: Start the qmgr utility by entering

```
qmgr
```

Step 24: For queues to accept jobs, they must be enabled. Enable the queues using the `enable queue` command. For example, to enable the batch queue named *b*, enter

```
enable queue b
```

Step 25: For queues to run jobs, they must be started. Start the queues using the `start queue` command. For example, to start the batch queue named *b*, enter

```
start queue b
```

Step 26: Repeat Step 23 through Step 25 for all queues on this host.

Step 27: Exit qmgr by entering

```
exit
```

Step 28: Repeat Step 1 through Step 27 for each host.

Step 29: Take a snapshot of the system using the `qsnapshot` command. This command allows you to save the current CXbatch queue configuration as a series of qmgr commands. The information is printed to the screen by default and can be output to a file to be used to restore the CXbatch configuration. For example, to save the new configuration to a file named batch_qconfig, enter

```
qsnapshot > batch_qconfig
```

Figure 11 illustrates the output from `qsnapshot` command.

Figure 11 Taking a snapshot of the system

```
# qsnapshot > batch_qconfig
SET ACC_LOGFILE /dev/null
SET AID_MASK = 1
SET DEFAULT BATCH_REQUEST PRIORITY 31
SET DEFAULT DESTINATION_RETRY TIME 72
SET DEFAULT DESTINATION_RETRY WAIT 5
SET MAIL_UID 0
SET MANAGERS root:m test:m janedoe:m johndoe:m
SET SHELL_STRATEGY LOGIN
CREATE BATCH_QUEUE short PRIORITY = 48 RUN_LIMIT = 1 IMPORT_DIR = Available
SET ACCOUNTING = Off short
SET ACTIVITY_ID_OFFSET = 0 short
SET COREFILE_LIMIT = ( UNLIMITED ) short
SET DATA_LIMIT = ( UNLIMITED ) short
SET NICE_VALUE_LIMIT = ( 0 ) short
SET PER_PROCESS CPU_LIMIT = ( 300.0 ) short
SET PER_PROCESS PERMFILE_LIMIT = ( UNLIMITED ) short
SET STACK_LIMIT = ( UNLIMITED ) short
SET WORKING_SET_LIMIT = ( 10 mb ) short
SET MAXIMUM REQUEST_PRIORITY 63 short
ADD ALIAS s short
CREATE BATCH_QUEUE long PRIORITY = 32 RUN_LIMIT = 1 IMPORT_DIR = Available
SET ACCOUNTING = On long
SET ACTIVITY_ID_OFFSET = 0 long
SET COREFILE_LIMIT = (UNLIMITED) long
SET DATA_LIMIT = (UNLIMITED) long
SET NICE_VALUE_LIMIT = (0) long
SET PER_PROCESS CPU_LIMIT = (36000.0) long
SET PER_PROCESS PERMFILE_LIMIT = (UNLIMITED) long
SET STACK_LIMIT = (UNLIMITED) long
SET WORKING_SET_LIMIT = (10 mb) long
```

Creating pipe queues

Perform the following steps to create and customize pipe queues.

Step 1: Decide on the pipe queues you want on each host.

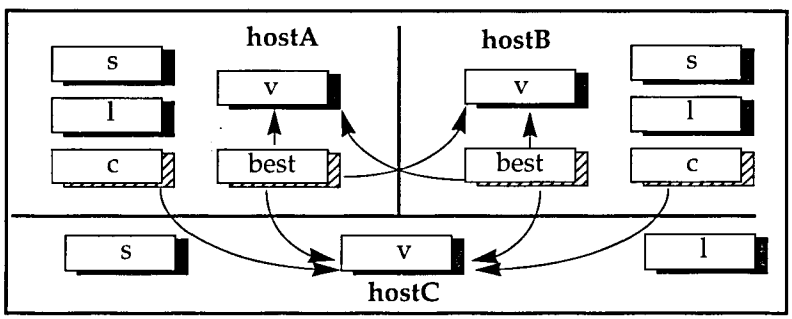
Step 2: Log in as a CXbatch manager on the host machine for which you are adding pipe queues.

Step 3: Decide if the new pipe queue will route jobs based on load balancing or simply route the request to the first destination that will accept the request. Load balancing quickly places the requests into queues without waiting until a queue is empty. This alleviates wasted CPU cycles by polling the queues. Each job placed in a queue causes the load average information for that queue to inflate. See Chapter 1, "CXbatch processing" for more details on load balancing.

Step 4: Determine the destination queue or queues where this pipe queue can route requests.

In the sample configuration, there are two pipe queues on h*ostA* and two on h*ostB*: one named *c* and one named *best*. The *c* queues on *hostA* and *hostB* route jobs to the *v* queue on *hostC*. On *hostA* and *hostB*, the *best* queue routes jobs to the *v* queues on *hostA*, *hostB*, and *hostC*. Figure 12 illustrates the complete sample configuration.

Figure 12 Sample configuration



Step 5: Start the qmgr utility by entering

```
qmgr
```

The qmgr prompt appears:

```
Mgr:
```

Step 6: Create the queues using the `create pipe_queue` command within qmgr. The format is

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

where

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

priority is the inter-queue priority for the queue. This priority affects which queue is looked at first for the next job to run. *priority* can be any number from 0 to 63; 0 is the lowest priority and 63 the highest.

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

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

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

See the pipeclient(8) man page for more details.

destination is the name or names of destination queues where this pipe queue can route requests. If you specify more than one destination, separate items in the list with commas and surround the list with parentheses. For example, to designate three destinations, enter **`(des1, des2, des3)`**.

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

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

See the create pipe_queue reference page in Chapter 6 of this document for more details.

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

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

Step 7: Decide on whether or not the queue has unrestricted access. If unrestricted, any user or group can submit batch requests to the queue. If restricted, only users and groups specified in the next two steps have access to the queue. Use the `set no_access` command if you want to restrict access to a queue. The format is

```
set no_access queuename
```

where *queuename* is the name of the queue for which you wish to restrict access.

Step 8: If you set the access restriction parameter to no access in Step 7, specify the users who will have access to the queue. Use the `add user` command. The format is

```
add users = user queuename
```

where

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

user can be either the user name or UID. If you specify a UID, you must enclose it in square brackets [ ].

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

Step 9: If you set the access restriction parameter to no access in Step 7, specify the groups who will have access to the queue. Use the `add group` command. The format is

```
add group = group queuename
```

where

group is one or more groups who have access to the queue. If you specify more than one group, you must separate items in the list with commas and surround the list with parentheses. *group* can be either the group name or GID. If you specify a GID, you must enclose it in square brackets [ ].

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

Step 10: Check that the attributes were correctly set using the `show long queue` command. The format for the `show long queue` command is

`show long queue` *queuename*

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

`show long queue best`

Figure 13 illustrates the output for this command.

Figure 13 Viewing pipe queue attributes

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

Step 11: If you want to change attributes for a queue, use the `add destination`, `set destination`, `set pipe_client`, `set priority`, or `set run_limit` commands within qmgr. Refer to Chapter 6, "qmgr commands" for a complete list of these commands and their formats.

Step 12: For queues to accept jobs, they must be enabled. Enable the queues using the `enable queue` command. For example, to enable the pipe queue named *best*, enter

`enable queue best`

Step 13: For queues to run jobs, they must be started. Start the queues using the `start queue` command. For example, to start the pipe queue named *best*, enter

`start queue best`

Step 14: Repeat Step 1 through Step 13 for each pipe queue on this host.

Step 15: Exit qmgr by entering

`exit`

Step 16: Take a snapshot of the system using the `qsnapshot` command. This command saves the current CXbatch queue configuration as a series of qmgr commands. The information is saved to the screen by default and can be output to a file to be used to restore the CXbatch configuration. For example, to save the new configuration to a file named batch_qconfig, enter

```
qsnapshot > batch_qconfig
```

Figure 14 illustrates the output from `qsnapshot` command.

Figure 14 Taking a snapshot of the system

```
# qsnapshot > batch_qconfig
SET ACC_LOGFILE /dev/null
SET AID_MASK = 1
SET DEFAULT BATCH_REQUEST PRIORITY 31
SET DEFAULT DESTINATION_RETRY TIME 72
SET DEFAULT DESTINATION_RETRY WAIT 5
SET MAIL_UID 0
SET MANAGERS root:m test:m janedoe:m johndoe:m
SET SHELL_STRATEGY LOGIN
CREATE BATCH_QUEUE short PRIORITY = 48 RUN_LIMIT = 1 IMPORT_DIR
= Available
SET ACCOUNTING = Off short
SET ACTIVITY_ID_OFFSET = 0 short
SET COREFILE_LIMIT = ( UNLIMITED ) short
SET DATA_LIMIT = ( UNLIMITED ) short
SET NICE_VALUE_LIMIT = ( 0 ) short
SET PER_PROCESS CPU_LIMIT = ( 300.0 ) short
SET PER_PROCESS PERMFILE_LIMIT = ( UNLIMITED ) short
SET STACK_LIMIT = ( UNLIMITED ) short
SET WORKING_SET_LIMIT = ( 10 mb ) short
SET MAXIMUM REQUEST_PRIORITY 63 short
ADD ALIAS s short
CREATE BATCH_QUEUE long PRIORITY = 32 RUN_LIMIT = 1 IMPORT_DIR =
Available
SET ACCOUNTING = On long
SET ACTIVITY_ID_OFFSET = 0 long
SET COREFILE_LIMIT = (UNLIMITED) long
SET DATA_LIMIT = (UNLIMITED) long
SET NICE_VALUE_LIMIT = (0) long
SET PER_PROCESS CPU_LIMIT = (36000.0) long
SET PER_PROCESS PERMFILE_LIMIT = (UNLIMITED) long
SET STACK_LIMIT = (UNLIMITED) long
SET WORKING_SET_LIMIT = (10 mb) long
```

For more information, refer to the qsnapshot(8), qmgr(8), and qmapmgr(8) man pages.

Step 17: Repeat Step 1 through Step 16 for each host.

Deleting queues

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

Step 1: Log in as a CXbatch manager on the host machine for which you are deleting queues.

Step 2: Start the qmgr utility by entering

```
qmgr
```

The qmgr prompt appears:

```
Mgr:
```

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

```
disable q queuename
```

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

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

```
stop q queuename
```

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

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

```
delete q queuename
```

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

Configuring and activating CXbatch accounting

The accounting system provided with ConvexOS tracks system resources an individual or group uses. However, it does not include information unique to CXbatch, such as job and queue priorities. If this information is important to you, you must activate the CXbatch accounting system.

The CXbatch accounting system saves information according to the structure defined in the /usr/include/batch-acct.h file. Figure 15 illustrates the CXbatch accounting structure.

Figure 15 Example batch accounting structure from /usr/include/batch-acct file

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

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

Perform the following steps to activate the CXbatch accounting system.

Step 1: Log in as a CXbatch manager on the host machine for which you are configuring accounting.

Step 2: Start the qmgr utility by entering

```
qmgr
```

The qmgr prompt appears:

```
Mgr:
```

Step 3: Set the activity ID offset for the queue.

If accounting is enabled, jobs are billed to billing activities. Valid billing activities and their unique identification numbers are assigned by the system manager when setting up the accounting system. See *Managing ConvexOS/Secure: Configuration Guide* for details on setting up valid billing activities.

When a job is submitted for execution, it is assigned a job ID. This job ID is the same as the billing activity ID. When a job is submitted to a batch queue for execution, the job ID is calculated by adding the activity ID offset to the billing activity ID.

For example, assume there are five activities to which users can bill jobs. These activities and the activity ID numbers assigned to them are:

- default (activity ID of 000)
- test (activity ID of 100)
- train (activity ID of 200)
- support (activity ID of 300)
- development (activity ID of 400)

Assume a user charges a job to the *development* activity ID (activity ID 400) and submits the job for execution to the short queue. Also assume the short queue has an activity ID offset of 1. CXbatch assigns the job a job ID of 401. From this number, you can assume the job passed through the batch system using the short queue (assuming you have not assigned any other queues the same activity ID offset).

When assigning numbers to billing activities in /etc/activities, it is important not to assign consecutive numbers because it prevents you from gathering accurate billing information. For example, assume there are five activities to which users can bill jobs. These activities and the activity ID numbers assigned to them are:

- default (activity ID of 0)
- test (activity ID of 1)
- train (activity ID of 2)
- support (activity ID of 3)
- development (activity ID of 4)

Assume a user charges the job to the *test* activity ID (activity ID 1) and submits the job for execution to the short queue. Also assume the short queue has an activity ID offset of 1. CXbatch assigns the job a job ID of 2, incorrectly charging the job to the *train* activity instead of a batch job for project *test*.

Use the `set activity_id_offset` command to set the activity ID offset. The format for this command is

`set activity_id_offset=`*offset queuename*

where

offset is the offset for this queue. This number is typically a number from 1 to 9, with 0 reserved for jobs that are not submitted to batch queues.

queuename is the name of the queue you are configuring.

For example, to set the activity ID offset to 1 for the *s* queue, enter

```
set activity_id_offset=1
```

Step 4: Using the `show long queue` command, view queue attributes to verify the change took place. Figure 16 illustrates the use of this command to check the activity ID offset.

Figure 16 Viewing queue attributes

```
Mgr: sho long q s
s@hostC; type=BATCH; [ENABLED, INACTIVE]; pri=48
 0 exit; 0 run; 0 stage; 0 queued; 0 wait; 0 hold; 0 arrive;
                        .
                        .
                        .
 Activity ID offset: 1
```

Step 5: Set the activity mask for the queue using the `set aid_mask` command. This mask reflects the spacing between activity IDs defined in /etc/activities file. For example, to set the activity mask to 100 enter

```
set aid_mask=100
```

Step 6: Specify the log file where accounting data is collected using the `set acc_logfile` command. For example, to specify a log file named /usr/adm/batchacct, enter

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

Step 7: Using the `show parameters` command, view queue parameters to verify the change took place. Figure 17 illustrates use of this command to check the accounting log file.

Figure 17 Checking the accounting log file

```
Mgr: show parameters
  .
  .
  .
Accounting log file = /usr/adm/batch-acct
```

Step 8: Activate CXbatch accounting with the `set accounting` command. You can activate and deactivate batch accounting on a queue-by-queue basis. When activated, CXbatch saves batch job information in an accounting log file (if one is specified) and, if requested with `qsub -me`, sends mail to the user detailing what resources were used. For example, to activate accounting for the *s* queue enter

```
set accounting=on s
```

Step 9: Using the `show long queue` command, view queue attributes to verify the change took place. Figure 18 illustrates the use of this command to check whether or not accounting has been turned on.

Figure 18 Viewing queue attributes

```
Mgr: sho long q v
v@hostC; type=BATCH; [DISABLED, STOPPED]; pri=48
0 exit; 0 run; 0 stage; 0 queued; 0 wait; 0 hold; 0 arrive;
                    .
                    .
                    .
Accounting: On
```

Step 10: Repeat Step 3 through Step 9 for each queue on this host.

Step 11: Exit qmgr by entering

```
exit
```

Step 12: Repeat Step 1 through Step 11 for each host.

Enabling Checkpoint Restart

Checkpoint Restart is a standard feature of ConvexOS. It saves the state of selected processes or process hierarchies to disk files so they can be restarted from the saved state. Checkpoint Restart is useful for application programs that must run for long lengths of time and that—once halted—cannot be started from the beginning without wasting significant time and resources. Such applications can be saved to files (checkpointed) either by operator intervention or by a periodically executed script that includes Checkpoint Restart commands. If the application is halted, either by a system crash, by a scheduled shutdown, or by an operator, it can be restarted as it was when it was last checkpointed. If desired, the process can be restarted under control of a debugger.

CXbatch allows users and batch administrators to use the ConvexOS Checkpoint Restart functionality when submitting batch jobs. Perform the following steps to enable Checkpoint Restart.

Step 1: Log in as a CXbatch manager on the host you wish to configure.

Step 2: Create a checkpoint directory in which CXbatch will store checkpointed files. For example, to create a directory named /usr/nqs/chkpnt enter

```
touch /usr/nqs/chkpnt
```

Step 3: Set the permissions on this directory to 755. The world must be able to read the directory in order to restart checkpointed requests.

Step 4: Start the qmgr utility by entering

```
qmgr
```

The qmgr prompt appears:

```
Mgr:
```

When a request is checkpointed, CXbatch creates checkpoint files. These files are placed in the specified checkpoint directory and are removed by CXbatch when the request completes successfully or when the request is removed. This directory must be accessible by CXbatch. CXbatch has no default checkpoint directory.

Step 5: Specify the checkpoint directory you created in Step 2 to CXbatch. Use the `set checkpoint_dir` command to assign a checkpoint directory. For example, to set the checkpoint directory to /usr/nqs/chkpnt, enter

```
set checkpoint_dir /usr/nqs/chkpnt
```

The name of the queue is automatically appended onto the checkpoint directory you specify. For example, if you set the checkpoint directory to /usr/nqs/chkpnt, the *long* queue uses /usr/nqs/chkpnt/long and the *short* queue /usr/nqs/chkpnt/short.

In the case that a checkpointed request fails to restart, the checkpoint files are not removed by CXbatch from the checkpoint directory. When that request is resubmitted, and completes successfully, CXbatch removes the files from the checkpoint directory. If that request is never manually resubmitted, you must remove the checkpoint files from the checkpoint directory.

Step 6: Enable Checkpoint Restart using the `set chkpntable` command. The format is

```
set chkpntable = value
```

where *value* can be one of the following:

`Yes` Requests running in the queue are automatically checkpointed when CXbatch shuts down. The user can override this setting for individual requests using the `-nc` option of `qsub`. Requests submitted to this queue can also be manually checkpointed at any time by the owner of the request, a CXbatch operator, or a CXbatch manager, unless they were submitted with the `-nc` option. Requests submitted with the `-nc` option are not automatically checkpointed and cannot be manually checkpointed.

`Available` Requests submitted with the `-c` option to `qsub` are automatically checkpointed when CXbatch shuts down. Requests not submitted with the `-c` option are not automatically checkpointed when CXbatch shuts down. Requests can be manually checkpointed at any time by the owner of the request, a CXbatch operator, or a CXbatch manager, unless they were submitted with the `-nc` option. Requests submitted with the `-nc` option are not automatically checkpointed and cannot be manually checkpointed.

`No` Requests are not automatically checkpointed if CXbatch shuts down and cannot be manually checkpointed.

For example, to set checkpoint to available on the *s* queue, enter

```
set chkpntable = avail s
```

Step 7: Specify whether or not open files are copied during checkpoint restart using the `set copy_open_files` command. The format is

```
set copy_open_files={yes|no} queuename
```

where

`yes`	Preserves the state of open files within a process when the request is checkpointed.
`no`	Does not preserve the state of open files within a process when the request is checkpointed
queuename	is the queue you are configuring.

Step 8: Repeat Step 6 through Step 7 for each queue you wish to configure with checkpoint restart on this host.

Step 9: Repeat Step 1 through Step 8 for each CXbatch host you wish to configure with checkpoint restart.

Establishing a shell strategy

Next you must determine which shell to use to execute request script-file commands. A shell strategy determines the command interpreter used to interpret the batch request script commands if a request is submitted to the queue without a shell interpreter specified. Perform the following steps to set the shell strategy.

Step 1: Log in as a CXbatch manager on the host you are configuring.

Step 2: Start the qmgr utility by entering

```
qmgr
```

The `qmgr` prompt appears:

```
Mgr:
```

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

```
set shell_strategy option
```

where *option* can be one of the following:

`fixed=`*shell* specifies the shell that will interpret script file commands, where *shell* can be `csh`, `ksh`, or `sh`. Specify the absolute path name. The shell must exist and be executable or this command fails.

`free` specifies that the user's login shell (as defined in the /etc/passwd file) is used to interpret commands. CXbatch 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, that shell interprets the script file commands. Otherwise, *sh* is used.

`login` specifies that the user's default login shell (as defined in the /etc/password file) is used to interpret the script file commands.

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

```
set shell_strategy free
```

Step 4: Exit qmgr by entering

```
exit
```

Step 5: Repeat these steps for each host.

Configuring error logging

When a CXbatch utility encounters an error, it writes a message directly to the user's terminal. Usually, the user can interpret these messages.

When daemons, such as nqsdaemon and netdaemon (and netserver, shepherd, pipeclient, and pipeldav), encounter problems, however, they communicate with logdaemon. logdaemon handles error logging for daemons according to their level of severity. Table 4 shows these levels and the actions taken by logdaemon for each level.

Table 4 Severity levels

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

In Table 4 column 2, "log error" means the error message is sent to the syslog daemon at the given syslog level; stdout is the standard output of nqsdaemon.

syslog handles these messages as defined in the /etc/syslog.conf file set up by the system manager. You must specify what should be done with these messages by modifying the /etc/syslog.conf file. Perform the following steps to modify the /etc/syslog.conf file.

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

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

facility.level send_message_here

where

facility is the part of the system that generates the message. The CXbatch logdaemon sends the error message to syslog with a facility parameter of LOG_BATCH. Enter `LOG__BATCH` for facility.

level describes the error level of the message. This can be any error level listed in Table 4. When

an entry is selected, errors are recorded for that entry level and all preceding entry levels in the chart. For example, if you specify `Info`, you receive error messages from `Info`, `Warn`, `Error`, and `Fatal`.

send_message_here can be one of the following:

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

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

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

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

Step 3: If you specified a path name in the *send_message_here* field of the syslog.conf file, create those files using the `touch` command. For example, in the above example, messages are sent to the /usr/adm/batchlog file. Create this file by entering

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

Step 4: Reinitialize the syslog daemon, syslogd by entering

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

Step 5: Check the /etc/rc2.d/rc.std file to be sure the following line exists.

```
$EPA $Ex /usr/etc/syslogd & echo -n ' syslogd'
```

This line automatically starts the syslogd daemon each time the system is booted.

Step 6: If the line shown in Step 5 does not exist in the /etc/rc2.d/rc.std file, add it using an editor.

Step 7: If you included debug messages for logging, decide on the level of debug messages you wish logged. Significant debug messages are sent to logdaemon only if the debug level is greater than 0. Figure 19 illustrates the level of logging if the debug level is greater than 0.

