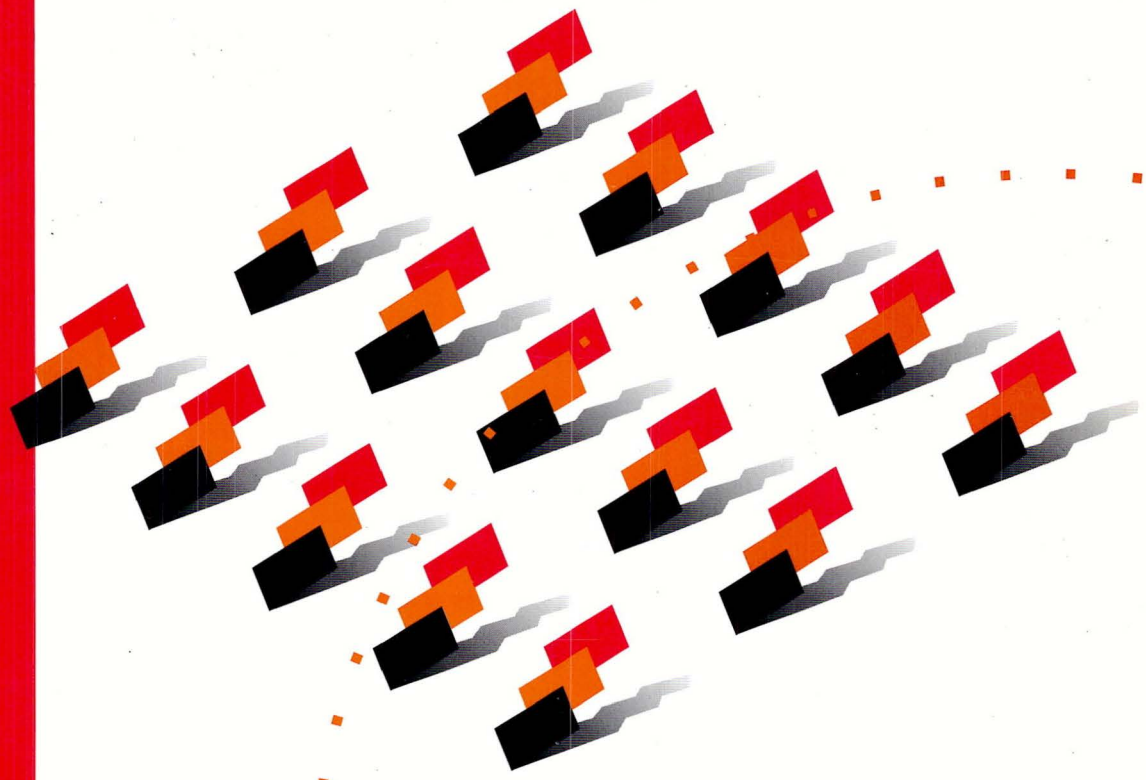


CONVEX

CXbatch System Manager's Guide

Fourth Edition

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

Order No. DSW-182

Fourth Edition
February 1994

CONVEX Press
Richardson, Texas
United States of America

CONVEX
CXbatch
System Manager's Guide

Order No. DSW-182

Printed in the United States of America

Revision information for

CONVEX
CXbatch
System Manager's Guide

Edition	Document No.	Description
Fourth		
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

Preface

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.

Most information in this manual applies equally to ConvexOS and ConvexOS/Secure. Where this is not true, it is explicitly stated.

If you are running ConvexOS/Secure, you may be running in an evaluated configuration. This means that you are running ConvexOS/Secure in the environment evaluated by the National Computer Security Center (NCSC) for its level of trust. Restrictions and requirements for running ConvexOS/Secure in an evaluated configuration are described throughout this document.

Refer to the *Managing ConvexOS/Secure: Configuration Guide* for more details about trusted systems.

Using 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 Chapter 3 and Chapter 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 Chapter 6 and Chapter 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 online man pages appear in the form adb(1), where the name of the man page is followed by its section number enclosed in parentheses.

Command syntax

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.

Notes, cautions, and warnings

This document presents notes, cautions, and warnings in the following formats.

Note

A Note highlights supplemental information.

Caution

A Caution highlights information necessary to avoid damage to the system.

Warning

A warning highlights information necessary to avoid injury to personnel.

Associated documents

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

For more information on the ConvexOS or ConvexOS/Secure 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.
- *Managing ConvexOS/Secure: Configuration Guide* (DSW-577) explains how to configure ConvexOS/Secure.
- *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 (DSW or DHW number) or the exact title of the document.