Figure 19 Logging with debug level greater than zero

```
04/20/90 12:12 CXbatch(INFO): New logfile.
04/21/90 12:12 CXbatch(INFO): Time=Wed April 21 12:12:46 CDT1990.
04/21/90 12:12 CXbatch(DEBUG): main(): Configuration loaded.
04/21/90 12:12 CXbatch(DEBUG): main(): Rebuild queue state.
04/21/90 12:12 CXbatch(DEBUG): main(): Queue state rebuilt.
04/21/90 12:12 CXbatch(DEBUG): main(): Enabling virtual timers.
04/21/90 12:12 CXbatch(DEBUG):main():Looping to read request packets.
```

If the debug level is 0, only the first two lines will appear.

Step 8: If you want to change the debug level, log in as a CXbatch manager on the host you wish to configure.

Step 9: Start the qmgr utility by entering

```
qmgr
```

The qmgr prompt appears:

```
Mgr:
```

Step 10: Set the debug level using the `set debug` command. The format is

```
set debug level
```

where *level* can be one of the following:

0 No debugging information is displayed.

1 Minimum debugging information is displayed.

2 Maximum debugging information is displayed.

Step 11: Repeat Step 1 through Step 10 for each host.

Automating the operation of queues

In some environments, it may not be desirable to allow jobs to be submitted or run from CXbatch queues during certain times of the day. By using the cron utility, you can automatically control what time of day jobs can be submitted and when they can be run. cron executes commands at specified dates and times according to the instructions found in the /usr/lib/crontab file.

Any requests in the queue when the queue is aborted are lost. Therefore, any requests you want to save must be suspended and resumed later before the queue is aborted.

Perform the following steps to automate the starting and stopping of queues.

Step 1: Log in as the superuser on the host you are configuring.

Step 2: Edit the /usr/lib/crontab file to add entries for automatically starting, stopping, and aborting queues.

Figure 20 illustrates a sample /usr/lib/crontab file that automates the availability of queues. The *v* queue on *hostA* stops at 10 a.m. and restarts at 5 p.m. on Monday through Friday. At 10:05 a.m., all jobs still running in the *v* queue are killed. Because the queue is left enabled, users can submit jobs between 10 a.m. and 5 p.m., but the jobs do not start executing until after 5 p.m.

Figure 20 Sample /usr/lib/crontab file

```
0 10 * * 1-5 /usr/convex/qmgr stop queue v
5 10 * * 1-5 /usr/convex/qmgr abort queue v 0
0 17 * * 1-5 /usr/convex/qmgr start queue v
```

Each line of the crontab file represents one activity; each field in this line is separated by spaces or tabs. The first five fields in a .crontab entry are integers that specify when the command should be performed. The format is:

minute hour day_of_month month day_of_week command

where

minute	can be any number between 0 and 59.
hour	can be any number between 0 and 23.
day_of_month	can be any number between 1 and 31.
month	can be any number between 1 and 12.
day_of_week	can be any number between 1 and 7, where 1 equals Monday, 2 equals Tuesday, and so on.

command is the command that is executed when the time element is met. A percent (%) character in this field is translated as a newline character.

Each of the first five fields can be one value or a list of values separated by commas. Use an asterisk (*) to specify all legal values. To specify an inclusive range, separate two numbers with a minus sign (-).

See the crontab(5) man page for more details on specifying commands.

Step 3: Run `tellcron` to notify the cron utility of the changes made. Enter

```
tellcron
```

Step 4: Repeat these steps for each host.

Notifying users of changes

After you have completed configuring CXbatch, notify your users of changes and recommendations about the new CXbatch system.

Remote access capabilities

When CXbatch is configured on more than one machine, a user can submit jobs to remote machines in the batch network if the user has access privileges on the remote machine. If the machines in the batch network are not listed in the /etc/hosts.equiv file and users want access to a remote machine, they must create a .rhosts file in their home directory. If this file is not created, they will not have access to the remote machine. Tell your users which method you have used to configure CXbatch and whether or not they should create an individual file in their home directory.

Miscellaneous information

You should also furnish users with the following information:

- Names and aliases of any CXbatch queue
- Default shell strategy established for the CXbatch system
- Import attribute setting of each CXbatch queue
- Assignment of batch manager and operator privileges
- Default limits of each CXbatch queue
- Hours that each CXbatch queue accepts and runs requests

Controlling operation of queues

3

There are several commands available with the qmgr utility that allow a CXbatch manager or operator to control operation of queues by

- Aborting the queue
- Purging the queue of queue requests
- Moving queue requests to another queue
- Disabling the queue
- Enabling the queue
- Starting the queue
- Stopping the queue

You must start the qmgr utility before you can use these commands. To start qmgr, log in as a CXbatch manager or operator and then enter

```
qmgr
```

The following prompt appears:

```
Mgr:
```

How to use each of these commands is described in this chapter.

Removing queue requests

You can remove queue requests from a queue in one of three ways:

- Abort the queue
- Purge the queue
- Move the queue requests to another queue

How to perform each of these actions is described in the following sections.

Aborting queues

You can use the `abort queue` command to abort all queue requests that are running. CXbatch sends a SIGTERM signal to each process running in the queue, then sends a SIGKILL signal to any process that continued to run after the SIGTERM signal. All requests aborted are deleted from the queue and all output files associated with the requests are returned to the appropriate destination. Another queue request can then run in the queue. The format is

```
abort queue queue [seconds]
```

where

queue is the name of the queue you wish to abort.

seconds is the number of seconds to wait before executing the SIGKILL signal after the SIGTERM signal is sent. If a *seconds* value is not specified, the delay is 60 seconds.

Purging queues

You can use the `purge queue` command to drop all queue requests from the queue. These queue requests are irretrievable. Running requests in the queue are allowed to complete. The format is

```
purge queue queue
```

where *queue* is the name of the queue you wish to purge.

Moving queues

You can use the `move queue` command to move all requests currently not running in the queue to another queue. Requests are moved regardless of queue limit violations, access restrictions, or attribute violations. The format is

`move queue` *queue des_queue*

where

queue is the name of the queue whose requests you wish to move.

des_queue is the destination queue where you wish the requests moved.

Disabling and enabling queues

For queues to accept jobs, they must be enabled. To prevent them from accepting jobs, they must be disabled. This section describes how to enable and disable queues.

Enabling queues

Use the `enable queue` command to allow a queue to accept jobs for processing. The format is

```
enable queue queue
```

where *queue* is the name of the queue you wish to enable.

Disabling queues

Use the `disable queue` command to prevent a queue from accepting jobs for processing. The format is

```
disable queue queue
```

where *queue* is the name of the queue you wish to enable.

Starting and stopping queues

For queues to run jobs, they must be started. To prevent queues from running jobs, they must be stopped. This section describes how to start and stop queues.

Starting queues

Use the `start queue` command to specify that a queue can run batch jobs sent to that queue. Once started, queue requests in the queue are eligible for selection. The format is

```
start queue queue
```

where *queue* is the name of the queue you wish to start.

Stopping queues

Use the `stop queue` command to specify that a queue cannot run batch jobs sent to that queue. Once stopped, requests in the queue currently running are allowed to complete. All other requests are "frozen" in the queue (not eligible for selection). New requests can still be submitted, but are "frozen" like the other requests in the queue. The format is

```
stop queue queue
```

where *queue* is the name of the queue you wish to stop.

Controlling queue requests 4

Once a batch job is submitted to a queue, it becomes a queue request. Once in a queue, the CXbatch manager or operator can:

- Delete a request
- Place a request on hold
- Take a request off hold
- Move a request
- Change a request's priority
- Suspend a request
- Take a request off suspension
- Checkpoint a request
- Restart a checkpointed request
- Force a request to run

This chapter describes how to perform each of these tasks. You must log in as a CXbatch manager or operator before performing these tasks.

Displaying status of queue requests

To perform most of the actions described in this chapter, you must know the unique identifier assigned to the queue request you are acting on. You can display this and other information on queue requests using the `qstat` command. The format for this command 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 queues and queue requests.
- `-m` Displays the date and time requests will run.
- `-u` *username* Displays only those requests belonging to the specified *username*.
- `-x` Displays additional information about queues.

See the qstat(1) man page or *CONVEX CXbatch User's Guide* for more details.

queuename is the name of the queue for which you wish status information. If you do not specify a queue, information for all queues on the requested host is displayed.

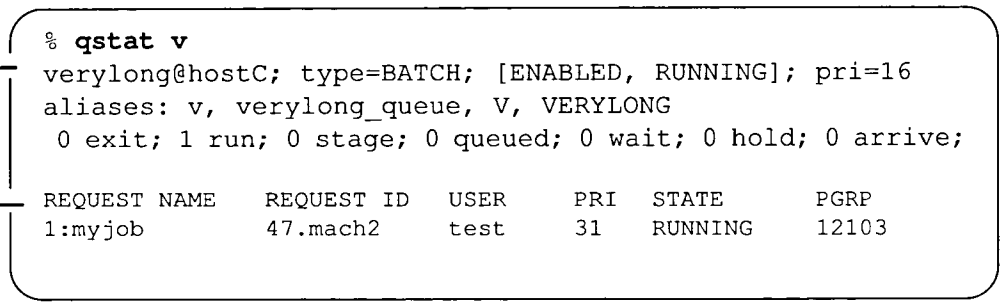
hostname is the name of the machine that receives the request. If *hostname* is omitted, the local host is assumed.

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

```
qstat v
```

Figure 21 illustrates the output for this command.

Figure 21 Standard `qstat` output



There are two sections to the standard output for the `qstat` command: information on the queues and information on each individual request in the queue. The information important to the actions described in this chapter is the information on queue requests. This information is described below:

```
REQUEST NAME    REQUEST ID   USER     PRI   STATE       PGRP
1:myjob         47.mach2     test     31    RUNNING     12103
      ①           ②          ③       ④       ⑤          ⑥
```

① Name assigned to the request.

② Unique identifier assigned to request when it is submitted to the queue.

③ User submitting the request.

④ Intra-queue priority assigned to request. This priority affects which job in a queue is executed next. This number can be from 0 to 63; 0 is the lowest priority and 63 the highest.

⑤ State of the request. This can be:

`ARRIVING`	Request is arriving at the queue.
`CHECKPOINTED`	A failed attempt was made to restart the checkpointed request. You can attempt to manually restart the request using the qrestart command. Refer to Chapter 4, "Controlling queue requests," for details on using this command.
`DEPARTING`	Request is departing from the queue but has not yet been received by the destination queue.
`EXITING`	Batch request has completed executing and will exit from the system after the required output files are returned 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 executing.
WAITING	Request is waiting for a specified amount of time to pass before attempting to execute. This could be because it was submitted with a future date and time specified for running, or because a pipe queue could not route the request and will attempt to route it later.
SUSPENDED	Request is suspended from running. It will remain suspended until manually resumed.

⑥ Process group of the request, if available to the local CXbatch daemon. This information is displayed only for processes that are running.

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

Deleting queue requests

There are two commands to delete a batch request in a queue: the qmgr utility `delete request` command and the CXbatch `qdel` command. Each of these commands are described in the following sections.

Using the delete request command

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

```
qmgr
```

The format for the `delete request` command is

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

where *request_id* is the number assigned to the request when it is submitted to CXbatch. This number can be for any request, whether or not it is executing. You can specify more than one request on the command line. If a request is running, all processes of the request are sent a SIGKILL signal. Deleted requests are removed from the queue and discarded.

Use the `qstat` command to find the *request_id*. Refer to the "Displaying status of queue requests" section in this chapter for details on using `qstat`.

Using the qdel command

The `qdel` command is a CXbatch command that is executed from a shell command line. The format is

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

where

option can be one of the following:

`-u` *username*
: Allows you to 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 allows the superuser, CXbatch manager, or CXbatch operator to delete someone else's request from a queue.

`-k`
: Sends a SIGKILL (-9) signal to an executing request. The request then exits and is deleted. If a request is running, all processes of the request are sent a SIGKILL signal.

sig
: Sends the specified signal to an executing request where *sig* can either be the signal number or signal name found in the /usr/include/signal.h file.

request_id is the number or numbers assigned to one or more requests when they are submitted to CXbatch. This number can be for any request, whether or not it is executing.

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

Use the `qstat` command to find the *request_id*. Refer to the "Displaying status of queue requests" section in this chapter for details on using `qstat`.

hostname is the name of the machine where the queue containing the request resides. If *hostname* is omitted, the local host is assumed.

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

```
qdel -u smith 291
```

Preventing queue requests from executing

It may some times be desirable to prevent a specific queue request from executing for a period of time after it has been submitted to a queue and later allowing the queue request to execute. The `hold request` and `release request` commands allow you to do this.

If a queue request is placed on hold by a CXbatch manager or operator, only a manager or operator can remove the hold.

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

```
qmgr
```

The format for each of these commands is described in the following sections.

Placing a queue request on hold

You can use the `hold request` command to place a queue request on hold preventing it from executing. The format is

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

where *request_id* is the number assigned to the request when it is submitted to CXbatch. You can specify more than one request on the command line. The request must be in the queued state to place it on hold.

Use the `qstat` command to find the *request_id*. Refer to the "Displaying status of queue requests" section in this chapter for details on using `qstat`.

Removing the hold on a queue request

You can use the `release request` command to remove the hold on a queue request, making it eligible for execution. The format is

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

where *request_id* is the number assigned to the request when it is submitted to CXbatch. You can specify more than one request on the command line. The request must be in the holding state in order to be released.

Use the `qstat` command to find the *request_id*. Refer to the "Displaying status of queue requests" section in this chapter for details on using `qstat`.

Suspending executing requests

It may some times be necessary to suspend a request that is executing and later restart it. The `suspend request` and `resume request` commands allow you to do this.

The `suspend request` and `resume request` commands are qmgr utility commands. You must start qmgr before you can use this command. To start qmgr, enter

```
qmgr
```

The format for each of these commands is described in the following sections.

Suspending an executing request

You can use the `suspend request` command to temporarily "freeze" execution of a queue request. The request must be checkpointable. This means that the queue is either set to checkpoint=available and the job was submitted with the `-c` option to `qsub`, or is set to checkpoint=yes and the job was not submitted withe the `-nc` option to `qsub`.

Only checkpointable requests can be suspended. If a request fails to checkpoint, the request continues to execute. The format is

`suspend request` *request_id* [*request_id ...*]

where *request_id* is the number assigned to the request when it is submitted to CXbatch. You can specify more than one request on the command line.

Once suspended, the request is checkpointed and execution is terminated. However, the queue request remains in the queue.

Use the `qstat` command to find the *request_id*. Refer to the "Displaying status of queue requests" section in this chapter for details on using `qstat`.

Resuming a suspended request

You can use the `resume request` command to resume execution of a suspended request. Resumed requests start in the queued state. Once the resumed request is about to enter the running state, it is restarted from its checkpointed state instead of being run in its entirety. The format is

`resume request` *request_id* [*request_id ...*]

where *request_id* is the number assigned to the request when it is submitted to CXbatch. You can specify more than one request on the command line.

Use the `qstat` command to find the *request_id*. Refer to the "Displaying status of queue requests" section in this chapter for details on using `qstat`.

Moving a request to another queue

You can use the `move request` command to move a queue request from one queue to another. The request cannot be running. The `move request` is a qmgr utility command. You must start qmgr before you can use this command. To start qmgr, enter

```
qmgr
```

The format for the `move request` command is

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

where

request_id is the number assigned to the request when it is submitted to CXbatch. You can specify more than one request on the command line. If any queue limits, access restrictions, or attributes prevent the request from being placed in the receiving queue, the queue request is not moved.

Use the `qstat` command to find the *request_id*. Refer to the "Displaying status of queue requests" section in this chapter for details on using `qstat`.

queue is the name of the queue where you wish the queue request to be moved.

Changing the queue request priority

You can use the `modify request` command to change the priority of a queue request. The request cannot be running. The `modify request` is a qmgr utility command. You must start qmgr before you can use this command. To start qmgr, enter

```
qmgr
```

The format for the `modify request` command is

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

where

`priority` = *value* specifies the new priority of the queue 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 CXbatch manager or operator can raise a request's priority.

request_id is the number assigned to the request when it is submitted to CXbatch. You can specify more than one request on the command line.

Use the `qstat` command to find the *request_id*. Refer to the "Displaying status of queue requests" section in this chapter for details on using `qstat`.

Two methods of checkpointing after request is submitted

There are two methods to checkpoint a request after it is running: using the `qchkpnt` command (a CXbatch command) and using the `chkpnt request` command (a CXbatch qmgr utility command). Each command is described in the following sections.

CXbatch uses nqsdaemon running as root to checkpoint requests. Checkpoint files are owned by the owner of the request. These conditions must be met before you can checkpoint a request:

- User requesting the checkpoint must be the owner of the request or must have CXbatch manager or operator privileges.
- Request must be running in a batch queue.
- Request must be checkpointable. This means that either the queue is set to checkpoint=available and the job was submitted with the `-c` option to `qsub`, or is set to checkpoint=yes and the job was not submitted with the `-nc` option to `qsub`.

Using the qchkpnt command

The `qchkpnt` command is a CXbatch command and can be executed from the shell command line. The format is

`qchkpnt` [*option*] *request_id* [*request_id ...*]

where

option can be one of the following:

`-e` *number unit*
: specifies the time that must pass between checkpoints, where *number* is the number of *units* that must pass and *unit* can be `minutes`, `hours`, `days`, or `weeks`. Checkpointing begins when the request begins running and ends when the request is terminated.

  If you do not specify the `-e` option, the request is checkpointed immediately. You can cancel checkpointing by specifying 0 for *number*.

`-f`
: forces a request to be checkpointed, even if conditions exist that inhibit checkpointing.

request_id is the number assigned to the request in the queue. You can specify more than one request on the command line.

By default, when a batch request is successfully submitted, CXbatch displays the identification number of the request on the terminal of the user submitting the request. You can also use the `qstat` command to find the *request_id*. Refer to the "Displaying status of queue requests" section in this chapter for details on using `qstat`.

For example, the following command checkpoints the request identified as 162 every hour:

```
qchkpnt -e 1 hour 162
```

When a request is checkpointed, an exit status describing results of the checkpoint is returned to the user's terminal submitting the checkpoint request:

- If no errors are encountered, the exit status is zero.
- If one or more of the requests are not checkpointed, the exit status is the number of requests that did not get checkpointed.
- If a fatal error occurs and none of the requests are checkpointed, the exit status is one of the codes listed in Table 5.

Table 5 Exit statuses

Exit Status	Reason
`EX_USAGE`	Syntax error
`EX_OSFILE`	Batch file system does not exist, cannot be opened, or has an error
`EX_TEMPFAIL`	Temporary failure; retry the command at a later time
`EX_NOPERM`	User does not have sufficient permission to use command

Refer to the qchkpnt(1) man page for more information.

Using the chkpnt request command

The `chkpnt` command is a qmgr utility command. You must start qmgr before you can use this command. To start qmgr, enter

```
qmgr
```

The format for the `chkpnt` command is

`chkpnt request` *request_id* `[`*request_id* `...]`

where *request_id* is the number assigned to the request when it is submitted to CXbatch. You can specify more than one request on the command line.

Use the `qstat` command to find the *request_id*. Refer to the "Displaying status of queue requests" section in this chapter for details on using `qstat`.

Restarting checkpointed requests

CXbatch automatically restarts checkpointed requests when CXbatch is restarted. A request that fails to restart remains in the checkpointed state. If a request fails to restart, you can use the `qrestart` command to manually resubmit it for execution. A restarted request has the original privileges of the request's owner.

The following conditions must be met before you can restart a request:

- The invoking user must be the owner of the request or must have CXbatch manager or operator privileges.
- The request must have been properly checkpointed.
- Request must be in a checkpointed state.

The format for this command is

`qrestart [-f]` *request_id* `[`*request_id ...*`]`

where

`-f` forces the restart. Since you do not generally need to manually restart a checkpointed request unless it has failed to restart automatically, you will usually need to use this option.

request_id is the number assigned to the request in the queue. You can restart several requests with the same command by listing multiple *request_ids*.

By default, when a batch request is successfully submitted, CXbatch displays the identification number of the request on the terminal of the user submitting the request. You can also use the `qstat` command to find the *request_id*. Refer to the "Displaying status of queue requests," section in this chapter for details on using `qstat`.

Sometimes requests cannot be restarted. These types of requests fall into two categories:

- The request fails to meet requirements for checkpointability listed at the beginning of this section.
- One of the request's processes is holding a nonrestartable request.

Refer to the qrestart(1) man page for more information.

Forcing a queue request to run

There are two methods to force a queue request to begin executing immediately: using the `qrun` command (a CXbatch command) and using the `run request` command (a CXbatch qmgr utility command). Each command is described in the following sections.

Using the qrun command

You can use the `qrun` command to force a queue request to begin executing immediately. If running the request exceeds the current run limit of the queue, the queue's run limit is increased by one until the request finishes executing. The format is

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

where *request_id* is the number assigned to the request when it is submitted to CXbatch. You can specify more than one request on the command line.

Use the `qstat` command to find the *request_id*. Refer to the "Displaying status of queue requests" section in this chapter for details on using `qstat`.

Using the run request command

You can use the `run request` command to force a queue request to begin executing immediately. If running the request exceeds the current run limit of the queue, the queue's run limit is increased by one until the request finishes executing.

The `run request` is a qmgr utility command. You must start qmgr before you can use this command. To start qmgr, enter

```
qmgr
```

The format is

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

where *request_id* is the number assigned to the request when it is submitted to CXbatch. You can specify more than one request on the command line.

Use the `qstat` command to find the *request_id*. Refer to the "Displaying status of queue requests" section in this chapter for details on using `qstat`.

Generating accounting reports

5

If you have enabled CXbatch accounting, you can use the `qsa` utility to generate reports on the accounting data. You can process the accounting records of:

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

This chapter describes how to generate CXbatch accounting reports.

To use `qsa` to process accounting information, CXbatch accounting must be enabled for each queue and an accounting log file must be defined. Refer to Chapter 2, "Configuring CXbatch," for details on how to do this.

Perform the following steps to generate CXbatch reports.

Step 1: Log in. Whether or not you have to log in as root depends on the permissions set on the accounting log file.

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

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

where

mode specifies the desired output. This can be one of the following:

`-r` Raw mode. Formats each record that matches the specified constraints and writes it to the user's stdout. Figure 22 illustrates one record from raw mode output.

Figure 22 Raw mode output

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

`-x` Extended mode. Formats each record that matches the specified constraints and writes it to user's stdout. It differs from raw mode in that it adds time spent waiting in queues, time spent executing, and time between submission and completion, and it converts all time values to ASCII. Figure 23 illustrates one record from extended mode output.

Figure 23 Extended mode output

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

`-s` Summing mode. Processes each record that matches the specified constraints and appends to stdout totals for CPU time consumed, user CPU time consumed, system CPU time consumed, and I/O operations performed. Figure 24 illustrates one record from summing mode output.

Figure 24 Summing mode output

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

`-a` Averaging mode. Processes each record that matches the specified constraints and appends to stdout averages for CPU time consumed, user CPU time consumed, system CPU time consumed, I/O operations performed, time spent waiting in queue, time spent executing, and time between submission and completion. Without the `-q` option, gives averages for every queue on your system in which accounting has been turned on. Figure 25 illustrates one record from averaging mode output.

Figure 25 Averaging mode output

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

constraints controls which records are selected for processing. You can specify records by queue, by user, or both. If none are present, `qsa` processes all records. Constraints can be one or more of the following:

`-Q` Processes records in all queues. Records are grouped by each queue that appears in the accounting file. This flag cannot be used with the `-q` flag.

`-q` *queuename*
Processes records in the specified queue where *queuename* can be one or more names of queues. Records are grouped by queues specified in one or more occurrences of this flag. This flag cannot be used with the `-Q` flag.

If no queue name is specified, `qsa` displays accounting information for all queues listed in the accounting file.

`-U` Processes records for all users where *username* can be one or more names of users. Records are grouped by each user that appears in the accounting file. This file cannot be used with the `-u` flag.

`-u` *username*
Processes records for specific users. The `-u` flag causes accounting records to be processed grouped by users specified in one or more occurrences of this flag. This flag cannot be used with the `-U` flag.

If no *username* is specified, `qsa` displays accounting information for every account in the system for which accounting has been turned on.

acct_file specifies the account file where the data is stored. If no account file is specified, CXbatch uses /usr/adm/batchacct.

For more information, refer to the qsa(8) man page.

qmgr commands

6

This chapter contains a description of each qmgr command. Each command description includes:

- Definition of command syntax
- Description of command parameters
- Examples

The first two or three characters of each word in each command are unique, and you need to enter only those characters for CXbatch to recognize the command. These accepted abbreviations are indicated in the "Format" section of each command description as uppercase letters. Commands can be entered in any combination of uppercase and lowercase characters.

ABORT QUEUE

abort all executing requests in a queue

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

Description: Aborts all executing requests in the specified queue by sending a SIGTERM signal to each process of each request running in the queue. After the time indicated in *seconds* has passed, CXbatch also sends a SIGKILL signal to all remaining processes.

CXbatch manager or operator privileges are required to use this command.

Parameters:

queuename

Name of the queue to abort.

seconds

Amount of real time CXbatch waits after sending a SIGTERM signal before sending a SIGKILL signal. If you omit *seconds*, CXbatch assumes a default value of 60 seconds.

ADD ALIAS

add an alias to a queue

Format: `ADd Alias` *alias queuename*

Description: Adds an alternative name for a queue. You can use the queue name or any alias assigned to the queue on the command line to reference a queue.

CXbatch manager privileges are required to use this command.

Parameters: *alias*

An alternate name by which a queue can be referenced. The name must be unique to all queues.

queuename

Name of the queue assigned the alias.

ADD DESTINATION

add destination queues to pipe queue destination set

Format:

`ADd DEStination = destination queuename`

Description:

Adds one or more destination queues to an existing destination set for a local pipe queue. If you specify more than one destination, you must separate items in the list with commas and surround the list with parentheses. For example, to designate three destinations, enter (*des1*, *des2*, *des3*).

Any machine name where a destination queue resides must be defined in the local system's network host table. See Chapter 2, "Configuring CXbatch," in this manual for details on how to add machine names to the local network host table.

CXbatch manager privileges are required to use this command.

Parameters:

destination

Name of the destination queue that is added. The syntax of the destination queue name must be in one of the following forms. Where shown, square brackets [ ] are required.

local_queue_name

local_queue_name@local_machine_name

remote_queue_name@remote_machine_name

remote_queue_name@[*remote_machine_mid*]

queuename

Name of the pipe queue for which you are defining destinations.

ADD GROUPS

add groups to existing access list

Format: `ADd Groups = group queuename`

Description: Adds one or more groups to the existing list of groups allowed access to a queue. If you specify more than one group, you must separate items in the list with commas and surround the list with parentheses. For example, to designate three groups, enter (*group1*, *group2*, *group3*).

The queue must be set to "no access" to add groups. See the `set no_access` command for details on how to do this.

CXbatch manager privileges are required to use this command.

Parameters:

group

Name of the group that is added to the access list. *group* can be either the group name or the numerical group ID. If the group ID is used, it must be enclosed in square brackets.

queuename

Name of the queue to which access is granted.

ADD MANAGERS

grant a user access to qmgr commands

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

Description: Grants one or more users access to qmgr commands.

CXbatch manager privileges are required to use this command.

Parameters: *username*

Name of the user that will receive the access. The syntax of *username* must be in one of the following forms. Where shown, square brackets [ ] are required.

local_account_name

[*local_account_id*]

[*remote_user_id*]@ *remote_machine_name*

[*remote_user_id*]@[*remote_machine_mid*]

option

Designates whether the user is granted manager or operator privileges. `m` grants manager access; `o` grants operator access.

ADD USERS

add users to the existing access list

Format: `ADd Users = user queuename`

Description: Adds one or more users to the existing list of users allowed access to a queue. If you specify more than one user, you must separate items in the list with commas and surround the list with parentheses. For example, to designate three users, enter (*user1*, *user2*, *user3*).

The queue must be set to "no access" to add users. See the `set no_access` command for details on how to do this.

CXbatch manager privileges are required to use this command.

Parameters: *user*

Name of the user that is added to the access list. This can be either the user name or the user ID. If the user ID is used, it must be enclosed in square brackets [ ].

queuename

Name of the queue to which access is granted.

CHKPNT REQUEST

checkpoint a running request

Format: `CHkpnt Request` *request_id* `[`*request_id ...*`]`

Description: Checkpoints one or more running requests. Checkpointing saves the state of the request in a set of checkpoint files for restart later. The request continues to run. Only running requests may be checkpointed.

CXbatch manager privileges are required to use this command.

Parameters: *request_id*

Unique identifier assigned to the request when the job was submitted to the queue. You can use the `qstat` command to find the *request_id*. Refer to Chapter 4 or the qstat(1) man page for details on using the `qstat` command.

CREATE BATCH_QUEUE

create a new CXbatch batch queue

Format:

```
CReate Batch_queue queuename PRiority=priority [PIpeonly]
[Run_limit=run-limit] [Import_dir=import-option]
[Share_policy User|Fixed=username]
```

Description:

Creates a new CXbatch batch queue. If you have installed CONVEX Share Scheduler (even if it is not running), you must first create an /etc/passwd entry for the queue. Refer to *CONVEX Share Scheduler System Manager's Guide* for details on how to create password entries for batch queues.

CXbatch manager privileges are required to use this command.

Parameters:

queuename

Name of the queue being created. This name can consist of any printable nonblank character except for the following: @, comma, equal sign, left or right parenthesis. The queue name cannot start with a digit (0-9).

`PRiority` = *priority*

Inter-queue priority for the queue. This priority affects which queue is looked at first for the next job to run. *priority* can be any number from 0 to 63; 0 is the lowest priority and 63 the highest.

`PIpeonly`

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

`Run_limit` = *run-limit*

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

`Import_dir` = *import-option*

Describes whether or not the current working directory for a request is imported (mounted on the machine processing the request) before the request is executed. This can be:

`Yes`	Queue automatically imports the current working directory for any request executing in the queue. The user can override this setting for individual requests using the `-ni` option of `qsub`.
`Available`	Allows the user to specify importation of the current working directory for any request submitted to the queue using the `-i` option of `qsub`.

No — Queue does not allow the current working directory to be imported for requests submitted to the queue. Requests requiring imported directories are rejected when submitted.

`Share_policy User|Fixed=`*use*

Specifies where the CPU usage charges are charged. If Share is installed on your system (whether or not it is activated), you must include a Share policy setting or the command will not be processed. This can be:

Fixed=*user* — Charges CPU usage from this queue to the user specified as *user*. *user* can be either the user name or UID. Because resources are not charged to their own accounts, this gives users incentive to use batch queues for processing jobs.

User — Charges CPU usage from this queue to the user submitting the job.

CREATE PIPE_QUEUE

create a new CXbatch pipe queue

Format:

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

Description:

Creates a new CXbatch pipe queue.

CXbatch manager privileges are required to use this command.

Parameters:

queuename

Name of the queue being created. This name can consist of any printable nonblank character except for the following: @, comma, equal sign, left or right parenthesis. It must not start with a digit (0-9).

`PRiority` = *priority*

Inter-queue priority for the queue. This priority affects which queue is looked at first for the next job to run. *priority* can be any number from 0 to 63; 0 is the lowest priority and 63 the highest.

`Server` = *(server)*

Name of the server that transports requests submitted to the queue to one of the destination queues. This can be either:

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

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

See the pipeclient(8) man page for more details.

`Destination` = *destination*

Name of one or more destination queues where this pipe queue can route its requests. If you specify more than one destination, you must separate items in the list with commas and surround the list with parentheses. For example, to designate three destinations, enter (*des*1, *des*2, *des*3).

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

local_queue_name

local_queue_name@local_machine_name

remote_queuename@remote_machine_name

remote_queuename@[remote_machine_mid]

`PIpeonly`

Specifies that the queue can only accept requests submitted from a pipe queue. Otherwise, the queue can accept requests from any source.

`Run_limit` = *run-limit*

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

DELETE ALIAS

delete an alias from a queue

Format: `DElete Alias` *alias*

Description: Deletes an alternate name from a queue.

CXbatch manager privileges are required to use this command.

Parameters: *alias*

Alias that is deleted from a queue.

DELETE DESTINATION

delete destination queues

Format:

`DElete DEStination =` *destination queuename*

Description:

Deletes one or more destination queues from an existing destination set for a local pipe queue. If you specify more than one destination, you must separate items in the list with commas and surround the list with parentheses. For example, to designate three destinations, enter (*des1*, *des2*, *des3*).

Any request being transferred to the deleted destination is allowed to complete successfully before the destination is deleted.

If all destinations for a pipe queue are deleted in this manner, the pipe queue is effectively stopped, although its actual status remains unchanged. The addition of a new destination for a pipe queue that has been effectively stopped in this manner immediately starts the queue running again.

CXbatch manager privileges are required to use this command.

Parameters:

destination

Name of the destination queue to delete. The syntax for the name must be in one of the following forms. Where shown, square brackets [ ] are required.

local_queue_name

local_queue_name@local_machine_name

remote_queue_name@ remote_machine_name

remote_queue_name@[remote_machine_mid]

queuename

Name of the pipe queue from which you want the destination removed.

DELETE GROUPS

delete groups from existing access list for queue

Format: `DElete Groups = group queuename`

Description: Deletes one or more groups from the existing list of groups allowed access to a queue. If you specify more than one group, you must separate items in the list with commas and surround the list with parentheses. For example, to designate three groups, enter (*group1*, *group2*, *group3*).

You can only use this command to delete groups that were added with the `add groups` command. The queue must be set to "no access" to delete groups. See the `set no_access` command for details on how to do this.

CXbatch manager privileges are required to use this command.

Parameters: *group*

Name of the group to remove from the access list. *group* can be either the group name or the numerical group ID. If the group ID is used, it must be enclosed in square brackets.

queuename

Name of the queue to which access is denied.

DELETE MANAGERS

delete manager from existing access list

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

Description: Deletes one or more managers from the CXbatch manager access list. You cannot delete the superuser account (root) from the CXbatch manager set.

CXbatch manager privileges are required to use this command.

Parameters: *username*

Name of the user that is being deleted from the access list. The syntax of *username* must be in one of the following forms. Where shown, square brackets [ ] are required.

local_account_name

[*local_account_id*]

[*remote_user_id*]@ *remote_machine_name*

[*remote_user_id*]@[*remote_machine_mid*]

option

Designates whether the user was granted manager or operator privileges. `m` removes manager access; `o` removes operator access.

DELETE QUEUE

delete the named queue

Format: `DElete Queue` *queuename*

Description: Deletes a queue. To delete a queue, no requests can be present in the queue, and the queue must be disabled. See the `disable queue` command for details on how to do this.

CXbatch manager privileges are required to use this command.

Parameters: *queuename*

Name of the queue to delete.

DELETE REQUEST

delete requests from local CXbatch machine

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

Description: Deletes one or more requests from the local machine. You can delete any request, whether or not it is executing. If the specified request is running, CXbatch sends a SIGKILL signal to all processes in the request. Deleted requests are removed from the queue and discarded.

CXbatch manager or operator privileges are required to use this command if you are deleting a request you do not own.

Parameters: *request_id*

Unique identifier assigned to the request when the job was submitted to the queue. You can use the `qstat` command to find the *request_id*. Refer to the qstat(1) man page for details on using the `qstat` command.

DELETE USERS

delete users from the existing access list

Format: `DElete Users = user queuename`

Description: Deletes one or more users from the list of users allowed access to a queue. If you specify more than one user, you must separate items in the list with commas and surround the list with parentheses. For example, to designate three users, enter (*user1*, *user2*, *user3*).

You can only use this command to delete users if they were added with the `add users` command. The queue must be set to "no access" to delete users. See the `set no_access` command for details on how to do this.

CXbatch manager privileges are required to use this command.

Parameters:

user

Name of the user to remove from the access list. *user* can be either the user name or the user ID. If the user ID is used, it must be enclosed in square brackets.

queuename

Name of the queue to which access is denied.

DISABLE QUEUE

prevent queue from accepting requests

Format: `DIsable Queue` *queuename*

Description: Prevents a queue from accepting jobs for processing.

CXbatch manager or operator privileges are required to use this command.

Parameters: *queuename*

Name of the queue to disable.

ENABLE QUEUE

enable queue to accept new requests

Format: `ENable Queue` *queuename*

Allows a queue to accept jobs for processing. If the specified queue is already enabled, no operation is performed.

CXbatch manager or operator privileges are required to use this command.

Parameters: *queuename*

Name of the queue to enable.

EXIT

exit from qmgr utility

Format: `EXit`

Description: Exits from the CXbatch qmgr utility program. You can also end the qmgr utility by sending an end-of-file character. The end-of-file character is typically **CTRL-d**.

Parameters: None

HELP

invoke the qmgr HELP facility

Format: `HElp [command]`

Description: Invokes the qmgr HELP facility and displays information about a qmgr command or topic.

Parameters: *command*

Command for which you want more information. If you do not specify a command, qmgr displays information describing what commands are available.

HOLD REQUEST

put requests on hold, preventing their execution

Format:

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

Description:

Places one or more requests on hold, preventing their execution. The request remains in the queue in a hold status until it is released with the `release request` command. Requests that are running cannot be put on hold; you can only place requests in a queued state on hold.

CXbatch manager or operator privileges are required to use this command if you are placing a request on hold you do not own.

Parameters:

request_id

Unique identifier assigned to the request when the job was submitted to the queue. You can use the `qstat` command to find the *request_id*. Refer to the qstat(1) man page for details on using the `qstat` command.

MODIFY REQUEST

modify the attributes of one or more requests

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

Description: Modifies the priority of one or more requests. Changing the priority allows jobs to be reordered in the queue so they run in a different order. You cannot modify the priority of a request that is running.

An operator or manager can raise or lower a request's priority. Users can only lower the priority of jobs they own.

Parameters: *value*

New priority of the queue request. This can be a number between 0 and 63; 0 is the lowest priority and 63 the highest.

request_id

Unique identifier assigned to the request when the job was submitted to the queue. You can use the `qstat` command to find the *request_id*. Refer to the qstat(1) man page for details on using the `qstat` command.

MOVE MY REQUEST

move your requests to another queue

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

Description: Moves one or more requests from their current queue to a different queue. If any queue limits, attributes, or access restrictions are violated, the request is not moved. You cannot move a request that is running.

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

Parameters: *request_id*

Unique identifier assigned to the request when the job was submitted to the queue. You can use the `qstat` command to find the *request_id*. Refer to the qstat(1) man page for details on using the `qstat` command.

queuename

Name of the queue where you want the queue request to be moved.

MOVE QUEUE

move all requests from one queue to another

Format: `MOVe Queue` *queuename des_queue*

Description: Moves all non-running requests in the queue to another queue. Requests are moved regardless of any queue limit, access restrictions, or attribute violations.

CXbatch manager or operator privileges are required to use this command.

Parameters: *queuename*

Name of the queue whose requests you wish to move.

des_queue

Name of the destination queue where you wish the requests moved.

MOVE REQUEST

move requests to another queue

Format:

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

Description:

Moves one or more requests that are not running to a different queue. You cannot move requests that are running. CXbatch does not check for queue limit violations, access restrictions, or attribute violations at the indicated queue before moving the request.

CXbatch manager or operator privileges are required to use this command.

Parameters:

request_id

Unique identifier assigned to the request when the job was submitted to the queue. You can use the `qstat` command to find the *request_id*. Refer to the qstat(1) man page for details on using the `qstat` command.

queuename

Name of the queue where you wish the queue request to be moved.

PURGE QUEUE

remove all requests not executing from the queue

Format: `Purge Queue` *queuename*

Description: Removes all requests that are not running from a queue. Running requests are allowed to complete. Purged requests are irretrievably lost.

CXbatch manager or operator privileges are required to use this command.

Parameters: *queuename*

Name of the queue to purge.

RELEASE REQUEST

release request(s) from hold, making them eligible for execution

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

Description: Removes the hold on one or more queue requests, making them eligible for execution. The request must be in a holding state in order to be released.

CXbatch manager or operator privileges are required to release a request you do not own. If a request is put on hold by a batch operator or manager, only a batch operator or manager can release it.

Parameters: *request_id*

Unique identifier assigned to the request when the job was submitted to the queue. You can use the `qstat` command to find the *request_id*. Refer to the qstat(1) man page for details on using the `qstat` command.

RESUME REQUEST

resumes execution of suspended request

Format: `RESume Request` *request_id* [*request_id ...*]

Description: Resumes execution of a suspended request, making the request eligible to run. When the request is about to be rerun, it starts execution from its suspended state.

CXbatch manager operator privileges are required to use this command.

Parameters: *request_id*

Unique identifier assigned to the request when the job was submitted to the queue. You can use the `qstat` command to find the *request_id*. Refer to the qstat(1) man page for details on using the `qstat` command.

RUN REQUEST

force listed requests to begin executing immediately

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

Description: Forces one or more requests to execute immediately. If running the request exceeds the run limit of the queue, the queue's run limit is increased until the request finishes executing.

CXbatch manager or operator privileges are required to use this command.

Parameters: *request_id*

Unique identifier assigned to the request when the job was submitted to the queue. You can use the `qstat` command to find the *request_id*. Refer to the qstat(1) man page for details on using the `qstat` command.

SET AID_MASK

set the activity ID mask

Format: `SEt AId_mask=`*mask-value*

Description: The `set aid_mask` command sets the activity ID mask. The activity ID of a request is calculated by taking the submitter's activity ID, subtracting the modulus of the submitter's activity ID and the activity ID mask, and adding the queue activity ID offset. The formula for calculating the activity ID of a request is as follows:

```
request_aid = submitter_aid - (submitter_aid%aid_mask+queue_aid_offset)
```

CXbatch manager privileges are required to use this command.

Parameters: *mask-value*

Activity ID mask. Usually, this mask is the same as the spacing between the activity IDs in the /etc/activities file.

SET ACC_LOGFILE

change the name of the accounting log file

Format: `SEt ACC_logfile` *logfile_name*

Description: Defines the name of the file that collects CXbatch accounting information.

CXbatch manager privileges are required to use this command.

Parameters: *logfile_name*

Name of the log file in which to collect CXbatch batch accounting information.

SET ACCOUNTING

turn accounting on or off for a batch queue

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

Description: Turns accounting on or off for a CXbatch batch queue.

CXbatch manager privileges are required to use this command.

Parameters: *queuename*

Name of the queue for which accounting is enabled or disabled.

SET ACTIVITY_ID_OFFSET

set the activity ID offset for a batch queue

Format: `SEt ACTivity_id_offset=`*offset-value queuename*

Description: Establishes the activity ID offset for a CXbatch batch queue. The queue named as a parameter of the command must already exist.

CXbatch manager privileges are required to use this command.

Parameters: *offset-value*

This is the number added to the user's activity ID to create the request ID.

queuename

Name of the queue that has the offset.

SET CHECKPOINT DIRECTORY

set the directory to hold checkpoint restart files

Format: `SEt CHEckpnt_directory` *directory_name*

Description: Defines the directory in which to store checkpoint files created by the CXbatch system. No checkpointing can take place until a checkpoint directory is specified. This directory must be readable by all users.

CXbatch manager privileges are required to use this command.

Parameters: *directory_name*

Name of the directory in which to store checkpoint files. The named directory must already exist.

SET CHKPNTABLE

set the checkpointable attribute of the queue

Format: `SEt CHKpntable=`*option queuename*

Description: Defines whether or not requests in a queue are automatically checkpointed if the CXbatch system shuts down.

CXbatch manager privileges are required to use this command.

Parameters: *option*

Can be one of the following:

`Yes`	Requests running in the queue are automatically checkpointed when CXbatch shuts down. The user can override this setting for individual requests using the `-nc` option of `qsub`. Requests submitted to this queue can also be manually checkpointed at any time by the owner of the request, a CXbatch operator, or a CXbatch manager, unless they were submitted with the `-nc` option. Requests submitted with the `-nc` option are not automatically checkpointed and cannot be manually checkpointed.
`Available`	Requests submitted with the `-c` option to `qsub` are automatically checkpointed when CXbatch shuts down. Requests not submitted with the `-c` option are not automatically checkpointed when CXbatch shuts down. Requests can be manually checkpointed at any time by the owner of the request, a CXbatch operator, or a CXbatch manager, unless they were submitted with the `-nc` option. Requests submitted with the `-nc` option are not automatically checkpointed and cannot be manually checkpointed.
`No`	Requests are not automatically checkpointed if CXbatch shuts down and cannot be manually checkpointed.

queuename

Name of the queue that you are setting.