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
- All other locations, contact the local CONVEX office.

You can also use the `contact` utility, if you would like to report any problems you may have with ConvexOS or its associated

documentation. For more information refer to the contact(1) man page or the appendix "Reporting problems" in the *ConvexOS Primer, Managing ConvexOS: Operations Guide*, or *Managing ConvexOS/Secure: Operations Guide*.

CXbatch processing 1

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).

Caution

For ConvexOS/Secure sites, networking is not supported in the evaluated configuration. It is recommended that you do not provide network sccess in a trusted environment. This caution applies throughout the document where network configurations are discussed.

When a user submits a request, 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. When running ConvexOS, The CXbatch manager assigns operator privileges to a user in order for them to act as a CXbatch operator.

When running ConvexOS/Secure, operators are designated through the authif utility. See the chapter "Setting up administrative accounts" in *Managing ConvexOS/Secure: Configuration Guide* for details.

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`	When running ConvexOS, start CXbatch on the local machine. When running ConvexOS/Secure, this command is disabled. CXbatch should be started only by booting the system.

CXbatch managers

Users with CXbatch manager privileges have access to all qmgr commands. Refer to the qmgr(8) man page for a complete list.

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	Is 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 not 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
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
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
226	nqs	Restart request
227	nqs	Checkpoint done
228	nqs	Copy open files on checkpoint
229	nqs	Keep checkpoint files upon apr event
230	nqs	set next sequence number
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, due to an apr (automatic processor recovery) event, 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 memory size limit = UNLIMITED
Per-request memory size limit = UNLIMITED
Per-process execution nice value = 0
Per-process stack size limit = UNLIMITED
Per-process CPU time limit = 36000.0
Per-request CPU time limit = UNLIMITED
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 several 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
11 Copy open files on checkpoint: No
12 Keep checkpoint files: No
13 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.

11 `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).

Keep checkpoint files defines whether or not CXbatch preserves a requests checkpoint files when the request aborts due to an apr (automatic processor recovery) event. Of course the request had to have been checkpointed for this to be meaningful.

12 `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
   Per-process data size limit = UNLIMITED
2  Per-process permanent file size limit = UNLIMITED
3  Per-process memory size limit = UNLIMITED
   Per-request memory size limit = UNLIMITED
4  Per-process execution nice value = 0
5  Per-process stack size limit = UNLIMITED
   Per-process CPU time limit = 36000.0
6  Per-request CPU time limit = UNLIMITED
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.

`Per-process memory size limit` is the maximum total address space that a process can have for this request. If this limit is exceeded, ConvexOS sends the appropriate signal to the offending process.

`Per-request memory size limit` is the cumulative maximum total address space that all processes in a single request can have. If this limit is exceeded, ConvexOs sends the appropriate signal to all processes in the request.

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, ConvexOs sends a signal to the process. If this signal is not caught or ignored, the process exits.

`Per-request CPU time limit` is the maximum cumulative total CPU time for all processes in this single request. If this limit is exceeded, ConvexOs sends the appropriate signal to all processes in the request.

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 on the system console.

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*.

Step 7 If nameserver is not running, skip this step. If nameserver is running, use the `add name` command to specify the fully qualified host name as an alias for each machine you are adding. For example, if the fully qualified name for host A is hostA.berkley.edu, enter

```
add name hostA.berkley.edu 1
```

where 1 is the MID assigned to hostA.

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 8 Exit qmapmgr by entering

```
exit
```

Step 9 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 10 Repeat Step 3 through Step 9 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

When running ConvexOS, refer to this section to set up CXbatch managers and operators for each machine. When running ConvexOS/Secure, refer to the chapter "Setting up administrator accounts" in Managing ConvexOS/Secure: Configuration Guide to assign managers and operators.

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:

- The `add manager` command in qmgr gives users access to all commands defined by CXbatch as operator or manager commands.
- The `op` utility in ConvexOS specifies access to some or no commands.

Using add manager command

Under ConvexOS, 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. The op utility is not available in ConvexOS/Secure.

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:

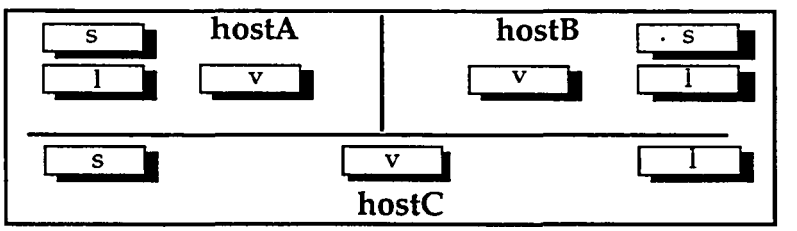
- **Batch**—Batch queues are used only to execute CXbatch batch requests. They hold requests for scheduled, perhaps delayed, processing by subsystems within CXbatch.
- **Pipe**—Pipe queues are routing queues that do not directly process requests but, instead, transmit requests to other queues on either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

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

- **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.