SET COPy_open_files

sets the status of the copy_open attribute of a queue

Format: `SEt COPy_open_files {Yes|No}` *queuename*

Description: Defines whether or not the open regular files of a request are copied to and from the checkpoint directory on checkpoint and restart.

CXbatch manager privileges are required to use this command.

Parameters: *queuename*

Name of the queue to which this attribute is assigned.

SET COREFILE_LIMIT

set per-process maximum core file size limit for queue

Format:

`SEt CORefile_limit = (`*limit*`)` *queuename*

Description:

Sets the maximum size of a core file for a batch request process. If a process attempts to create a core file exceeding this maximum size, the core file is not written. CXbatch uses this limit if a request is submitted to the queue without a maximum core file size limit defined.

When a request is submitted to a queue, CXbatch checks any core file size limit defined to ensure that it does not exceed the core file size limit set for the queue, if one is configured.

The maximum size a core file is assigned to a request when the request is queued. If the limit for the queue is changed after the request is queued, the queued request is not affected. However, if the previously queued request creates a core file that exceeds the new limit, a warning message is displayed.

The core file size of any process in the running request cannot exceed the maximum in force when the request was queued.

CXbatch manager privileges are required to use this command.

Parameters:

limit

Maximum size in bytes a core file can be for any process. This can be any integer representing less than 100,000,000 bytes with an optional fractional part. The syntax for *limit* is

integer `[`*.fraction*`]` `[`*units*`]`

where *integer* and *fraction* are strings of up to eight decimal digits and *units* is one of the following:

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

If you omit *units*, bytes is assumed. You can specify no limit should be applied by using the words `unlimited` for the value of *limit*.

queuename

Name of the queue to which *limit* is assigned.

SET DATA_LIMIT

set per-process maximum data-segment size limit for queue

Format:

```
SEt DAta_limit = (limit) queuename
```

Description:

Sets the maximum size of a data segment for a batch request process. If a process attempts to create a data segment exceeding this maximum size, the request is rejected. CXbatch uses this limit if a request is submitted to the queue without a maximum data segment size limit defined.

When a request is submitted to a queue, CXbatch checks any data segment size limit defined to ensure that it does not exceed the data segment size limit set for the queue, if one is configured.

The maximum size in bytes a data segment can be is assigned to a request when the request is queued. If the limit for the queue is changed after the request is queued, the queued request is not affected. However, if the previously queued request uses a data segment that exceeds the new limit, a warning message is displayed.

The data segment size of any process in the running request cannot exceed the maximum in force when the request was queued.

CXbatch manager privileges are required to use this command.

Parameters:

limit

Maximum size in bytes a data segment can be for any process. This can be any integer representing less than 100,000,000 bytes with an optional fractional part. The syntax for *limit* is

integer [.*fraction*] [*units*]

where *integer* and *fraction* are strings of up to eight decimal digits and *units* is one of the following:

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

If you omit *units*, bytes is assumed. You can specify that no limit should be applied by using the word `unlimited` for the value of *limit*.

queuename

Name of the queue to which *limit* is assigned.

SET DEBUG

enable or disable CXbatch debugging output

Format: `SEt DEBug` *level*

Description: Defines the type of output generated when debugging.

CXbatch manager privileges are required to use this command.

Parameters: *level*

Specifies the amount of debug information to display. This can be:

`0` Displays no debugging information.

`1` Displays minimum debugging information.

`2` Displays maximum debugging information.

SET DEFAULT BATCH_REQUEST PRIORITY

set the default batch request intra-queue priority

Format: `SEt DEFault Batch_request Priority` *priority*

Description: Sets the default batch request intra-queue priority. This priority is assigned to any batch request submitted to the queue without a priority assignment. This priority determines the relative ordering of requests within a queue.

CXbatch manager privileges are required to use this command.

Parameters: *priority*

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

SET DEFAULT BATCH_REQUEST QUEUE

set the default queue used for batch requests

Format: `SEt DEFault Batch_request Queue` *queuename*

Description: Sets the default queue to be used for batch requests. This queue is used to process any batch request submitted to the queue without a queue assignment.

CXbatch manager privileges are required to use this command.

Parameters: *queuename*

Name of the queue that serves as the default CXbatch queue.

SET DEFAULT DESTINATION_RETRY TIME

set default total time allowed for resubmission of request

Format: `SEt DEFault DEStination_retry Time` *retry-time*

Description: Sets the default number of hours that a pipe queue destination can be unreachable before the request fails.

CXbatch manager privileges are required to use this command.

Parameters: *retry-time*

Amount of time in hours that can elapse while CXbatch tries to resubmit a request to a destination queue through a pipe queue. After this time is met, the request fails.

SET DEFAULT DESTINATION_RETRY WAIT

set default time to wait before resubmitting a request

Format:

```
SEt DEFault DEStination_retry Wait wait-time
```

Description:

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

When a pipe queue destination fails to accept a request because of a network failure or remote server failure, the request is not transferred and the destination is disabled for the number of minutes set by *wait-time*. At the end of this time, the destination is reenabled and retried as appropriate.

To prevent an infinite number of retries, you can use the command `set default destination_retry time`, which prevents a pipe queue destination from being endlessly retried.

CXbatch manager privileges are required to use this command.

Parameters:

wait-time

Amount of time in minutes that CXbatch waits before resubmitting a request to a destination queue through a pipe queue.

SET DESCRIPTION

change or delete the queue description

Format: `SEt DESCription = (`*description*`)` *queuename*

Description: Defines the description associated with the queue.

CXbatch manager privileges are required to use this command.

Parameters: *description*

A character string that describes the queue. This string can be up to 79 characters long. Enter the word `NULL` to delete a description.

queuename

Name of the queue whose description is being defined.

SET DESTINATION

create a new destination set for pipe queue

Format:

`SEt DESTination =` *destination queuename*

Description:

Creates a new destination set for a pipe queue. If you specify more than one destination, you must separate items in the list with commas and surround the list with parentheses. For example, to designate three destinations, enter `(`*des1*`,` *des2*`,` *des3*`)`.

All previous destinations specified for the queue are removed. Any machine name where a destination queue resides must be defined in the local system's network host table. See Chapter 2, "Configuring CXbatch," in this guide for details on how to add machine names to the local network host table.

CXbatch manager privileges are required to use this command.

Parameters:

destination

Name of the destination queue that is added. The syntax of the destination queue name must be in one of the following forms. Where shown, square brackets [ ] are required.

local_queue_name

local_queue_name@*local_machine_name*

remote_queue_name@*remote_machine_name*

remote_queue_name@[*remote_machine_mid*]

where *remote_queue_name* is the name of the destination queue and *remote_machine_mid* is the machine ID (entered as an integer) of the remote machine.

queuename

Name of the pipe queue for which you are defining destinations.

SET GLOBAL PER_USER RUN_LIMIT

set the global per-user run-limit

Format: `SEt Global Per_user Run_limit = run-limit`

Description: Controls the number of requests a single user can have running in all CXbatch queues at one time.

CXbatch manager privileges are required to use this command.

Parameters: *run-limit*

Total number of requests a user can have running at one time.

SET IMPORT_DIR

change the import attribute for a batch queue

Format: `SEt Import_dir = option queuename`

Description: Describes whether or not the current working directory for a request is imported (mounted on the machine processing the request) before the request is executed.

When importing, if a request does not originate from the same machine, CXbatch tries to access the remote directory using NFS mounts.

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

CXbatch manager privileges are required to use this command.

Parameters: *option*

Can be one of the following:

`Yes`	Queue automatically imports the current working directory for requests executing in the queue. The user can override this setting for individual requests using the `-ni` option of `qsub`.
`Available`	Allows the user to specify importation of the current working directory using the `-i` option of `qsub`.
`No`	Queue does not allow the current working directory to be imported. Requests requiring imported directories are rejected when submitted.

queuename

Name of the queue that is being set.

SET MAIL

set the account name appearing in "from:" portion of mail sent by CXbatch

Format: `SEt MAIl` *accountname*

Description: Defines the account name that appears in the "from:" portion of mail sent by CXbatch. This mail notifies users (when appropriate) of events concerning their CXbatch request. The default account name is the superuser account.

CXbatch manager privileges are required to use this command.

Parameters: *accountname*

Name of the user account that will be described as the sender for CXbatch mail. It can consist of any printable nonblank character except for the following: @, comma, equal sign, left or right parenthesis.

SET MANAGERS

set the list of authorized qmgr managers and operators

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

Description: Defines a list of authorized qmgr managers and operators. CXbatch managers have full privileges for all qmgr commands; CXbatch operators have privileges for a subset of qmgr commands. Existing authorizations, other than root, are deleted.

CXbatch manager privileges are required to use this command.

Parameters: *username*

Name of a user that is granted access. The syntax of *username* must be in one of the following forms. Where shown, square brackets [ ] are required.

local_account_name

[*local_account_id*]

[*remote_user_id*]@ *remote_machine_name*

[*remote_user_id*]@[*remote_machine_mid*]

option

Designates whether the user is granted manager or operator privileges. `m` grants manager access; `o` grants operator access.

SET MAXIMUM REQUEST_PRIORITY

set maximum priority of a request

Format: `SEt MAXimum Request_priority` *priority queuename*

Description: Defines the maximum intra-queue priority that can be assigned a request submitted to the queue. A request that specifies a priority higher than the maximum for the queue has its priority silently lowered to the queue maximum.

CXbatch manager privileges are required to use this command.

Parameters: *priority*

Maximum priority that can be assigned to requests submitted to the queue. This can be an integer between 0 and 63; 0 is the lowest priority and 63 the highest.

queuename

Name of the queue to which the maximum priority is applied.

SET NICE_VALUE_LIMIT

set per-process nice value limit for queue

Format:

`SEt NIce_value_limit =` *nice-value queuename*

Description:

Defines the maximum nice value that can be assigned to a process in the queue. This value also acts as the default nice value. This value is assigned to any request submitted to the queue without a nice value specified.

The user-specified nice value cannot be numerically less than the nice value as configured for the batch queue. When a request is submitted to a queue, CXbatch checks to be sure a nice value is not assigned that exceeds this limit. If an attempt is made to queue a batch request that asks for a smaller nice value, the request is rejected.

CXbatch manager privileges are required to use this command.

Parameters:

nice-value

Maximum nice value that can be assigned to a process in the queue. This can be an integer between -64 to 64.

queuename

Name of the queue that is assigned the nice value limit.

SET NO_ACCESS

delete all groups and users from existing access list for queue

Format: `SEt NO_Access` *queuename*

Description: Deletes all groups and all users from the access list for a queue, rendering it inaccessible by any user or group. Requests in the queue are allowed to remain.

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

Superuser privileges are not affected by this command; the superuser always has access to all enabled queues, even in the absence of the appropriate entries in the access list.

CXbatch manager privileges are required to use this command.

Parameters: *queuename*

Name of the queue that will have restricted access.

SET NO_DEFAULT BATCH_REQUEST QUEUE

indicates that no default batch queue exists

Format: `SEt NO_Default Batch_request Queue`

Description: Specifies that there is to be no default batch queue for the CXbatch network. All jobs submitted to CXbatch must have a queue specification.

CXbatch manager privileges are required to use this command.

Parameters: None

SET PER_PROCESS CPU_LIMIT

set per-process maximum CPU time limit for queue

Format:

```
SEt PER_Process Cpu_limit = (limit) queuename
```

Description:

Sets the maximum amount of CPU time a batch request process can use. CXbatch uses this limit if a request is submitted to the queue without a maximum CPU time limit defined.

When a request is submitted to a queue, CXbatch checks any CPU time limit defined to ensure that it does not exceed the CPU time limit set for the queue, if one is configured. If an attempt is made to queue a batch request that asks for a larger CPU time limit than the limit allowed by the queue, the request is rejected.

The maximum CPU time limit is assigned to a request when the request is queued. If the limit for the queue is changed after the request is queued, the queued request is not affected. However, if the previously queued request uses an amount of CPU time that exceeds the new limit, a warning message is displayed.

If any process tries to use a greater amount of CPU time than the limit set, CXbatch sends the signal SIGXCPU to the offending process. If the signal is not caught or is ignored, the process exits.

CXbatch manager privileges are required to use this command.

Parameters:

limit

Maximum CPU time allowed for any process in any request. The syntax for limit is:

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

White space may appear anywhere between the elements except around the decimal point. Milleseconds are accepted but ignored on CONVEX systems.

queuename

Name of the queue to which the limit is assigned.

SET PER_PROCESS PERMFILE_LIMIT

set per-process maximum permanent file size limit for queue

Format:

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

Description:

Sets the maximum size a permanent file can be for a batch request process. CXbatch uses this limit if a request is submitted to the queue without a maximum permanent file size limit defined.

When a request is submitted to a queue, CXbatch checks any permanent file size limit defined to ensure that it does not exceed the permanent file size limit set for the queue, if one is configured. If an attempt is made to queue a batch request that asks for a larger permanent file size limit than the limit allowed by the queue, the request is rejected.

The maximum permanent file size limit is assigned to a request when the request is queued. If the limit for the queue is changed after the request is queued, the queued request is not affected. However, if the previously queued request uses a permanent file size that exceeds the new limit, a warning message is displayed.

If any process tries to use a greater amount of CPU time than the limit set, CXbatch sends the signal SIGXCPU to the offending process. If the signal is not caught or is ignored, the process exits.

If the file size of any process created by a request becomes greater than this limit, CXbatch sends the signal SIGXFSZ to the offending process. If the signal is not caught or is ignored, the process exits.

CXbatch manager privileges are required to use this command.

Parameters:

limit

Maximum size in bytes a permanent file produced by any process can be. This can be a single integer representing less than 100,000,000 bytes with an optional fractional part. The syntax for *limit* is

integer [.*fraction*] [*units*]

where *integer* and *fraction* are strings of up to eight decimal digits and *units* is one of the following:

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

If you omit *units*, bytes is assumed. You may specify that no limit should be applied by using the word `unlimited` for the value of *limit*.

queuename

Name of the queue to which the limit is assigned.

SET PER_USER RUN_LIMIT

set the per-user run limit

Format: `SEt Per_user Run_limit = run-limit queuename`

Description: Controls the number of requests a user is allowed to have running in a queue at any one time. Per-user run limits are applied after the per queue limits are applied.

CXbatch manager privileges are required to use this command.

Parameters: *run-limit*

Total number of requests a user can have running in a queue at any one time. A *run-limit* of 0 disables the enforcement of this limit.

queuename

Name of the queue that is assigned a user run limit.

SET PIPE_CLIENT

change pipe client associated with pipe queue

Format:

`SEt PIpe_client = (`*client*`) `*queuename*

Description:

Sets the pipe client associated with a pipe queue.

CXbatch manager privileges are required to use this command.

Parameters:

client

Name of the pipe client associated with the pipe queue. You must define the full path name of the pipe client, followed by any arguments required by the program. CONVEX supplies two pipe clients: /usr/lib/nqs/pipeclient and /usr/lib/nqs/pipeldav. Refer to the *CONVEX CXbatch System Manager's Guide* or the pipeclient(8) man page for more information.

queuename

Name of the queue that is associated with the pipe client. The queue cannot be a batch queue.

SET PRIORITY

change the inter-queue priority of existing queue

Format: `SEt PRiority = priority queuename`

Description: Sets the inter-queue priority of an existing CXbatch queue. It affects which queue is looked at first for the next job to run.

CXbatch manager privileges are required to use this command.

Parameters: *priority*

Inter-queue priority assigned to the queue. This can be any number between 0 and 63; 0 is the lowest priority and 63 the highest.

queuename

Name of the queue that is assigned the priority.

SET RUN_LIMIT

change the run limit for existing queue

Format:

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

Description:

Changes the run limit of a queue. The run limit controls the maximum number of requests that can run in the queue at any given time.

CXbatch manager or operator privileges are required to use this command.

Parameters:

run-limit

Maximum number of requests that can run in the queue at any given time. If you omit *run-limit*, the value defaults to one.

queuename

Name of the queue that is assigned the run limit.

SET SHARE_POLICY FIXED

set a queue's share policy to charge a fixed uid's share group account

Format: `SEt SHAre_policy FIxed =` *username queuename*

Description: Specifies that CPU usage charges for a queue are charged to a specified username. Because resource usage is not charged to their own accounts, this gives users incentive to use batch queues for processing jobs.

CXbatch manager privileges are required to use this command.

Parameters: *username*

Name of the user account to charge CPU usage.

queuename

Name of the queue that is assigned the share policy.

SET SHARE_POLICY USER

set a queue's share policy to charge the request submitter

Format: `SEt SHAre_policy user` *queuename*

Description: Specifies that CPU usage charges for a queue are charge to the user submitting the request.

CXbatch manager privileges are required to use this command.

Parameters: *queuename*

Name of the queue whose share policy is set to `user`.

SET SHELL_STRATEGY FIXED

set a specific shell to interpret shell script commands

Format:

```
SEt SHell_strategy FIxed = (shell)
```

Defines which shell to use to execute request script-file commands if an interpreter is not specified when the script is submitted to CXbatch. A shell strategy determines the command interpreter used to interpret the batch request script commands.

CXbatch manager privileges are required to use this command.

Parameters:

shell

This can be csh, ksh, or sh. Specify the absolute path name of the shell to interpret batch request shell script commands. The shell must exist and be executable. If not, you will get the error message: TCML_EACCESS.

SET SHELL_STRATEGY FREE

set the login shell as initial shell to interpret shell script commands

Format:

```
SEt SHell_strategy FRee
```

Description:

Specifies the user's login shell (as defined in the /etc/passwd file) as the initial shell to use to interpret commands if an interpreter is not specified when the script is submitted to CXbatch. CXbatch then supplies the name of the script file to the login shell as standard input. Your login shell then reads the first line of the script file. If the first line specifies a shell, that shell interprets the script file commands. Otherwise, *sh* is used. In other words, the request is interpreted as though it were run interactively.

CXbatch manager privileges are required to use this command.

Parameters:

None

SET SHELL_STRATEGY LOGIN

set the login shell to interpret all shell script commands

Format: `SEt SHell_strategy Login`

Description: Sets the user's default login shell as the shell to interpret the batch request shell script commands if an interpreter is not specified when the script is submitted to CXbatch. This default login shell is defined in the /etc/passwd file.

CXbatch manager privileges are required to use this command.

Parameters: None

SET STACK_LIMIT

set per-process maximum stack size limit for queue

Format:

`SEt STack_limit = (`*limit*`)` *queuename*

Description:

Sets the maximum size of a stack segment for a batch request process. CXbatch uses this limit if a request is submitted to the queue without a maximum stack segment size limit defined.

When a request is submitted to a queue, CXbatch checks any stack segment size limit defined to ensure that it does not exceed the stack segment size limit set for the queue, if one is configured. If an attempt is made to queue a batch request that asks for a larger stack segment size limit than the limit allowed by the queue, the request is rejected.

The maximum stack segment size limit is assigned to a request when the request is queued. If the limit for the queue is changed after the request is queued, the queued request is not affected. However, if the previously queued request uses a stack segment size that exceeds the new limit, a warning message is displayed.

The stack size of any process created by the request cannot exceed the maximum in force when the request was queued.

CXbatch manager privileges are required to use this command.

Parameters:

limit

Maximum size in bytes a permanent file produced by any process can be. This can be a single integer representing less than 100,000,000 bytes with an optional fractional part. The syntax for *limit* is

integer `[` *.fraction* `]` `[` *units* `]`

where *integer* and *fraction* are strings of up to eight decimal digits and *units* is one of the following:

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

If you omit *units*, bytes is assumed. You may specify that no limit should be applied by using the words `unlimited` for the value of *limit*.

queuename

Name of the queue to which the limit is assigned.

SET UNRESTRICTED_ACCESS

give all users access to queue

Format: `SEt Unrestricted_access` *queuename*

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

CXbatch manager privileges are required to use this command.

Parameters: *queuename*

Name of the queue made accessible to all users and groups.

SET WORKING_SET_LIMIT

set per-process maximum working-set size limit for queue

Format:

`SEt Working_set_limit =` (*limit*) *queuename*

Description:

Sets the maximum size of a working set for a batch request process. CXbatch uses this limit if a request is submitted to the queue without a maximum working set size limit defined.

When a request is submitted to a queue, CXbatch checks any working set size limit defined to ensure that it does not exceed the working set size limit set for the queue, if one is configured. If an attempt is made to queue a batch request that asks for a larger working set size limit than the limit allowed by the queue, the request is rejected.

The maximum working set size limit is assigned to a request when the request is queued. If the limit for the queue is changed after the request is queued, the queued request is not affected. However, if the previously queued request uses a working set size that exceeds the new limit, a warning message is displayed.

When physical memory becomes scarce, the system prefers to take physical memory away from those processes exceeding their working-set limit.

CXbatch manager privileges are required to use this command.

Parameters:

limit

Maximum size in bytes a permanent file produced by any process can be. This can be a single integer representing less than 100,000,000 bytes with an optional fractional part. The syntax for *limit* is

integer `[` *.fraction* `]` `[` *units* `]`

where *integer* and *fraction* are strings of up to eight decimal digits and *units* is one of the following:

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

If you omit *units*, bytes is assumed. You may specify that no limit should be applied by using the word `unlimited` for the value of *limit*.

queuename

Name of the queue to which the limit is assigned.

SHOW

display information about CXbatch system

Format: `SHOW` *option*

Description: Provides information about the status of CXbatch, resource limits, queues, managers, or parameters.

Parameters: *option*

Specifies the desired operation. This can be one of the following:

ALL	Displays the standard information concerning limits supported, managers, parameters, and queues.
LIMITS_SUPPORTED	Displays the resource limits supported on the local machine.
LONG QUEUE	Displays the status of CXbatch queues on the local host in an expanded format.
MANAGERS	Displays the list of authorized CXbatch managers.
PARAMETERS	Displays the general CXbatch settings.
QUEUE	Displays the status of CXbatch queues on the local host in a short format.

The following pages describe each option in detail.

SHOW ALL

display standard information about CXbatch system

Format: `SHOw All`

Description: Displays standard information about the status of CXbatch resource limits, queues, managers, and settings.

Parameters: None

Following is an example of the output from this command:

```
Queues:

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

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

Managers:
 root:m
 batchop:m
 leader:m

CXbatch operating parameters:
 Debug level =  0
 Default batch_request priority = 31
 Default batch_request queue = long
 Default destination_retry time = 72 hours
 Default destination_retry wait = 5 minutes
 Global per-user runlimit = NONE
 Checkpoint directory = /dir1/cxbatch/qrun/user/user.chkpnt
 Accounting log file = /dev/null
 Activity ID mask = None
 Mail account = root
 Next available sequence number = 201
 Batch request shell choice strategy = FREE

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

SHOW LIMITS_SUPPORTED

display process and request limits enforced on local machine

Format: `SHOw LImits_supported`

Description: Displays the process and request limits enforced on the local machine. If a limit is not supported on the local machine and you include the limit with the `qsub` command, it is ignored.

Parameters: None

Following is an example of the output from this command:

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

SHOW LONG QUEUE

display queue status information in long format

Format: `SHOw LOng Queue [queuename] [username]`

Description: Displays status information about CXbatch queues in a long format.

Parameters:

queuename

Name of the queue for which the status is displayed. If no queue name is specified, CXbatch displays status information for all CXbatch queues. If a queue name is specified, CXbatch displays status information for the named queue only. The display orders the requests within the queue.

username

Name of the user whose request status is displayed. If you omit *username*, the status of each request in the queue is displayed.

Following is an example of the output from this command:

```
Queue for very long jobs.
verylong@mach1;  type=BATCH;  [ENABLED, INACTIVE];  pri=16 aliases: v,
verylong_queue, V, VERYLONG
 0 exit; 0 run; 0 stage; 0 queued; 0 wait; 0 hold; 0 arrive;
 Run_limit = 1;
 Accounting: Off
 Activity ID offset: 0
 Add: maximum request priority: 63
 Cumulative system space time = 1703.580000 seconds
 Cumulative user space time = 2020.910000 seconds
 Unrestricted access
 Import directory: Yes
 Checkpoint : Not Available
 Share policy fixed = Verylong
 Per-process core file size limit = UNLIMITED
 Per-process data size limit = UNLIMITED
 Per-process permanent file size limit = UNLIMITED
 Per-process execution nice value = 4
 Per-process stack size limit = UNLIMITED
 Per-process CPU time limit = UNLIMITED
 Per-process working set limit = UNLIMITED
```

SHOW MANAGERS

display the list of authorized CXbatch managers

Format: `SHOw Managers`

Description: Displays the list of authorized CXbatch managers and operators.

Parameters: None

Following is an example of the output from this command:

```
root:m
batchop:o
leader:m
```

SHOW PARAMETERS

display current values for CXbatch operating parameters

Format: `SHOw Parameters`

Description: Displays the current values for the general CXbatch operating settings.

Parameters: None

Following is an example of the output from this command:

```
Debug level = 5
Default batch_request priority = 31
Default batch_request queue = long
Default destination_retry time = 72 hours
Default destination_retry wait = 5 minutes
Global per-user runlimit = NONE
Checkpoint directory = /mach1/cxbatch/qrun/user.chk-
pnt
Accounting log file = /dev/null
Activity ID mask = None
Mail account = root
Next available sequence number = 201
Batch request shell choice strategy = FREE
```

SHOW QUEUE

display queue status information in short format

Format: `SHOw Queue [queuename] [username]`

Description: Displays status information about CXbatch queues in a short format.

Parameters:

queuename

Name of the queue for which status is displayed. If no queue name is specified, CXbatch displays status information for all CXbatch queues. If a queue name is specified, CXbatch displays status information for the named queue only. The display orders the requests within the queue.

username

Name of the user whose request status is displayed. If you omit *username*, the status of each request in the queue is displayed.

Following is an example of the output from this command:

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

SHUTDOWN

shut down CXbatch on the local host

Format: `SHUtdown [seconds] [force]`

Description:

Shuts down CXbatch on the local host. CXbatch sends a SIGTERM signal to each process of each request that is running. Unlike `abort queue`, `shutdown` requeues all of the requests it kills. All requests terminated as a result of the `shutdown` command are requeued for later execution. Requests that are successfully checkpointed are restarted from their checkpointed state when CXbatch is restarted.

After the time indicated in *seconds* has passed, CXbatch also sends a SIGKILL signal to all remaining processes for each request that is running in the named queue.

CXbatch manager or operator privileges are required to use this command.

Parameters:

seconds

Amount of real time that CXbatch waits before sending a SIGKILL signal to all remaining processes for each request. The SIGKILL signal is sent only to those processes that have not changed process groups. If you omit *seconds*, CXbatch assumes a default value of 60 seconds.

force

Forces CXbatch to shutdown, regardless of any checkpoint erros incurred during the shutdown.

START CXBATCH

start CXbatch

Format: `STArt CXbatch`

Description: Starts CXbatch from on the local machine. You cannot start CXbatch using this command the first time you install it.

CXbatch manager or operator privileges are required to use this command.

Parameters: None

START QUEUE

start the queue

Format: `STArt Queue` *queuename*

Description: Starts a queue making requests in the queue eligible for execution. If the queue is already started, you get a transaction completion message; no jobs are aborted.

CXbatch manager or operator privileges are required to use this command.

Parameters: *queuename*

Name of the queue to start.

STOP QUEUE

stop the queue

Format: `STOp Queue` *queuename*

Description: Stops a queue preventing requests in the queue from running. They remain in the queue and will be run when the queue is restarted. Any requests in the queue that are currently running are allowed to complete execution.

A queue that has been stopped can still accept new requests. However, no requests that reside within the named queue are run until the queue has been explicitly started again by the `start queue` command.

CXbatch manager or operator privileges are required to use this command.

Parameters: *queuename*

Name of the queue to stop.

SUSPEND REQUEST

suspend a specified request

Format: `SUSpend Request` *request_id* [*request_id ...*]

Description: Temporarily freezes the execution of one or more requests. The requests are checkpointed, then stopped from executing.

CXbatch manager or operator privileges are required to use this command.

Parameters: *request_id*

Unique identifier assigned to the request when the job was submitted to the queue. You can use the `qstat` command to find the *request_id*. Refer to the qstat(1) man page for details on using the `qstat` command.

qmapmgr commands

7

This chapter contains a description of each qmapmgr command. You must be in the qmapmgr utility in order to use these commands. Each command description includes

- Definition of command syntax.
- Description of command parameters.
- Examples.

The first two characters of each word in each command are unique, and you need to enter only those characters for CXbatch to recognize the command. These accepted abbreviations are indicated in the "Format" section of each command description as uppercase characters. Commands may be entered in any combination of uppercase and lowercase characters.

ADD MID

add a machine alias to the database

Format: `Add Mid` *mid principal-name*

Description: Adds an new machine identification to the CXbatch database.

Superuser privileges are required to use this command.

Parameters: *mid*

Machine identification (MID) number of the machine being added to the CXbatch database.

principle-name
Name of the machine that is added to the network. The name must be unique and must correspond to an entry in the /etc/hosts file.

Examples: `Mapmgr:` **`add mid 99 host1`**

This command adds the machine named host1 to the CXbatch database and assigns it an mid of 99. Following is an example of the output from this command:

```
NMAP_SUCCESS: Successful completion.
```

ADD NAME

add a machine alias to the database

Format: `Add Name` *alias mid*

Description: Adds an machine alias to the CXbatch database. You can use the machine name or any alias assigned to the machine on the command line for any command that requires a machine name.

Superuser privileges are required to use this command.

Parameters: alias

An alternate name by which this machine is identified. This alias is only known to the local CXbatch host and is not recognized across machines.

mid
The machine identification (MID) number of the machine receiving the alias. The MID must already exist in the CXbatch database.

Examples: `Mapmgr:` **`add name test 99`**

This command adds the alias *test* to the machine with the MID of 99 in the CXbatch database. Following is an example of the output from this command:

```
NMAP_SUCCESS: Successful completion.
```

CHANGE NAME

change a machine name associated with MID

Format:

`CHange Name` *mid new-name*

Description:

Changes the name of a machine associated with a machine identification (MID) number in the CXbatch network.

Superuser privileges are required to use this command.

Parameters:

mid

The MID of the machine whose name is changed. The MID must already exist in the CXbatch database.

new-name

New machine name assigned to the MID. The new name must be unique and must correspond to an entry in the /etc/hosts file.

Examples:

`Mapmgr:` **`change name 99`**

This command changes the name of the machine with MID 99 to *master* in the CXbatch database. Following is an example of the output from this command:

```
NMAP_SUCCESS: Successful completion.
```

CREATE

create a CXbatch database

Format: `CReate`

Description: Creates a new CXbatch database.
Superuser privileges are required to use this command.

Parameters: None

DELETE MID

delete a machine from the database

Format: `Delete Mid` *mid*

Description: Deletes a machine from the CXbatch database.

Superuser privileges are required to use this command.

Parameters: *mid*

Machine identification (MID) number of the machine to delete.

Examples: `Mapmgr:` **`delete mid 99`**

This command deletes the machine with MID 99 from the CXbatch database. Following is an example of the output from this command:

```
NMAP_SUCCESS: Successful completion.
```

DELETE NAME

delete a machine alias from the database

Format: `Delete Name` *alias*

Description: Deletes an alternate name used to reference a machine. Superuser privileges are required to use this command.

Parameters: *alias*
Alternate name to delete.

Examples: `Mapmgr:` **`delete name test`**

This command deletes the alias test in the CXbatch database:

```
NMAP_SUCCESS: Successful completion.
```

EXIT

exit from `qmapmgr` utility

Format: `Exit`

Description: Exits from the CXbatch qmapmgr utility. You can also exit qmapmgr by entering the **`quit`** command or by pressing **CTRL-d**.

Parameters: None

GET MID

display the MID that matches machine name

Format: `Get Mid` *principal-name*

Description: Displays the machine identification (MID) number for a machine.

Parameters: *principal-name*

Name of the machine whose MID is requested. This name can be either the actual machine name or an alias assigned to that machine.

Examples: `Mapmgr:` **`get mid master`**

This command displays the MID of the machine named master in the CXbatch database. Following is an example of the output from this command:

```
NMAP_SUCCESS: Successful completion.
Mid = 99.
```

GET NAME

display the machine name that matches MID

Format: `Get Name` *mid*

Description: Displays the machine name for a machine identification (MID) number.

Parameters: *mid*

The MID of the machine whose name is requested.

Examples: `Mapmgr:` **`get name 99`**

This command displays the machine name with MID 99 in the CXbatch database. Following is an example of the output from this command:

```
Name = master.
```

HELP

invoke the HELP facility

Format: `Help`

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

Parameters: None

Examples: `Mapmgr:` **`help`**

This command displays all available qmapmgr commands. Following is an example of the output from this command:

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

QUIT

exit from qmapmgr utility

Format: `Quit`

Description: Exits from the CXbatch qmapmgr utility. You can also exit qmapmgr by entering the **`exit`** command or by pressing **CTRL-d**.

Parameters: None

SHOW

display all entries in the database

Format: `SHow`

Description: Displays all entries in the CXbatch database. It lists the machine identification (MID) numbers with their corresponding machine names and aliases in MID order.

Parameters: None

Examples: `Mapmgr:` **`show`**

This command displays all MIDs in the CXbatch database and corresponding machine names and aliases. Following is an example of the output from this command:

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

CXbatch run-time directory hierarchy

A

This appendix describes the CXbatch runtime directory hierarchy. The chart on the following two pages provides information on the structure of the CXbatch directories, what the directories and files contain, and what they are used for.

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

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

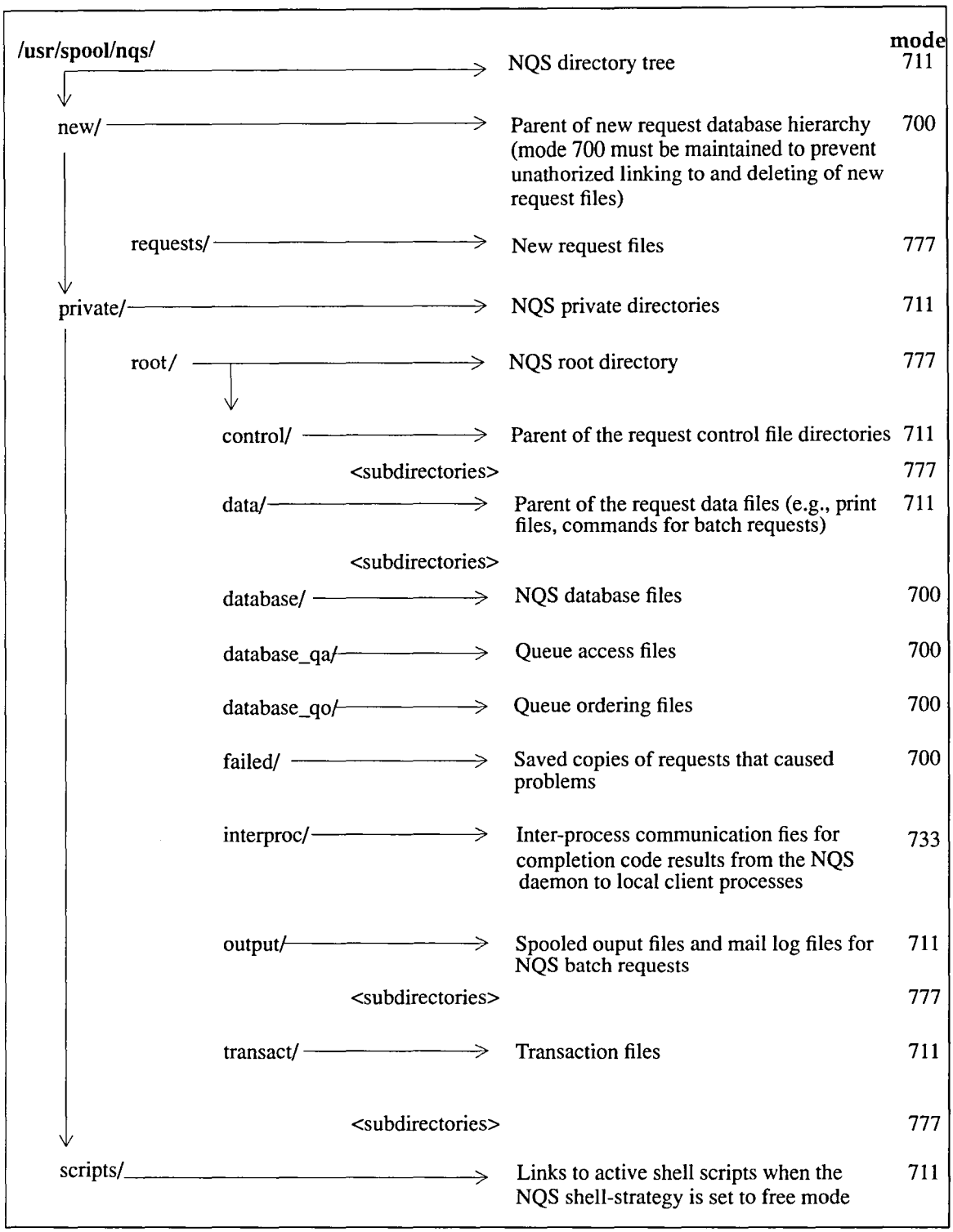
For more information on the contents or access mode of any of these files, please contact the CONVEX Technical Assitance Center (TAC).

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

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

Look in the directory named failed for output from jobs whose resource limits disallow putting output where requested, or from jobs that failed. You must be logged in as root to remove files from the directory named failed.

Figure 26 The CXbatch runtime hierarchy



Man pages

B

This appendix contains the CONVEX CXbatch man pages presented in alphabetical order. The man pages included are:

- batch-acct(5)
- nqsdaemon(8)
- pipeclient(8)
- qchkpnt(1)
- qdel(1)
- qjlist(1)
- qlimit(1)
- qmapmgr(8)
- qmgr(8)
- qps(1)
- qrestart(1)
- qrun(8)
- qsa(8)
- qsnapshot(8)
- qstat(1)
- qsub(1)
- qwatch(8)

NAME

batch-acct – CXbatch accounting file

DESCRIPTION

Batch accounting can be performed by CXbatch on a per queue basis. Batch accounting is enabled with the *qmgr* command *set accounting*. The accounting log file is specified with *set acc_logfile*. By default, accounting is off and the log file is */dev/null*.

CONVEX provides qsa(8) for processing and reporting the batch accounting data. The log file is in binary format, and its structure is described in the C include file */usr/include/batch-acct.h*.

FILES

/usr/include/batch-acct.h

SEE ALSO

qmgr(8), qsa(8)

NAME

nqsdaemon, netdaemon, logdaemon – CXbatch daemons

SYNOPSIS

/usr/lib/nqs/nqsdaemon

DESCRIPTION

nqsdaemon, *netdaemon*, and *logdaemon* are CXbatch daemons that are normally started at boot time from the */etc/rc.local* file. *nqsdaemon* automatically starts *netdaemon* and *logdaemon*.

nqsdaemon handles all local transactions, including submits, deletes, job scheduling, and system configuration. *netdaemon* handles all possibly remote transactions, including submits and file staging. *nqsdaemon* and *netdaemon* contact *logdaemon* when they need to print an error message. *logdaemon* sends the message to syslogd and notifies the batch managers if the error is fatal. See *qmgr*(8) for more information on defining CXbatch managers.

SEE ALSO

qdel(1), qjlist(1), qlimit(1), qstat(1), qsub(1), pipeclient(8), qmapmgr(8), qmgr(8), syslogd(8).

FILES

/usr/lib/nqs – directory containing CXbatch daemons
/etc/nmap – directory that contains network database

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

pipeclient, pipeldav – CXbatch pipe clients

SYNOPSIS

/usr/lib/nqs/pipeclient
/usr/lib/nqs/pipeldav [**-w** *weight*] [*host scale* ...]

DESCRIPTION

CXbatch supports pipe queues that are responsible for routing and delivering requests to other (possibly remote) queue destinations. With each pipe queue, there is an associated pipe client that is spawned to handle each request released from the queue for routing and delivery. When a pipe client is spawned, it is given complete freedom to route the request to any of the destinations in its destination set. CONVEX supplies two pipe clients, *pipeclient* and *pipeldav*, that use different methods for determining which destination should get the request.

The standard pipe client, *pipeclient*, routes the request to the first destination that will accept the request. Destinations may reject the request due to queue limit violations, lack of account authorization, and many other reasons.

pipeldav sorts the destination list by load factor, and tries destinations with low load factors first. The load factor is calculated as follows:

load factor = (*load avg* + *queue length* * *weight*) / *scale*

The *load average* is the current system load average on the destination machine. The *queue length* is the number of requests in the destination queue. The *weight* and *scale* are specified on the *pipeldav* command line and are the job weight and host scale factor.

See *qmgr*(8) for more information on configuring pipe queues.

RESTRICTIONS

The pipe client *pipeldav* gets load average information from *rstatd*. As a result, *pipeldav* is limited to load balancing over machines running *rstatd*.

SEE ALSO

qmgr(8)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qchkpnt – checkpoint CXbatch request(s).

SYNOPSIS

qchkpnt [**–e** *freq*] [**-f**] *request-id* ...

DESCRIPTION

qchkpnt checkpoints the requests whose *request-ids* are listed on the command line. The current state of all the processes comprising the request are saved in a set of checkpoint files. The checkpoint files are stored in the CXbatch *checkpoint directory*. A successfully checkpointed request will be restarted from its checkpointed state instead of being re-run.

Only running batch requests may be checkpointed. To checkpoint a request, the invoking user must be the owner of the request or have CXbatch operator privileges.

The options have the following meanings:

–e *freq* Normally the request is checkpointed immediately. When the **-e** option is present the request will be checkpointed periodically at intervals of *freq*.

The *freq* is specified as <[*number*] *unit*>, where *number* is a positive integer and *unit* is [*Hours* | *Days* | *Weeks*]. A *number* of zero will cancel periodic checkpointing.

-f Force checkpoint even if one of the processes of the request hold non-checkpointable resources.

DIAGNOSTICS

qchkpnt returns an exit status describing what it did. If there were no errors, the exit status is zero. If one or more of the requests weren't checkpointed, the exit status is the number of requests that weren't checkpointed. If a fatal error occurs and none of the requests are checkpointed (e.g., a syntax error), the exit status is one of the codes defined in *<sysexits.h>*.

EX_USAGE The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.

EX_OSFILE Some batch system file does not exist, cannot be opened, or has an error.

EX_TEMPFAIL Temporary failure; retry the command at a later time.

EX_NOPERM You did not have sufficient permission to perform the operation.

EX_SOFTWARE Too many request-ids were specified.

SEE ALSO

qsub(1), qrestart(1), chkpnt(1), restart(1), qmgr(8)

NOTES

CXbatch is an optional product; for more information, please contact your CONVEX sales representative.

NAME

qdel – delete or signal CXbatch request(s).

SYNOPSIS

qdel [**–k**] [*–signo*] [**–u** *username*] *request-id*[*@ host*] **...**

DESCRIPTION

qdel deletes all queued CXbatch requests whose *request-ids* are listed on the command line. Additionally, if the flag **-k** is specified, the default signal of **SIGKILL** (9) is sent to any running request whose *request-id* is listed on the command line. This causes the receiving request to exit and be deleted. If the flag *–signo* is present, the specified signal is sent instead of the **SIGKILL** signal to any running request whose *request-id* is listed on the command line. *signo* can be either the signal number or the signal name. The signal names are as given in */usr/include/signal.h*, stripped of the common SIG prefix. In the absence of the **-k** and *–signo* flags, *qdel* will not delete a *running* CXbatch request.

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

A CXbatch request is always uniquely identified by its *request-id*, no matter where it is in the network of the machines. A *request-id* is always of the form *seqno* or *seqno.hostname*, where *hostname* identifies the machine from whence the request was originally submitted, and *seqno* identifies the sequence number assigned to the request on the originating host. If the *hostname* portion of a *request-id* is omitted, the local host is always assumed. The local host is searched for each given *request-id*, unless a different host is specified with *@ host*.

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

DIAGNOSTICS

qdel returns an exit status describing what it did. If there were no errors, the exit status is zero. If one or more of the requests were not deleted, the exit status is the number of requests that weren't deleted. If a fatal error occurs and none of the requests are deleted (e.g., a syntax error), the exit status is one of the codes defined in *<sysexits.h>*.

EX_USAGE	The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.
EX_NOUSER	The user specified with *-u* did not exist.
EX_NOHOST	The host specified did not exist.
EX_OSFILE	Some batch system file does not exist, cannot be opened, or has an error.
EX_TEMPFAIL	Temporary failure; retry the request at a later time.
EX_NOPERM	You did not have sufficient permission to perform the operation.
EX_SOFTWARE	Too many request-ids were specified.

CAVEATS

When a CXbatch request is signaled by the methods discussed above, the proper signal is sent to *all* processes comprising the CXbatch *request* that are in the same *process group*. Whenever a CXbatch request is spawned, a new *process group* is established for all processes in the request. However, should one or more processes of the request successfully execute a *setpgrp*() system call,

such processes will *not* receive any signals sent by the *qdel*(1) command. This can lead to "rogue" request processes that must be killed by other means such as the *kill*(1) command.

SEE ALSO

qjlist(1), qlimit(1), qstat(1), qsub(1), qmgr(8), kill(2), setpgrp(2), signal(3c)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qjlist – list the commands in a batch job.

SYNOPSIS

qjlist *request-id*[@ *host*] ...

DESCRIPTION

qjlist lists the commands in each CXbatch request whose *request-ids* are listed on the command line. To list the commands in a CXbatch request, the invoking user must be the owner (the submitter of the request). The only exception to this rule occurs when the invoking user is the *superuser* or has CXbatch operator privileges as defined in the CXbatch manager database. Under these conditions, the invoker may specify any batch request.

A CXbatch request is always uniquely identified by its *request-id*. A *request-id* is always of the form *seqno* or *seqno.hostname*, where *hostname* identifies the machine from whence the request was originally submitted, and *seqno* identifies the sequence number assigned to the request on the originating host. If the *hostname* portion of a *request-id* is omitted, the local host is always assumed. The local host is searched for each given *request-id* unless a different host is specified with @ *host*.

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

qjlist returns an exit status describing what it did. If there were no errors, the exit status is zero. If one or more of the requests were not listed, the exit status is the number of requests that weren't listed. If a fatal error occurs and none of the requests are listed (e.g., a syntax error), the exit status is one of the codes defined in *<sysexits.h>*.

EX_USAGE	The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, or bad syntax in a parameter.
EX_NOHOST	The host specified did not exist.
EX_OSFILE	Some batch system file does not exist, cannot be opened, or has an error.
EX_NOPERM	You did not have sufficient permission to perform the operation.

SEE ALSO

qdel(1), qlimit(1), qstat(1), qsub(1), qmgr(8)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

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

SYNOPSIS

qlimit [*host-name ...*]

DESCRIPTION

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

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

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

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

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

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

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

Upon the successful queuing of a request in a batch queue, the set of limits under which the request will execute is frozen and will not be modified by subsequent *qmgr*(8) commands that alter the limits of the containing batch queue.

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

The three possible shell strategies are called:

- *fixed*
- *free*
- *login*

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

A shell strategy of *fixed* means that the same shell chosen by the system administrator is used to execute *all* batch requests (in the absence of a shell specification).

A shell strategy of *free* runs the batch request script *exactly* as would an interactive invocation of the script and is the default CXbatch shell strategy.

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

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

DIAGNOSTICS

qlimit returns an exit status describing what it did. If there were no errors, the exit status is zero. If one or more of the hosts were invalid or couldn't be reached, the exit status is the number of hosts that were invalid or couldn't be reached. If a fatal error occurs, the exit status is one of the codes defined in *<sysexits.h>*.

EX_OSFILE Some batch system file does not exist, cannot be opened, or has an error.

SEE ALSO

qdel(1), qjlist(1), qstat(1), qsub(1), qmgr(8)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qmapmgr – configures the CONVEX Extended Batch System (CXbatch) network

SYNOPSIS

qmapmgr

DESCRIPTION

The *qmapmgr* command builds the network database that CXbatch uses to direct files and messages. CXbatch needs this network database file no matter how simple the machine configuration.

This network database consists of three primary elements:

mid	A machine ID number that is unique in the CXbatch network being configured. CXbatch uses this *mid* to identify a specific machine. A maximum value of (2**31)-1 is permissible.
principle name	A machine name that is unique in the CXbatch network being configured.
alias	Names, other than the *principle name*, of machines in the network. Aliases are known only to the local CXbatch host.

To gain entry to the CXbatch network mapping manager, enter the *qmapmgr* command. You can enter any of the commands described below at the prompt. You can leave *qmapmgr* using the *exit* or *quit* commands.

COMMANDS

You can abbreviate commands to their minimum unambiguous length. For example, you can abbreviate *change name* to *ch n*. However, *c n* is not acceptable because the command *create* exists. You should enter all commands on a single command line. You can enter the following commands at the prompt.

add mid *mid principal name*
Adds a new machine to the configuration with the specified *mid* and *principle name*.

add name *alias mid*
Adds an *alias* for the machine with the specified *mid*.

change name *mid principle name*
Changes the *principle name* of the machine with the specified *mid*.

create
Creates a new network configuration database file.

delete mid *mid*
Removes the machine with the specified *mid* from the configuration database.

delete name *alias*
Deletes the given *alias* from the configuration database.

exit
Leaves the *qmapmgr* program.

get mid *name*
Displays the id of the machine with the specified *principle name* or *alias*.

get name *mid*
Displays the *principle name* of the machine with the specified *mid*.

help
Displays the available commands.

quit

Leaves the *qmapmgr* program.

show

Displays all *mid* entries with their corresponding *principle name* and *aliases*.

SEE ALSO

qmgr(8)

FILES

/etc/nmap – directory that contains network database

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qmgr – CXbatch queue manager program

SYNOPSIS

qmgr [*command* [*options*]]

DESCRIPTION

qmgr is the primary utility used to configure, administer and operate the CXbatch system. *qmgr* recognizes three classes of users: managers, operators, and users. The superuser is a manager by default. All other managers and operators are assigned using *qmgr*. Managers can use any of the *qmgr* commands. Operators can use a subset of the commands. Users who are not granted manager or operator privileges can use only a few of the *qmgr* commands to display status information and manipulate their own requests.

The CXbatch system is made up of batch and pipe queues that transport and run batch requests. A batch queue holds requests for scheduled, perhaps delayed, processing. A pipe queue is a queue that can pass queued requests on to other pipe queues or batch queues. A batch request is a set of commands, or a shell script, that is to be run non-interactively with the results being returned to the user. The running of the request is handled by the CXbatch system rather than directly by the user.

qmgr can be invoked to run a single command given as the command line argument or, if no argument is given, as a command interpreter. As an interpreter, commands will be repeatedly prompted for and executed. The *qmgr* command set is extensive and will be discussed in functional groups below. Command keywords are case insensitive and can be abbreviated to their shortest unique substring. This is indicated in the command descriptions below by capitalizing the require substring. For example, when using the exit command, you must type at least 'ex' to distinguish it from the enable command. Thus the description for this command is:

EXit

Exit from the CXbatch manager subsystem.

INFORMATIONAL COMMANDS

The following commands return information about *qmgr* commands, CXbatch parameters and queue configuration. They are available to all users.

HElp [*command*]

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

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

SHOw All

Display the standard amount of information about *"limits supported"*, *managers*, *parameters*, and *queues*. See below.

SHOw LImits_supported

Display the list of CXbatch resource limit types that are meaningful on this machine. Note that users may request resource limits that are *not* meaningful on the machine where *qsub*(1) is invoked. If the request is to be executed on a remote machine where the limit is meaningful, then CXbatch honors it. Otherwise the unsupported limit is simply ignored.

SHOw LOng Queues [*queue* [*user*]]

Display in long format the status of all CXbatch queues on this host. If a *queue* is

specified, output is limited to that queue. If a *user* is specified, output downplays any requests not belonging to that user.

SHOw Managers

Display the list of authorized CXbatch managers.

SHOw Parameters

Display the general CXbatch parameters.

SHOw Queues [*queue* [*user*]]

Display the status of all CXbatch queues on this host. If a *queue* is specified, output is limited to that queue. If a *user* is specified, output downplays any requests not belonging to that user.

QUEUE MANAGEMENT

The following commands are used to define batch and pipe queues. They can be used only by CXbatch managers. For more information on the different types of queues, see the **QUEUE TYPES** section below.

CReate Batch_queue *queue* **PRiority** = *n* [**PIpeonly**]
[**Run_limit** = *n*] [**Import_dir** = {*Yes,No,Available*}]
[**Share_policy Fixed** = *user*] [**Share_policy User**]

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

The specification of the **Import_dir** attribute determines the availability of the user's current working directory to a request at runtime. See the **SEt Import_dir** command for more information.

The specification of a **Share_policy** affects how the CPU usage of jobs running in this *queue* is charged. See the **SEt SHAre_policy Fixed** and the **SEt SHAre_policy User** commands for more information.

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

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

ADd Alias *alias queue*

Add the specified queue *alias* to *"queue"*. A queue alias is an alternate name for a queue; any CXbatch command that requires a queue name will also work with an alias.

DElete Alias *alias*

Delete the specified queue *"alias"*. A queue alias is an alternate name for a queue; any CXbatch command that requires a queue name will also work with an alias.

DElete Queue *queue*

The *queue* is deleted. To delete a *queue*, no requests may be present in the *queue*, and

the *queue* must be disabled. (See "**DIsable Queue**" below.)

SEt DESCription = *(description) queue*

Change the description of the named CXbatch queue.

SEt PRiority = *priority queue*

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

QUEUE ACCESS

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

The following commands are used to grant access to queues. They can be used only by CXbatch managers.

In the **ADd** and **DElete** commands below a *group* or *user* can be specified in one of two ways: *name* or [*id*]. (The square brackets are literal characters used to indicate a gid or uid.)

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

The specified *group(s)* are added to the access list for *queue*.

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

The specified *user(s)* are added to the access list for *queue*.

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

The specified *group(s)* are deleted from the access list for *queue*.

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

The specified *user(s)* are deleted from the access list for *queue*.

SEt NO_Access *queue*

Specify that no one can place requests in *queue*.

SEt Unrestricted_access *queue*

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

BATCH QUEUE CONFIGURATION

The following commands are used to set the operating parameters for batch queues. They can only be used by CXbatch managers.

SEt ACCOunting = {*OFf,ON*} *queue*

Turn accounting on/off for a CXbatch batch queue. The queue named as a parameter of the command must already exist.

SEt ACTivity_id_offset = *offset queue*

Set the activity ID offset for a CXbatch batch queue. The queue named as a parameter of the command must already exist.

SEt CHKpntable = {*Yes,No,Available*} *queue*

Sets the status of the per-queue checkpoint resource. The queue must exist. The queue's

checkpoint resource value and the request's flags are used to determine if a request may be checkpointed.

If the checkpoint attribute is set to "Yes", then any request submitted to this queue will be checkpointed at CXbatch shutdown and may be checkpointed by the request owner or CXbatch operator, unless it explicitly requested not to be checkpointed. If this attribute is set to "Available", the request is checkpointable only if it specifically asked to be checkpointed. Requests in a queue with the checkpoint attribute set to "No" cannot be checkpointed by CXbatch.

SEt Import_dir = {*Yes,No,Available*} *queue*

Changes the **Import_dir** attribute for a *queue*. The queue must already exist. If the **Import_dir** attribute is set to *Yes*, any job submitted to this queue is run in the user's current working directory, unless the job specifically requests not to be imported. If the **Import_dir** attribute is set to *Available*, only jobs that specifically request to be imported are imported. If it is set to *No*, jobs are run in the home directory. When importing, if a job is executed on a remote machine, CXbatch tries to access the local directory using NFS mounts. **NOTE:** CXbatch will make temporary NFS mounts into the /tmp filesystem. Care should be taken that any automatic clean-up operations on the /tmp filesystem do not traverse NFS mount points.

SEt MAXimum Request_priority = *priority queue*

Set a limit on request priorities on a per queue basis. Requests queued with a priority higher than the queue's maximum will have their priority lowered to the maximum.

BATCH QUEUE LIMITS

The following commands are used to set the limits batch queues impose on their requests. Every batch queue on the local host have each of the following limits associated with it at all times. If a request already in the queue has asked for more than a new limit, it is given a grandfather clause and allowed to retain its original limits. A request which specifies a particular limit may only enter a batch queue if the queue's corresponding limit is greater than or equal to the request's limit. See the section on **LIMITS** below for more information on the syntax on the various limit arguments. These commands can only be used by CXbatch managers.

SEt CORefile_limit = (*limit*) *queue*

Set a per-process maximum core file size *limit* for a batch queue against which the per-process maximum core file size limit for a request may be compared.

SEt DAta_limit = (*limit*) *queue*

Set a per-process maximum data segment size *limit* for a batch queue against which the per-process maximum data segment size limit for a request may be compared.

SEt NIce_value_limit = *nice-value queue*

Set the UNIX *nice-value* limit for a batch queue, against which the *nice* value for a request may be compared. A request specifying a *nice-value* may only enter a batch queue if the queue's nice value is numerically less than (more willing to allow access to the CPU) or equal to the request's *nice* value. *nice-value* is an integer preceded by an optional negative sign.

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

Set a per-process maximum CPU time *limit* for a batch queue against which the per-process maximum CPU time limit for a request may be compared.

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

Set a per-process maximum permanent file size *limit* for a batch queue against which the

per-process maximum permanent file size limit for a request may be compared.

SEt STack_limit = (*limit*) *queue*

Set a per-process maximum stack segment size *limit* for a batch queue against which the per-process maximum stack segment size limit for a request may be compared.

SEt Working_set_limit = (*limit*) *queue*

Set a per-process maximum working set size *limit* for a batch queue against which the per-process maximum working set size limit for a request may be compared.

PIPE QUEUE CONFIGURATION

The following commands are used to set the operating parameters for pipe queues. They are only available to CXbatch managers.

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

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

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

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

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

Associate one or more *destination* queues with a particular pipe *queue*.

SEt PIpe_client = (*client*) *queue*

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

DESIGNATING MANAGERS AND OPERATORS

The following commands are used to specify CXbatch managers and operators. They can only be used by CXbatch managers.

A *manager* specification consists of an account name specification, followed by a colon, followed by either the letter *m* or the letter *o*. There are four ways to specify an account name:

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

(The square brackets are literal characters used to indicate a uid or mid.) If the account name specification is followed by *:m*, the account is designated as a CXbatch *manager* account, capable of using all *qmgr* commands. If the account name specification is followed by *:o*, the account is designated as a CXbatch *operator* account, capable of only using those commands appropriate for a CXbatch *operator*. The *root* account always has full privileges.

ADd Managers *manager* ...

The specified *manager(s)* are added to the list of authorized CXbatch managers with privileges as specified.

DElete Managers *manager* ...

The specified *manager(s)* are deleted from the list of authorized CXbatch managers.

SEt MANagers *manager ...*

The list of authorized CXbatch managers is set to the specified *manager(s)*.

GENERAL CXBATCH MANAGEMENT

The following commands are used to set the general operating parameters of CXbatch. Only CXbatch managers can use these commands.

SEt ACC_logfile *filename*

Change the file being used for CXbatch batch accounting.

SEt AId_mask = *mask*

Set the activity ID mask. Generally, this mask is the same as the spacing between aids in */etc/activities*. The following equation is used to determine the activity ID of the CXbatch job:

$$job\_aid = submitter\_aid - (submitter\_aid \ \% \ aid\_mask) + queue\_aid\_offset$$

where % is the modulus (remainder) function.

Warning: As shipped, the aid mask is one. When the mask is one, the above equation simplifies to *job_aid = submitter_aid + queue_aid_offset*. In this case, if a CXbatch job submits another CXbatch job, the second job has an activity ID of *submitter_aid + queue1_aid_offset + queue2_aid_offset*. This is generally not desirable! Setting the activity ID mask to the spacing in */etc/activities* prevents this from happening.

SEt CHEckpoint_directory *directory*

Specify the pathname of the checkpoint directory. All checkpoint files created by CXbatch after this command is issued will be placed in *directory*. Existing checkpoint files will not be moved. The *directory* must exist and be a valid directory.

SEt DEBug *level*

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

0 No debug
1 Minimum debug
2 Maximum debug

SEt DEFault Batch_request Priority *priority*

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

SEt DEFault Batch_request Queue *queue*

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

SEt DEFault DEStination_retry Time *retry_time*

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

SEt DEFault DEStination_retry Wait *interval*

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

SEt Global Per_user Run_limit = *run-limit*

Set the *global per-user run-limit* of the local system. The *global per-user run-limit* is the maximum number of requests that any given user can have running on the local system

at any given time. A *global per-user run-limit* of 0 turns off the enforcement of this limit.

SEt MAIl *userid*

Specify the *userid* used to send CXbatch mail.

SEt NO_Default Batch_request Queue

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

SEt SHEll_strategy FIxed = (*shell*)

Specify that *shell* should be used to execute all batch requests. *shell* must be the absolute path name of a command interpreter. (See the SHELL STRATEGIES section below for more information.)

SEt SHEll_strategy FRee

Specify that the *free* shell strategy should be used to execute all batch requests. The *free* shell strategy duplicates the shell choice that would have been made if the batch request script had been executed interactively. Under this strategy, the user's *login shell* is allowed to determine the shell to be used to execute the batch request. The user's *login shell* is the shell named within the user's entry in the password file (see *passwd*(4)). (See the SHELL STRATEGIES section below for more information.)

SEt SHEll_strategy Login