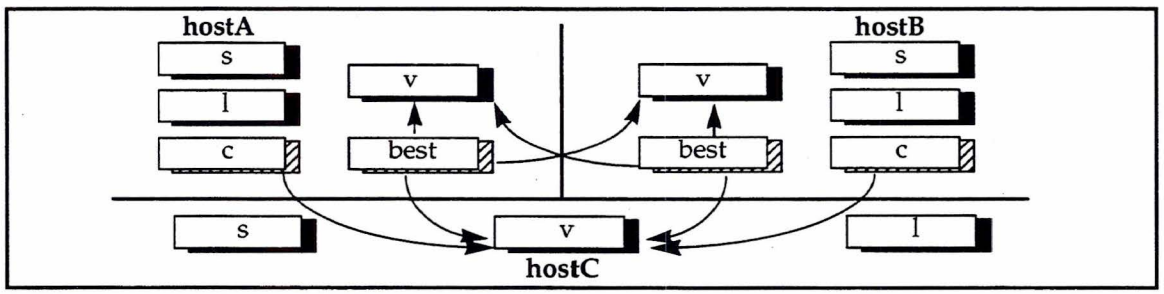
In the sample configuration, users who belong to the development group can submit long-running jobs to the *v* queue on either *hostA*, *hostB*, or *hostC* via the *best* pipe queue on *host*A. Users who belong to the design group can submit long-running jobs to the *v* queue on either *hostA*, *hostB* 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 When running ConvexOS, log in as a CXbatch manager on the host machine for which you are adding batch queues. Under ConvexOS/Secure, `su` to your general administrator account.

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 what 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 interqueue 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 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.

Note

Steps 8 and 9 describe only two of many resource limits CXbatch supports. You should decide on and set all appropriate resource limits; see qlimits(1) and qmgr(8).

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
Keep checkpoint files: 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 memory size limit = UNLIMITED
Per-request memory size limit = UNLIMITED
Per-process execution nice value = 0
Per-process stack size limit = UNLIMITED
Per-process CPU time limit = 600.0
Per-request CPU time limit = UNLIMITED
Per-process working set limit = UNLIMITED
```

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* or *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 root
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 CHKPNTABLE = Available short
SET COPY_OPEN_FILES = No short
SET Keep_chkpnt_files = No 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_REQUEST CPU_LIMIT = ( unlimited ) short
SET PER_PROCESS MEMORY LIMIT = ( unlimited ) short
SET PER_REQUEST MEMORY LIMIT = ( unlimited ) 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
SET PER_USER RUN_LIMIT = 0 short
SET DESCRIPTION = (Queue for short jobs.) short
ADD ALIAS s short
```

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.

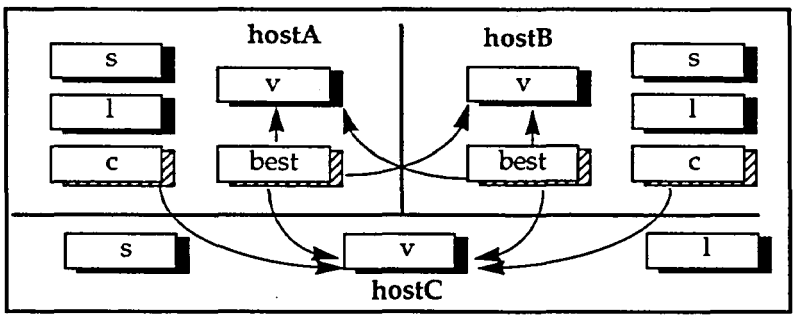
Step 2 When running ConvexOS, log in as a CXbatch manager on the host machine for which you are adding pipe queues. Under ConvexOS/Secure, su to your general administrative account.