Specify that the *login* shell strategy should be used to execute all batch requests. Under the *login* shell strategy, the user's login shell is used to execute the batch request. The login shell is the shell named in the password file (see *passwd*(4)). (See the SHELL STRATEGIES section below for more information.)

GENERAL CXBATCH OPERATION

The following commands are used for starting and shutting down CXbatch. They can be used by CXbatch managers or operators.

SHUtdown [**Force**] [*seconds*]

Shutdown CXbatch on the local host.

Each running checkpointable batch request is checkpointed, and, if the checkpoint succeeds, terminated. Successfully checkpointed requests will be restarted from their checkpointed state when CXbatch is rebooted.

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

When the optional **force** keyword is present CXbatch ignores any checkpoint errors incurred during the shutdown.

STArt Cxbatch

Start CXbatch on the local host. This command will fail if CXbatch is currently running on the local host.

QUEUE OPERATION

The following commands are used in operating queues in CXbatch. They can be used by CXbatch managers or operators.

ABort Queue *queue* [*seconds*]

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

DIsable Queue *queue*

Prevent any more requests from being placed in this queue.

ENable Queue *queue*

If the queue is already enabled, this command has no effect. Otherwise, the queue is enabled to accept new requests.

MOVe Queue *queue1 queue2*

Move all requests currently in *queue1* to *queue2*. The request is moved regardless of any queue limit violations, access restrictions, or attribute violations.

Purge Queue *queue*

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

STArt Queue *queue*

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

STOp Queue *queue*

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

QUEUE OPERATING PARAMETERS

The following commands are used to set the operating parameters of CXbatch queues. They can be used by CXbatch managers or operators.

SEt PER_User Run_limit = *run-limit queue*

Change the *per-user run-limit* of a CXbatch batch queue. The *per-user run-limit* determines the maximum number of requests that any given user can have running in the queue at any given time. A *per-user run-limit* of 0 turns off the enforcement of this limit.

SEt Run_limit = *run-limit queue*

Change the *run-limit* of a CXbatch batch or pipe queue. The *run-limit* determines the maximum number of requests that are allowed to run in the queue at any given time.

SEt SHAre_policy Fixed = *user queue*

Change the *share-policy* of a CXbatch batch queue. A queue with a fixed share policy will charge the CPU usage of jobs run in that queue to *account*. There are two ways to specify a user: *username* or [*id*]. (The square brackets are literal characters used to indicate a uid.)

SEt SHAre_policy User *queue*

Change the *share-policy* of a CXbatch batch queue. A queue with a user share policy will charge the CPU usage of jobs run in that queue to the account from which the job was submitted.

REQUEST OPERATIONS

The following commands operate on CXbatch requests. They can only be executed by CXbatch managers or operators.

MOVe Request *requestid ... queue*

Move the request(s) named by the *requestid(s)* to the named queue. The request is moved regardless of any queue limit violations, access restrictions, or attribute violations.

RESume Request *requestid ...*

Resume execution of suspended requests. Resumed requests start out in the *queued* state. Once the resumed request is about to enter the *running* state, it will be restarted from its checkpointed state instead of being re-run in its entirety.

RUn Request *requestid ...*

Force the request(s) named by the *requestid(s)* to begin executing immediately. If running the request would exceed the current run limit of the queue, then the queue's run limit will be increased by one until the request finishes executing.

SUspend Request *requestid ...*

Temporarily freeze execution of the named requests. The requests are checkpointed and terminated. Of course, a request that fails to checkpoint will continue to execute. Only checkpointable requests may be suspended in this manner.

USER REQUEST OPERATIONS

The following commands operate on CXbatch requests. These commands may be used by non-privileged user in dealing with their own requests. Otherwise, they are limited to use by CXbatch managers or operators.

CHKpnt Request *requestid ...*

Checkpoint the request(s) named by the *requestid(s).* The state of the named batch request(s) is saved into a set of checkpoint files stored in the *checkpoint directory.*

DElete Request *requestid ...*

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

HOLd Request *requestid ...*

Makes the specified request(s), named by *requestid(s),* ineligible for running. The request(s) must be in the *queued* state prior to being held. The **RELease Request** command removes the effect of **hold**.

If a request is held by an operator, only an operator can release it.

MODify Request *field = value requestid ...*

Change the field of the request(s) specified by *requestid(s)* to be *value.*
The legal values for *field* are:

Priority - change the intra-queue request priority. A user can only decrease a request's priority, *operator* privileges are required to raise a request's priority.

MOVe My_request *requestid ... queue*

Move your request(s) named by the *requestid(s)* to the named queue. The request is not moved if any queue limits, access restrictions, or attributes would have prevented the request from being submitted to the new queue.

RELease Request *requestid ...*

Makes the specified request(s), named by *requestid(s),* eligible for running. The request(s)

must be in the *holding* state prior to being released.

QUEUE TYPES

CXbatch supports two different queue types that provide two very different functions. These two queue types are known as *"batch"* and *"pipe"*.

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

Queues of type *"pipe"* are used to send CXbatch requests to other *"pipe"* queues or *batch* queues. In general, *"pipe"* queues act as the mechanism that CXbatch uses to transport *"batch"* requests to distant queues on other remote machines. It is also perfectly legal for a *"pipe"* queue to transport requests to queues on the same machine.

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

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

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

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

SHELL STRATEGIES

The execution of a batch request requires the creation of a shell process to interpret the shell script that defines the batch request. On many UNIX systems, there is more than one shell available (e.g., */bin/csh*, */bin/ksh*, */bin/sh*). To deal with this problem, CXbatch allows a shell pathname to be specified when a batch request is first submitted (*qsub* option *-s*).

If no particular shell is specified for the execution of the request, CXbatch must have some other means of deciding which shell to use when spawning the request. The solution to this dilemma has been to equip CXbatch with a batch request shell strategy that can be configured as necessary by the local system administrators.

The batch request shell strategy determines the shell to be used when executing a batch request on the local host that fails to identify any specific shell for its execution. Three such shell strategies can be configured for CXbatch, and they are known by the names of *fixed*, *free*, and *login*.

A shell strategy of *fixed* causes the request to be run by the *fixed shell*, the path name of which is configured by the system administrator. Thus, a particular CXbatch installation may be configured with a *fixed* shell strategy where the default shell used to execute all batch requests is defined as the Bourne shell.

A shell strategy of *free* causes the user's login shell (as defined in the password file), to be executed. This shell is, in turn, given a path name to the batch request shell script, and it is the user's login shell that actually decides which shell should be used to interpret the script. The *free* shell strategy therefore runs the batch request script exactly as would an interactive invocation of the script and is the default CXbatch shell strategy.

The third shell strategy of *login* causes the user's login shell (as defined in the password file) to be the default shell used to interpret the batch request shell script.

The strategies of *fixed* and *login* exist for host systems that are short on available free processes. In these two strategies, a single shell is executed, and that same shell is the shell that executes all of the commands in the batch request script (barring shell exec operations in any user startup files: *.profile*, *.login*, *.cshrc*).

The shell strategy as configured for any particular host can always be determined by the CXbatch *qlimit* command.

LIMITS

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

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

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

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

Whitespace can appear anywhere between the principal tokens, with the exception that no whitespace can appear around the decimal point. **NOTE:** The *milliseconds* value may be ignored if the system does not support such granularity.

Example time *"limit-values" are:*

center, tab(;); l l . 1234 : 58 : 21.29;– 1234 hrs 58 mins 21.290 secs 12345;– 12345 seconds 121.1;– 121.100 seconds 59:01;– 59 minutes and 1 second

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

.*fraction* [*units*]

or

integer [.*fraction*] [*units*]

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

center, tab(;); l l . *b*;– bytes *w*;– words *kb*;– kilobytes (2^10 bytes) *kw*;– kilowords (2^10 words) *mb*;– megabytes (2^20 bytes) *mw*;– megawords (2^20 words) *gb*;– gigabytes (2^30 bytes) *gw*;– gigawords (2^30 words)

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

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

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

If a batch request specifies a limit that cannot be enforced by the underlying UNIX implementation, the limit is ignored, and the batch request operates as though there were no limit (other than the obvious physical maximums) placed upon that resource type. (See the *"qlimit"*(1)

command to find out what limits are supported by a given machine.)

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

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

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

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

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

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

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

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

Upon the successful queuing of a request in a batch queue, the set of limits under which the request will execute is frozen and will not be modified by subsequent *qmgr*(8) commands that alter the limits of the containing batch queue.

DIAGNOSTICS

qmgr returns an exit status describing what the last *qmgr* command did. If there were no errors, the exit status is zero. If one or more of the operations failed, the exit status is the number of operations that failed. If a fatal error occurs and command completely fails, (ex, a syntax error), the exit status is one of the codes defined in *<sysexits.h>*.

EX_USAGE The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.

EX_NOHOST The host specified did not exist.

EX_OSFILE Some batch system file does not exist, cannot be opened, or has an error.

EX_TEMPFAIL Temporary failure; retry the request at a later time.

SEE ALSO

qdel(1), qjlist(1), qlimit(1), qstat(1), qsub(1), qmapmgr(8)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qps – display process status of CXbatch related processes

SYNOPSIS

qps [*queue-name ...*]
qps **–r** *request-id*
qps **–{p|q}** *process-id*

DESCRIPTION

qps prints information about CONVEX CXbatch related processes.

If no queues are specified, information is printed about all CXbatch related processes, including the daemon processes. Otherwise, information is printed for the specified queues only. Only batch queues on the local system are significant; pipe queues and remote queues do not have processes running on the local system.

Information about processes pertaining to a particular request can be obtained by using the **-r** *request-id* option. Only a single request-id can be specified.

For each process, *qps* prints the queue name (QUEUE), the request ID (REQ), the process ID (PID), the state (STAT) of the process, CPU time (TIME) used by the process (including both user and system time) and which command is running (COMMAND). More information about these fields can be found on the *ps(1)* man page.

Using the **-p** *process-id* option, you can inquire whether a particular process is running from within the CXbatch system. If it is, a line is printed stating the queue and request to which the process is related and *qps* exits with a status of 0. Otherwise, you are informed that the process is not running from within the CXbatch system and *qps* exits with a status of 1. For the purposes of this inquiry, the top level daemons are not considered to be running under the CXbatch system, but the CXbatch shepherd processes are.

The **-q** *process-id* option is a silent version of the **-p** option. Nothing is printed, but the exit status of *qps* is set appropriately. Only a single process-id can be specified for either of these options.

DIAGNOSTICS

qps returns an exit status of 0 if no errors occur, unless the the **-p** or **-q** options are specified, in which case the exit status is 0 or 1 as described above. If a fatal error occurs, the exit status is one of the codes defined in *<sysexits.h>*.

EX_USAGE	The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.
EX_OSERR	An internal call to *ps*(1) failed. If this error occurs, first make sure a '/bin/ls laxwg' returns valid output, then check the status of CXbatch.

SEE ALSO

ps(1), qstat(1)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qrestart – restart checkpointed CXbatch request(s).

SYNOPSIS

qrestart [-f] [*request-id*]

DESCRIPTION

qrestart restarts the requests whose *request-ids* are listed on the command line.

Only successfully checkpointed requests may be restarted. To restart a request, the invoking user must be the owner of the request or have CXbatch operator privileges.

qrestart is typically only used if the automatic restart of the request by CXbatch failed.

The options have the following meanings:

-f Force restart of request even if non-restartable conditions exist in any of the processes of the request.

DIAGNOSTICS

qrestart returns an exit status describing what it did. If there were no errors, the exit status is zero. If one or more of the requests were not restarted, the exit status is the number of requests that weren't restarted. If a fatal error occurs and none of the requests are restarted (e.g., a syntax error), the exit status is one of the codes defined in *<sysexits.h>*.

EX_USAGE The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.

EX_OSFILE Some batch system file does not exist, cannot be opened, or has an error.

EX_TEMPFAIL Temporary failure; retry the command at a later time. EX_SOFTWARE Too many request-ids were specified.

REFERENCES

qsub(1), qchkpnt(1), chkpnt(1), restart(1), qmgr(8)

NOTES

CXbatch is an optional product; for more information, please contact your CONVEX sales representative.

NAME

qrun – force CXbatch request(s) to run.

SYNOPSIS

qrun *request-id ...*

DESCRIPTION

The *qrun* command forces each CXbatch request whose *request-id* is listed on the command line to immediately begin executing.

If spawing a force-run request would cause the run limit of the queue to be exceeded, CXbatch increases the run limit by one while the forced-run request is running.

CXbatch operator privileges are required to use this command. The target request(s) must reside in batch queues.

DIAGNOSTICS

qrun returns an exit status describing what it did. If there were no errors, the exit status is zero. If one or more of the requests were not run, the exit status is the number of requests that weren't deleted. If a fatal error occurs and none of the requests are run (e.g., a syntax error), the exit status is one of the codes defined in *<sysexits.h>*.

EX_TEMPFAIL Temporary failure; retry the transaction at a later time.

EX_USAGE The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.

EX_OSFILE Some batch system file does not exist, cannot be opened, or has an error.

EX_NOPERM You did not have sufficient permission to perform the operation.

EX_SOFTWARE
Too many request ids were specified.

SEE ALSO

qmgr(8)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qsa – show accounting information on CXbatch requests

SYNOPSIS

qsa -r [**-QU**] [**-q** *queue*] [**-u** *username*] [*acct-file*]
qsa -x [**-QU**] [**-q** *queue*] [**-u** *username*] [*acct-file*]
qsa -a [**-QU**] [**-q** *queue*] [**-u** *username*] [*acct-file*]
qsa -s [**-QU**] [**-q** *queue*] [**-u** *username*] [*acct-file*]

DESCRIPTION

qsa processes CONVEX CXbatch accounting file and outputs data about requests. By default, *qsa* reads from */usr/adm/batchacct*. Another file may be optionally specified on the command line.

qsa operates in one of four modes specified by the following command line flags:

r Raw mode. Accounting records are formatted and written to the standard output.

x Extended mode. Accounting records are processed, formatted and written to the standard output. This mode differs from raw mode in that time spent waiting in queue, time spent executing, and turnaround time are calculated and all time values are converted to ASCII using *ctime(3)*.

a Averaging mode. Accounting records are processed and averages for turnaround time, time waiting in queue, execution time, CPU time in user mode, CPU time spent in system, and I/O operations performed are written.

s Summing mode. Accounting records are processed and totals for turnaround time, time waiting in queue, execution time, CPU time in user mode, CPU time spent in system, and I/O operations performed are written.

At least one of these mode flags must appear.

Constraint flags control which records *qsa* processes. If none are present, *qsa* processes all records in the accounting file. Using these flags, the user may specify records by queue, by user, or by both. The following flags specify constraints:

Q All queues. Accounting records are processed grouped by each queue appearing in the accounting file. This flag may not appear with the **-q** flag.

q *queue*
Specific queues. Accounting records are processed grouped by queues specified in one or more occurrences of this flag. This flag may not appear with the **-Q** flag.

U All users. Accounting records are processed grouped by each user appearing in the accounting file. This flag may not appear with the **-u** flag.

u *username*
Specific users. Accounting records are processed grouped by users specified in one or more occurrences of this flag. This flag may not appear with the **-U** flag.

DIAGNOSTICS

qsa returns an error status describing what it did. If there were no errors, the exit status is zero. If a fatal error occurs then the exit status is one of the codes defined in *<sysexits.h>*.

EX_USAGE One of the following incorrect usages was specified: More that 20 occurrences of the **-q** or **-u** options, respectively, none or more than one of the mode options, **-r**, **-x**, **-a**, **-s**, was specified, the **-Q** option was specified with the **-q** option, the **-U** option was specified with the **-u** option, or an invalid option was specified.

EX_DATAERR The accounting file was not a regular file.

EX_NOINPUT Cannot open accounting file or unable to access CXbatch database files.

SEE ALSO

qmgr(8), ctime(3)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qsnapshot – dump the current CXbatch queue or network configuration

SYNOPSIS

qsnapshot [**-m**]

DESCRIPTION

qsnapshot dumps the current CXbatch queue or networking configuration as a series of *qmgr*(8) or *qmapmgr*(8) commands. The resulting output is suitable as input to a subsequent *qmgr*(8) or *qmapmgr*(8) command. By default, *qsnapshot* dumps the CXbatch queue configuration. The **-m** option causes the *qmapmgr*(8) network database to be dumped instead.

The *qsnapshot* command is useful for copying the CXbatch configuration from one system to another or (by editing the output) duplicating a complex queue(s) on the local system.

Recreating CXbatch Databases

There are no *qmgr*(8) or *qmapmgr*(8) commands which clear out the old configurations from their respective database. If it is necessary to do this, shutdown CXbatch and use one or both of the commands given below. Recreating the qmapmgr database will require you to recreate the qmgr database if any of the MID to machine mappings are changed. Some MIDs may be referenced in the qmgr database. These commands should be executed with caution, as should any recursive *rm*(1) commands containing wildcard characters.

To empty the *qmgr*(8) database:

*rm -f /usr/spool/nqs/private/root/database/**

To empty the *qmapmgr*(8) database:

*rm -f /etc/nmap/**

DIAGNOSTICS

qsnapshot returns an exit status describing what it did. If there were no errors, the exit status is zero. If a fatal error occurs, then the exit status is one of the codes defined in *<sysexits.h>*.

EX_USAGE	The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.
EX_OSFILE	Some batch system file does not exist, cannot be opened, or has an error.

SEE ALSO

qmgr(8), qmapmgr(8)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qstat – display status of CXbatch queue(s)

SYNOPSIS

qstat [**–a**] [**–l**] [**–m**] [**–u** *user-name*] [**–x**] [*queue-name ...*] [*queue-name@ host-name ...*]

DESCRIPTION

qstat displays the status of CONVEX CXbatch queues.

If no queues are specified, the current state of each CXbatch queue on the local host is displayed. Otherwise, information is displayed for the specified queues only. Queues may be specified either as *queue-name* or *queue-name@ host-name*. In the absence of a *host-name* specifier, the local host is assumed.

For each selected queue, *qstat* displays a *queue header* (information about the queue itself) followed by information about requests in the queue. Ordinarily, *qstat* shows only those requests belonging to the invoker. The following flags are available:

-a	Shows all requests.
-l	Requests are shown in a long format.
-m	Requests are shown in a medium-length format.
-u *user-name*	Shows only those requests belonging to *user-name*.
-x	The queue header is shown in an extended format.

The *queue header* always includes the queue-name, queue type, queue status (see below), an indication of whether or not the queue is *pipeonly* (accepts requests from pipe queues only), and the number of requests in the queue. An extended queue header also displays the priority and run limit of a queue, as well as the access restrictions, cumulative use statistics, server and destinations (if a pipe queue), and resource limits (if a batch queue).

By default, *qstat* displays the following information about a request: the *request-name*, the *request-id*, the owner, the relative request priority, and the current request state (see below). For running requests, the process group is also shown, as soon as it becomes available to the local CXbatch daemon.

qstat -m shows the following additional information: if the request was submitted with the constraint that it not run before a certain time and date, the constraining time and date are also displayed.

qstat -l shows the time at which the request was created, an indication of whether or not mail will be sent, where mail will be sent, and the user name on the originating machine. If a batch queue is being examined, resource limits, planned disposition of *stderr* and *stdout*, any advice concerning the command interpreter, and the umask value are shown.

The relative ordering of requests within a queue does not always determine the order in which the requests are run. The CXbatch request scheduler is allowed to make exceptions to the request ordering for the sake of efficient machine resource usage. However, requests appearing near the beginning of the queue have higher priority than requests appearing later, and are usually run before requests appearing later on in the queue.

QUEUE STATE

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

The first property determines whether or not requests can be submitted to the queue. If they can, the queue is said to be *enabled*. Otherwise the queue is said to be *disabled*. One of the words CLOSED, ENABLED, or DISABLED appears in the queue status field to indicate the respective queue states of: enabled (with no local CXbatch daemon), enabled (and local CXbatch daemon is present), and disabled. Requests can only be submitted to the queue if the queue is enabled and the local CXbatch daemon is present.

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

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

If queued requests ready to run are only prevented from doing so by the CXbatch request scheduler, and one or more requests are presently running in the queue, the queue is said to be *running*. If the same circumstances prevail with the exception that no requests are presently running in the queue, the queue is said to be *inactive*. Finally, if the CXbatch daemon for the local host upon which the queue resides is not running, but the queue would otherwise be in the state of *running* or *inactive*, the queue is said to be *shutdown*. The queue states describing the second principal property of a queue are therefore respectively displayed as STOPPED, STOPPING, RUNNING, INACTIVE, and SHUTDOWN.

REQUEST STATE

The state of a request may be *arriving*, *holding*, *waiting*, *queued*, *routing*, *running*, *departing*, or *exiting*.

- *arriving* The request is being enqueued from a remote host.
- *holding* The request is presently prevented from entering any other state (including the *running* state), because a *hold* has been placed on the request.
- *waiting* The request was submitted with the constraint that it not run before a certain date and time, and that date and time have not yet arrived.
- *queued* The request is eligible to proceed (by *routing* or *running*).
- *routing* The request has reached the head of a pipe queue and is receiving service.
- *departing* A request is *departing* from the time the pipe queue turns to other work until the request has arrived intact at its destination.
- *running* The request has reached its final destination queue and is actually executing.
- *exiting* The batch request has completed execution and will exit from the system after the required output files have been returned (to possibly remote machines).

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

DIAGNOSTICS

qstat returns an exit status describing what it did. If there were no errors, the exit status is zero. If one or more of the queues were not listed, the exit status is the number of queues that weren't listed. If a fatal error occurs and none of the queues are listed (ex, a syntax error), the exit status is one of the codes defined in *<sysexits.h>*.

EX_USAGE The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.

EX_NOUSER The user specified with *-u* did not exist.

EX_OSFILE Some batch system file does not exist, cannot be opened, or has an error.

SEE ALSO

qdel(1), qjlist(1), qlimit(1), qsub(1), qmgr(8)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qsub – submit a CXbatch request.

SYNOPSIS

qsub [*flags*] [*script-file*]

DESCRIPTION

qsub submits a batch request to CONVEX CXbatch.

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

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

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

–a – run request after stated time
–b – set the billing activity for request
–c – request is checkpointable
–cp – specify periodic checkpoint frequency
–e – direct *stderr* output to stated destination
–eo – direct *stderr* output to the *stdout* destination
–h – put request on hold after submitting
–i – request requires current directory to be imported
–ke – keep *stderr* output on the execution machine
–ko – keep *stdout* output on the execution machine
–l – run request under a login shell
–l*x* – establish a resource limit
–mb – send mail when the request begins execution
–me – send mail when the request ends execution
–mu – send mail for the request to the stated user
–ni – don't import the current directory
–nr – declare that batch request is not restartable
–nc – request is not checkpointable
–o – direct *stdout* output to the stated destination
–p – specify intra-queue request priority
–q – queue request in the stated queue
–r – assign stated request name to the request
–s – specify shell to interpret the batch request script
–t – signal process when the job completes
–x – export all environment variables with request
–y – append accounting data to stdout output file
–z – submit the request silently

If you request that a batch request be run under a login shell, the system and user startup files will be read (*/etc/login* and *~/.login* for C-shell or */etc/profile* and *$HOME/.profile* for Bourne shell). A C-shell login shell will also read */etc/logout* and *~/.logout* while exiting. The *~/.cshrc* file is read by the C-shell regardless of whether or not it is executed as a login shell.

The environment string ENVIRONMENT=BATCH is added to the environment so that shell scripts (and the user's *.profile* (Bourne shell) or *.cshrc* and *.login* (C-shell) scripts), can test for batch request execution when appropriate, and not (for example) perform any setting of terminal characteristics, because a batch request is not connected to an input terminal. For example, if your login shell is C-shell, the following *.login* file prevents *stty*, *tset*, and *msgs* from being run during batch jobs.

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

If your login shell is Bourne shell, the following *.profile* file has the same effect.

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

When CXbatch is configured on more than one machine, it is possible for users to submit jobs to remote machines in the batch network. However, CXbatch must ensure that the submitter has permission to access the remote machine. This is accomplished with the *user equivalence* mechanism described in *hosts.equiv*(5) (the same method that is used by *rlogin* and *rsh*). If the machines in the batch network are not equivalenced in the */etc/hosts.equiv* file, users must create a *.rhosts* file in their home directories. If this is not done, the submitter will not have access to the remote machine. Read the *hosts.equiv*(5) man page for more information.

LONG DESCRIPTION

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

1. Read the first line of the *script file*.
2. If the current line contains only whitespace characters, or the first non-whitespace character of the line is ":", goto step 7.
3. If the first non-whitespace character(s) of the current line is not "#" or "$!", goto step 8.
4. If the second non-whitespace character in the current line is *not* the "@" character, or the character immediately following the second non-whitespace character in the current line is *not* a "$", goto step 7.
5. If no "-" is present as the character *immediately* following the "@$" sequence, goto step 8.
6. Process the *embedded* flag, stopping the parsing process upon reaching the end of the line, or upon reaching the first unquoted "#" or "$!" character(s).
7. Read the next *script file* line. Goto step 2.
8. End. No more *embedded* flags are recognized.

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

```
#
# Batch request script example:
#
# @$-a "11:30pm EDT" -lt "21:10, 20:00"
#         # Run request after 11:30 EDT by default,
#         # and set a maximum per-process CPU time
#         # limit of 21 minutes and ten seconds.
#         # Send a warning signal when any process
#         # of the running batch request consumes
#         # more than 20 minutes of CPU time.
# @$-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 UNIX
#         # 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.
#
make all
```

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

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

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

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

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

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

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

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

Some valid *date-time* examples are:

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

–b [*group.*]*activity*
Set the billing activity for request. The *group* and *activity* arguments refer to entries in the */etc/group* and */etc/activities* files respectively. The *group.activity* combination must correspond to an entry in the */etc/actwho* file. If *group* is omitted, the current primary group of the submitting user is assumed. A request aways runs under the submitting user's default group. The *group* argument to this option is used only in verifying a user's access to the selected *activity*. See the *bill*(1) man page for more information.

-c Specify that this request is checkpointable. A request queued with this flag is checkpointed automatically before a CXbatch shutdown, and may be explicitly checkpointed using CXbatch commands.

-cp *period* Declare that this request should be checkpointed periodically by CXbatch at intervals of *period*. The *period* is specified as <[*number*] *unit*>, where *number* is a positive integer and *unit* is [*Hours* | *Days* | *Weeks*].

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

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

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

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

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

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

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

–h Put request on hold after submitting. The request is put into a HOLD (user hold) rather that a QUEUED state at the time of submittal. The hold can be removed using the *qmgr*(8) 'RELease Request' command.

–i Some jobs may require access to files located in the directory from which a job is

submitted. This option tells CXbatch that the current working directory should be imported before running this job. If the execution queue is on the same machine as the current working directory, CXbatch will change directories before starting the job. However, if the execution queue is on another machine, CXbatch will import the current directory with NFS. **NOTE:** CXbatch will make temporary NFS mounts into the /tmp filesystem. Care should be taken that any automatic clean-up operations on the /tmp filesystem do not traverse NFS mount points.

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

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

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

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

–l The submitted request will be run under a login shell. This was the default behavior in the V1.0 release of CXbatch, but is now only done if explicitly requested. See the discussion above regarding shell start-up files and refer to the appropriate shell man page for details on the differences between login and non-login shells. **NOTE:** Running a job request under a login C-shell will cause the C-shell to issue a warning into the request's output file. This is a function of the C-shell that cannot be supppressed by CXbatch.

–lx *limit-argument*

Set a resource limit. Available limits are summarized in the following table.

Resource Limits		
Limit option	**Limited resource**	**Enforcement**
–lc *size-limit*	per-process corefile size	truncation
–ld *size-limit*[,*warn-limit*]	per-process data-segment	request denied
–lf *size-limit*[,*warn-limit*]	per-process perm-file	SIGXFSZ
–lF *size-limit*[,*warn-limit*]	per-request perm-space	none
–lm *size-limit*[,*warn-limit*]	per-process memory	none
–lM *size-limit*[,*warn-limit*]	per-request memory	none
–ln *nice-value*	per-process nice value	setpriority(2)
–ls *size-limit*[,*warn-limit*]	per-process stack-segment	request denied
–lt *time-limit*[,*warn-limit*]	per-process CPU time	SIGXCPU
–lT *time-limit*[,*warn-limit*]	per-request CPU time	none
–lv *size-limit*[,*warn-limit*]	per-process temp-file	none
–lV *size-limit*[,*warn-limit*]	per-request temp-space	none
–lw *size-limit*	per-process working set	paging

The *per-process corefile size limit* sets the maximum size of a corefile created by a process. The *per-process data-segment size limit* sets the maximum size to which a process's data-segment can grow. The *per-process perm-file size limit* sets the maximum size to which a permanent file written to by a process can grow. The *per-request perm-file space limit* sets the maximum file space which can be used by permanent files opened for writing by all of the processes in a batch request. The *per-process memory size limit* sets the maximum amount of memory a process can consume. The *per-request memory space limit* sets the maximum amount of memory which can be used by all of the processes in a batch request. The *per-process nice value* sets the nice value for all processes comprising the running batch request. The *per-process stack-segment size limit* sets the maximum size to which a process's stack-segment can grow. The *per-process CPU time limit* sets the maximum amount of CPU time a process can consume. The *per-request CPU time limit* sets the maximum amount of CPU time all of the processes that constitute a batch request can consume. The *per-process temp-file size limit* sets the maximum size to which a temporary file written to by a process can grow. The *per-request temp-file space limit* sets the maximum file space which can be used by temporary files opened for writing by all of the processes in a batch request. The *per-process working set size limit* sets the maximum amount of physical memory each process within a running batch request should use.

Enforcement of limits is done by the underlying UNIX implementation, normally through termination by a signal. Per-process limits are enforced only on the process exceeding the limit; per-request limits are enforced on all the processes which make up a request. The optional warning limits allow warning notification to be made processes where such warnings are supported by the underlying UNIX implementation. Not all UNIX implementations support all the resource limits listed above. The *qsub* command will pass all specified limits on to the destination machine. If a batch request specifies a limit, and the machine upon which the batch request is eventually run does not support the enforcement of that limit, the limit is simply ignored.

The 'Enforcement' column of the table above indicates the ConvexOS specific

enforcement policy of each resource limit. At this time, all *warn-limits* are treated identically to the hard limits. ConvexOS makes no distinction between permanent and temporary files; all files are treated as permanent.

See the section entitled **LIMITS** for more information on the implementation of batch request limits and for a description of the precise syntax of a the various limit arguments.

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

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

-mu *user-name*

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

-ni A queue can be configured so that it tries to import the originating directory of each job it runs. If this option is specified, CXbatch does not import the current directory.

-nc Advise CXbatch that this request is not checkpointable. A request queued with this option will not be checkpointed at CXbatch shutdown, nor is it possible to checkpoint the request with any other CXbatch commands.

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

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

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

When CXbatch is shutdown by an operator command to the *qmgr*(8) CXbatch control program, a **SIGTERM** signal is sent to all processes in all running CXbatch requests on the local host, and all queued CXbatch requests are barred from beginning execution. After a finite number of seconds have elapsed (typically sixty, but this value can be overridden by the operator), all remaining processes in all remaining running CXbatch requests are killed by the signal **SIGKILL**.

For a CXbatch request to be properly restarted after a CXbatch shutdown, the **-nr** flag must not be specified, and the spawned batch request shell must ignore **SIGTERM** signals (which is done by default). The spawned batch request shell also must not exit before the final **SIGKILL** arrives. Because the batch request shell is spawning commands and programs, waiting for their completion, this implies that the commands and programs being executed by the batch request shell must also be immune to **SIGTERM** signals, saving state as appropriate before being killed by the final **SIGKILL** signal.

See the **CAVEATS** section below for more discussion about the restartability of batch requests.

-o [*machine:*][[*/*]*path/*] *stdout-filename*

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

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

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

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

If no **–o** [*machine:*][[/]*path/*] *stdout-filename* flag is specified, all *stdout* output for the batch request is sent to the file whose name consists of the first seven characters of the *request-name* followed by the characters: "*.o*", followed by the request sequence number portion of the *request-id* discussed below. In the absence of the **–ko** flag, this default *stdout* output file is placed on the machine that originated the batch request in the current working directory, as defined when the batch request was first submitted. Otherwise, the file is placed in the user's home directory on the execution machine.

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

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

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

The CXbatch manager can assign a maximum request priority on a per queue basis. The maximum request priority is a ceiling on priorities of requests submitted to that queue. A request that specifies a priority higher than the maximum for that queue has its priority lowered to the maximum.

–q *queue-name*[*@ host*]
Specify the queue to which the batch request is to be submitted. If no **–q** *queue-name*[*@ host*] specification is given, the user's environment variable set is searched for the variable **QSUB_QUEUE**. If this environment variable is found, the character string value for **QSUB_QUEUE** is presumed to name the queue to which the request should be submitted. If the **QSUB_QUEUE** environment variable is not found, the request is submitted to the default batch request queue, *if* defined by the local system administrator. Otherwise, the request cannot be queued, and an appropriate error message is displayed. The *host* specifies the host where the queue resides. If no *host* is given, the local host is assumed. Not all hosts accept remote submissions.

–r *request-name*
Assign the specified *request-name* to the request. In the absence of an explicit **–r**

request-name specification, the *request-name* defaults to the name of the *script file* (leading path name removed) given on the command line. If no *script file* was given, the default *request-name* assigned to the request is **STDIN**.

In all cases, if the *request-name* is found to begin with a digit, the character "R" is prepended to prevent a *request-name* from beginning with a digit. All *request-names* are truncated to a maximum length of 15 characters.

–s *shell-name*

Specify the absolute path name of the shell that is used to interpret the batch request script. This flag unconditionally overrides any *shell strategy* configured on the execution machine for selecting which shell to spawn in order to interpret the batch request script.

In the absence of this flag, the CXbatch system at the execution machine uses one of three distinct *shell choice strategies* for the execution of the batch request. Any one of the three strategies can be configured by a system administrator for each CXbatch machine.

The three shell strategies are called:

- *fixed*
- *free*
- *login*

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

A shell strategy of *fixed* means that the same shell (as configured by the system administrator) is used to execute all batch requests.

A shell strategy of *free* runs the batch request script *exactly* as would an interactive invocation of the script and is the default CXbatch shell strategy.

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

The *shell strategy* configured for a particular CXbatch system can be determined by the *qlimit*(1) command.

–t *process-id* Signal process *process-id* when the job completes execution. The signal sent depends on how the job completed. If the job runs to normal completion, **SIGTERM** is sent. If the job is aborted while running, **SIGUSR1** is sent. If the job is deleted before it starts executing, **SIGUSR2** is sent.

–x Export all environment variables. When a batch request is submitted, the current values of the environment variables **HOME**, **SHELL**, **PATH**, **USER**, and **MAIL** are saved for later recreation when the batch request is spawned, as the respective environment variables **QSUB_HOME**, **QSUB_SHELL**, **QSUB_PATH**, **QSUB_USER**, and **QSUB_MAIL**. Unless the **–x** flag is specified, no other environment variables are exported from the originating host for the batch request. If the **–x** flag option is specified, all remaining environment variables whose names do not conflict with the automatically exported variables are also exported with the request. These additional environment variables are recreated under the same name when the batch request is spawned.

–y Append accounting data to the stdout output file if accounting is enabled for the

queue in which the job runs. This data includes: queue, host, sequence number, remote host, submission time, start time, completion time, time spent executing in user mode, and time spent executing in the system.

-z Submit the batch request silently. If the request is submitted successfully, no messages are displayed indicating this fact. Error messages are, however, always displayed.

If the batch request is successfully submitted, and the **-z** flag has not been specified, the *request-id* of the request is displayed to the user. A *request-id* is always of the form *seqno.hostname*, where *seqno* refers to the sequence number assigned to the request by CXbatch, and *hostname* refers to the name of originating local machine. This identifier is used throughout CXbatch to uniquely identify the request, no matter where it is in the network.

The following events take place in the following order when a CXbatch *batch* request is spawned:

1. The process that will become the head of the *process group* for all processes comprising the batch request is created by CXbatch.
2. Resource limits are enforced.
3. The real and effective group-id of the process is set to the group-id as defined in the local password file for the request owner.
4. The real and effective user-id of the process is set to the real user-id of the batch request owner.
5. The user file creation mask is set to the value that the user had on the originating machine when the batch request was first submitted.
6. It the user explicitly specified a shell by use of the **-s** flag (discussed above), then that user-specified shell is chosen as the shell that will be used to execute the batch request script. Otherwise, a shell is chosen based upon the *shell strategy* as configured for the local CXbatch system. (See the earlier discussion of the **-s** flag for a description of the possible *shell strategies* that can be configured for a CXbatch system.)
7. The environment variables of HOME and SHELL, are set from the user's password file entry, as though the user had logged directly into the execution machine.
8. The environment string: ENVIRONMENT=BATCH is added to the environment so that shell scripts (and the user's *.profile* (Bourne shell) or *.cshrc* and *.login* (C-shell) scripts) can test for batch request execution when appropriate, and not (for example) perform any setting of terminal characteristics, because a batch request is not connected to an input terminal.
9. The environment variables of QSUB_WORKDIR, QSUB_HOST, QSUB_REQNAME, and QSUB_REQID are added to the environment. These environment variables equate to the obvious respective strings of the working directory at the time that the request was submitted, the name of the originating host, the name of the request, and the request *request-id*.
10. All of the remaining environment variables saved for recreation when the batch request is spawned are added at this point to the environment. When a batch request is initially submitted, the current values of the environment variables HOME, SHELL, PATH, USER, and MAIL are saved for later recreation when the batch request is spawned. When recreated however, these variables are added to the environment under the respective names QSUB_HOME, QSUB_SHELL, QSUB_PATH, QSUB_USER, and QSUB_MAIL to avoid the obvious conflict with the local version of these environment variables. Additionally, all environment variables exported from the originating host by the **-x** option are added to the

environment at this time.

11. The current working directory is then set to the user's home directory on the execution machine, and the chosen shell is exec'd to execute the batch request script with the environment as constructed in the steps outlined above.

Unless the –l option is specified, the chosen shell is exec'd as a non-login shell and no start-up files (except the *~/.cshrc* for a C-shell) are read. If the –l option is selected, the chosen shell is exec'd as though it were the *login* shell. If the Bourne shell is chosen to execute the script, */etc/profile* and the user's *.profile* file are read. If the C-shell is chosen, */etc/login* and the user's *.cshrc* and *.login* scripts are read and when the job exits, */etc/logout* and the user's *.logout* script are read.

If the user did not specify a shell for the batch request, CXbatch chooses which shell is used to execute the shell script, based on the *shell strategy* as configured by the system administrator. (See the earlier discussion of the *-s* flag.)

In such a case, a *free* shell strategy instructs CXbatch to execute the login shell for the user (as configured in the password file). The login shell is in turn instructed to examine the shell script file and fork another shell *of the appropriate type* to interpret the shell script, behaving *exactly* as an interactive invocation of the script.

Otherwise no additional shell is spawned, and the chosen *fixed* or *login* shell sequentially executes the commands contained in the shell script file until completion of the batch request.

QUEUE TYPES

CXbatch supports three different queue types that serve to provide three very different functions. These three queue types are known as *batch*, *pipe*, and *network*.

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

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

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

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

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

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

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

QUEUE ACCESS

CXbatch supports queue access restrictions. For each queue of queue type other than *network*, access may be either *unrestricted* or *restricted*. If access is *unrestricted*, any request may enter the queue. If access is *restricted*, a request can only enter the queue if the requester or the requester's login group has been given access to that queue (see *qmgr*(8)). Requests submitted by the superuser are an exception; they are always queued, even if the superuser has not explicitly been given access.

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

LIMITS

CXbatch supports many batch request resource limit types that can be applied to a batch request. The existence of configurable resource limits allows a CXbatch user to set resource limits within which his or her request must execute.

The syntax used to specify a *limit-value* for one of the *limit-flags* (**-l***x*), is quite flexible and describes values for three general limit categories. These three general categories respectively deal with *time limits*, priority (*nice value*) limits and file/memory *size limits*.

All the *limit-flags* expect a single *limit-argument*. If the *limit-argument* consists any whitespace which will cause the it to be passed to qsub as multiple tokens, it should be enclosed within double quotes or otherwise escaped such that is is passed a a single, character-string token. The optional *warn-limit* should also be included as part of the same token. This also applies to *limit-arguments* associated with *limit-flags* embedded within the batch request *script file*.

For *finite* CPU time limits, the *limit-value* is expressed in the reasonably obvious format:

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

Whitespace can appear anywhere between the principal tokens, with the exception that no whitespace can appear around the decimal point. **NOTE:** The *milliseconds* value may be ignored if the system on which the request is run does not support such granularity.

Example time *limit-values are:*

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

Priority limits are expressed as a small, signed integer in the range acceptable for *nice values* on the execution machine. In general, increasingly negative *nice values* cause the relative execution priority of a process to increase, while increasingly positive *nice values* causes the relative priority to decrease. Because different UNIX implementations often support different finite ranges of *nice values*, CXbatch allows the specification of *nice values* that can eventually turn out to be outside the limits for the UNIX implementation running at the execution machine. In such cases, CXbatch simply binds the specified *nice values* limit to within the necessary range as appropriate.

For the *finite* size limits the acceptable syntax is:

.*fraction* [*units*]

or

integer [.*fraction*] [*units*]

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

insensitive strings:

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

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

For all limit types with the exception of the *nice* limit-value (–l*n*), it is possible to state that no limit should be applied. This is done by specifying a *limit-value* of "unlimited" or any initial substring thereof. Whenever an *infinite limit-value* is specified for a particular resource type, the batch request operates as though no explicit limits have been placed upon the corresponding resource, other than by the limitations of the physical hardware involved.

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

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

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

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

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

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

The identical internal conversion process as described in the preceding paragraphs is also performed for each *finite limit-value* configured for a particular batch queue using the *qmgr*(8) program.

After all of the applicable *limit-values* have been converted as described above, each resulting *limit-value* is then compared against the corresponding *limit-value* as configured for the destination batch queue. If, for every type of limit, the batch queue *limit-value* is *greater than* or *equal to* the corresponding batch request *limit-value*, the request can be successfully queued, provided that no other anomalous conditions occur. For request *infinity limit-values*, the corresponding queue *limit-value* must also be configured as infinity.

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

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

Upon the successful queuing of a request in a batch queue, the set of limits under which the request will execute is frozen and will not be modified by subsequent *qmgr*(8) commands that alter the limits of the containing batch queue.

CAVEATS

When a batch request is spawned, a new *process-group* is established such that all processes of the request exist in the same *process-group*. If the *qdel*(1) command is used to send a signal to a batch request, the signal is sent to all processes of the request in the created *process-group*. However, should one or more processes of the request choose to successfully execute a *setpgrp*(2) system call, such processes will *not* receive any signals sent by the *qdel*(1) command. This can lead to "rogue" requests whose constituent processes must be killed by other means such as the *kill*(1) command.

It is extremely wise for all processes of a CXbatch request to catch any SIGTERM signals. By default, the receipt of a SIGTERM signal causes the receiving process to die. CXbatch sends a SIGTERM signal to all processes in the established *process-group* for a batch request as a notification that the request should be prepared to be killed, either because of an *abort queue* command issued by an operator using the *qmgr*(8) program, or because it is necessary to shutdown CXbatch and all running requests as part of a general shutdown procedure of the local host.

It must be understood that the spawned *shell* ignores SIGTERM signals. If the current immediate child of the shell does not ignore or catch SIGTERM signals, it will be killed by the receipt of such, and the shell will go on to execute the next command from the script (if there is one). In any case, the shell will not be killed by the SIGTERM signal, though the executing command will have been killed.

After receiving a SIGTERM signal delivered from CXbatch, a process of a batch request typically has sixty seconds to get its "house in order" before receiving a SIGKILL signal (though the sixty second duration can be changed by the operator).

All batch requests terminated because of an operator CXbatch shutdown request that did not specify the **–nr** flag are considered restartable by CXbatch and are requeued (provided that the batch request shell process is still present at the time of the SIGKILL signal broadcast as discussed above), so that when CXbatch is rebooted, such batch requests will be respawned to continue execution. It is, however, up to the user to make the request restartable by the appropriate programming techniques. CXbatch simply spawns the request again as though it were being spawned for the first time.

Upon completion of a batch request, a mail message can be sent to the submitter (see the discussion of the **–me** flag above). In many instances, the completion code of the spawned Bourne or C-Shell is displayed. This is the value returned by the shell through the *exit*(2) system call.

Lastly, there is no good way to echo commands executed by unmodified versions of the Bourne and C shells. While the Bourne and C shells can be spawned in such a fashion as to echo the commands they execute, it is often very difficult to tell an echoed command from genuine output produced by the batch request.

Thus, one of the better ways to write the shell script for a batch request is to place appropriate lines in the shell script of the form:

echo "*explanatory-message*"

where the echoed message should be a meaningful message chosen by the user.

DIAGNOSTICS

qsub returns an exit status describing what it did. If there were no errors, the exit status is zero. If a fatal error occurs and the request is not submitted (e.g., a syntax error), the exit status is one of the codes defined in *<sysexits.h>*, or one if none of the *<sysexits.h>* codes are appropriate.

EX_USAGE	The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.
EX_OSFILE	Some batch system file does not exist, cannot be opened, or has an error.
EX_TEMPFAIL	Temporary failure; retry the request at a later time.
EX_NOPERM	You did not have sufficient permission to perform the operation.
EX_NOINPUT	The script file does not exist or is not readable.

SEE ALSO

bill(1), mail(1), qdel(1), qjlist(1), qlimit(1), qstat(1), qmgr(8) kill(2), setpgrp(2), signal(3c)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qwatch – watch status of CXbatch queue(s)

SYNOPSIS

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

DESCRIPTION

qwatch monitors the status of CONVEX CXbatch and periodically reports statistics about queues and/or requests.

Initially, *qwatch* displays queue headers for the first five queues. If there are more than five queues, additional pages can be displayed by typing the number of the desired page. Only five pages (25 queues) can be display by *qwatch*.

By typing the letter (a-y) corresponding to a queue, you can monitor the activity of that queue. The first page of the requests within the selected queue is displayed along with the queue header. If there are more that 16 requests in the queue, additional pages can be displayed by typing the number of the desired page. Only nine pages (144 requests) can be displayed by *qwatch*.

The command line switches, which may be given in any order, are:

-c *count* If *count* is positive, *qwatch* automatically exits after *count* screen updates have been completed. If *count* is 0, *qwatch* keeps updating until stopped. The default value for *count* is 0.

-i *interval* Specifies how many seconds to wait between each screen update. The default is five seconds.

The following keys are interpreted by *qwatch*:

<SPACE>

<RETURN> Force *qwatch* to update the screen and restart the timer.

<CTRL-L> Replot the screen.

[1-9] Select a display page. (Only pages with information on them may be selected.)

[a-y] Select a queue. (Only available from queues display page.)

- Return to queues display page from requests display page.

The *SIGINT* signal, usually generated by **<CTRL-C>** or **<DEL>**, is used to exit *qwatch*. Additionally, *qwatch* can be interrupted and later resumed with the *SIGTSTP* signal, which is typically generated by **<CTRL-Z>** from *csh*.

QUEUE STATE INFORMATION

Refer to the qstat(1) manpage for an explanation of the information displayed by *qwatch*.

DIAGNOSTICS

qwatch returns an exit status describing what it did. If there were no errors, the exit status is zero. If a fatal error occurs, then the exit status is one of the codes defined in *<sysexits.h>*.

EX_UNAVAILABLE
The terminal does not support the capabilities necessary for *qwatch* to work properly.

EX_USAGE The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.

SEE ALSO

qstat(1), syspic(8)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

Index

A

B

C

D

E

H

I

L

M

N

O

P

Q

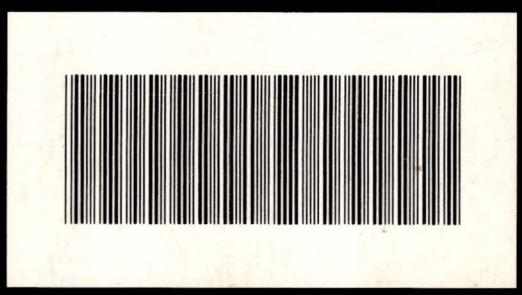
R

S

T

U

V



Order Number
DSW-182

Document Number
710-006730-004