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 *hostA* and two on *hostB*: 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 root
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 CHKPNTABLE = Available short
SET COPY_OPEN_FILES = No short
SET Keep_chkpnt_files = No 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_REQUEST CPU_LIMIT = ( unlimited ) short
SET PER_PROCESS MEMORY LIMIT = ( unlimited ) short
SET PER_REQUEST MEMORY LIMIT = ( unlimited ) 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
SET PER_USER RUN_LIMIT = 0 short
SET DESCRIPTION = (Queue for short jobs.) short
ADD ALIAS s short
```

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 When running ConvexOS, log in as a CXbatch manager on the host machine for which you are deleting queues. Under ConvexOS/Secure, su to your general administrative account.

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 When running ConvexOS, log in as a CXbatch manager on the host machine for which you are configuring accounting. Under ConvexOS/Secure, su to your general administrative account.

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 Configuration Guide* or *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 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 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 Change the owner and group of this directory to batch. Enter

```
chown batch.batch /usr/nqs/chkpnt
```

Step 4 Set the permissions on this directory to 755. The world must be able to read the directory in order to restart checkpointed requests.

Step 5 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 6 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 7 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 8 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 9 Repeat Step 7 through Step 8 for each queue you wish to configure with checkpoint restart on this host.

Step 10 Repeat Step 1 through Step 9 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 When running ConvexOS, log in as a CXbatch manager on the host you are configuring. Under ConvexOS/Secure, `su` to your general administrative account.

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 Under ConvexOS, check the /etc/rc/std file to be sure the following line exists. Under ConvexOS/Secure check the /etc/rc2.d/rc.std file:

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

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

Step 6 Add the line shown in Step 5 if it does not exist in the /etc/rc.local file under ConvexOS or the /etc/rc2.d/rc.local file under ConvexOS/Secure.

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.
```

Step 8 If you want to change the debug level, when running ConvexOS log in as a CXbatch manager on the host you wish to configure. Under ConvexOS/Secure, su to your general administrative account.

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 when running ConvexOS, log in as a CXbatch manager or operator. Under ConvexOS/Secure, su to your general administrative account (either manager or operator). 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. When running ConvexOS you must log in as a CXbatch manager or operator before performing these tasks. Under ConvexOS/Secure su to your general administrative account (either manager or operator).

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.

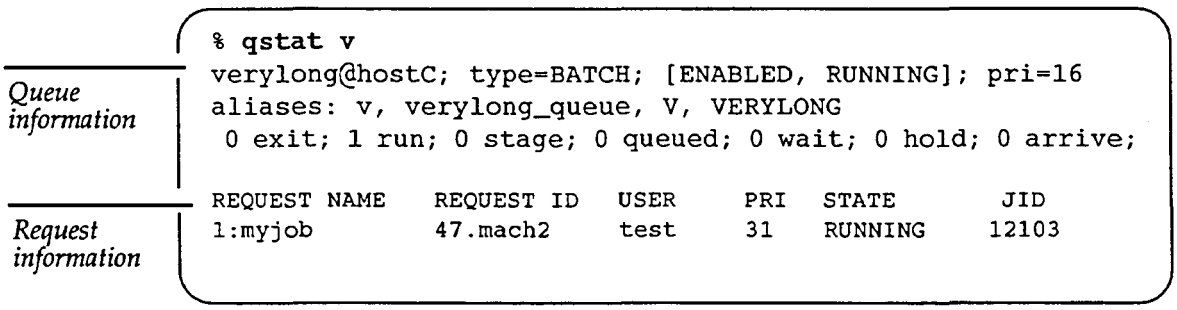
hostname is the name of the machine that receives the request. If *hostname* is omitted, the local host is assumed.

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       JID
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 Job id 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 is 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 sometimes 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 sometimes 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 request from one queue to another. The request cannot be running. If the request is running, suspend it first using the `suspend request` command.

`move request` is a qmgr utility command, so you must start qmgr before you can use `move request`. 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 *unit*s 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: When running ConvexOS, grants one or more users access to qmgr commands.

CXbatch manager privileges are required to use this command.

When running ConvexOS/Secure, manager and operator must be assigned using the authif utility. See the chapter "Setting up administrative accounts" in *Managing ConvexOS/Secure: Configuration Guide* for details.

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*

Interqueue 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*

Interqueue 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 (*des1*, *des2*, *des3*).

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.

When running ConvexOS/Secure, manager and operator must be deleted using the authif utility. See the chapter "Setting up administrative accounts" in *Managing ConvexOS/Secure: Configuration Guide* for details.

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 nonrunning 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. If the request is running you must first suspend the request using the `suspend request` command. 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 intraqueue 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 Keep_chkpnt_files

change the keep_chkpnt_files attribute for a batch queue

Format:

```
SEt Keep_chkpnt_files = option queuename
```

Description:

Defines whether or not requests checkpoint files are preserved or not following aborting due to an apr (automatic processor recovery) event.

CXbatch manager privileges are required to use this command.

Parameters:

option

Can be one of the following:

`Yes`	Requests running in the queue have their checkpoint files automatically preserved when the request aborts due to an apr event. The user can override this setting for individual requests using the `-nkc` option of `qsub`. Requests submitted with the `-nkc` option do not automatically have their checkpoint files preserved when aborting due to an apr event.
`Available`	Requests submitted with the `-kc` option to `qsub` have their checkpoint files automatically preserved when they abort due to an apr event. Requests not submitted with the `-kc` option do not automatically have their checkpoint files preserved when aborting due to an apr event.
`No`	Requests do not automatically have their checkpoint files preserved when aborting due to an apr event.

queuename

Name of the queue that you are setting.

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: When running ConvexOS, 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.

When running ConvexOS/Secure, manager and operator must be assigned using the authif utility. See the chapter "Setting up administrative accounts" in *Managing ConvexOS/Secure: Configuration Guide* for details.

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 intraqueue 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 warning limit set, ConvexOS sends the signal SIGINFO to the offending process. If the signal is not caught or is ignored, the process exits.

If any process tries to use a greater amount of CPU time than the hard limit set, ConvexOS sends the signal SIGKILL to the offending process.

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 MEMORY_LIMIT

set per-process maximum CPU time limit for queue

Format: `SEt PER_Process Memory_limit = (`*limit*`)` *queuename*

Description: Sets the cumulative maximum amount of memory a batch request process can use. CXbatch uses this limit if a request is submitted to the queue without a maximum process memory limit defined.

When a request is submitted to a queue, CXbatch checks any process memory size limit defined to ensure that it does not exceed the process memory size 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 process memory size limit than the limit allowed by the queue, the request is rejected.

The maximum process memory 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 an amount of memory that exceeds the new limit, a warning message is displayed.

If any process tries to use a greater amount of memory than the warning limit set, ConvexOS sends the signal SIGINFO to the offending process. If the signal is not caught or is ignored, the process exits.

If any process tries to use a greater amount of memory than the hard limit set, ConvexOS sends the signal SIGKILL to the offending process.

CXbatch manager privileges are required to use this command.

Parameters: *limit*

Maximum size in bytes of cumulative memory space 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 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_REQUEST CPU_LIMIT

set per-process maximum CPU time limit for queue

Format:

`SEt PER_Request Cpu_limit = (`*limit*`)` *queuename*

Description:

Sets the maximum amount of CPU time a batch request 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 request 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 request CPU time limit than the limit allowed by the queue, the request is rejected.

The maximum request 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 request tries to use a greater amount of CPU time than the warning limit set, ConvexOS sends the signal SIGINFO to all processes in the offending request. If the signal is not caught or is ignored, the request exits.

If any request tries to use a greater amount of CPU time than the hard limit set, ConvexOS sends the signal SIGKILL to all processes in the offending request.

CXbatch manager privileges are required to use this command.

Parameters:

limit

Cumulative maximum CPU time allowed for all 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_REQUEST MEMORY_LIMIT

set per-request maximum CPU time limit for queue

Format:

```
SEt PER_Request Memory_limit = (limit) queuename
```

Description:

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

When a request is submitted to a queue, CXbatch checks any request memory size limit defined to ensure that it does not exceed the request memory size 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 request memory size limit than the limit allowed by the queue, the request is rejected.

The maximum request memory 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 an amount of memory that exceeds the new limit, a warning message is displayed.

If any request tries to use a greater amount of memory than the warning limit set, ConvexOS sends the signal SIGINFO to all processes in the offending request. If the signal is not caught or is ignored, the request exits.

If any request tries to use a greater amount of memory than the hard limit set, ConvexOS sends the signal SIGKILL to all processes in offending request.

CXbatch manager privileges are required to use this command.

Parameters:

limit

Maximum size in bytes of cumulative memory space for any request. 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 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 interqueue 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*

Interqueue 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)
 Per-process memory size limit (-lm)
 Per-request memory size limit (-lM)
 Nice value (-ln)
 Stack segment size limit (-ls)
 Per-process cpu time limit (-lt)
 Per-request 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:

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

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 memory size limit = UNLIMITED
 Per-request memory size limit = UNLIMITED
 Per-process execution nice value = 4
 Per-process stack size limit = UNLIMITED
 Per-process CPU time limit = 600.0
 Per-request 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 =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 = /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: When running ConvexOS, 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.

When running ConvexOS/Secure, this command is disabled. CXbatch must be started only by booting the system.

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 runtime 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.

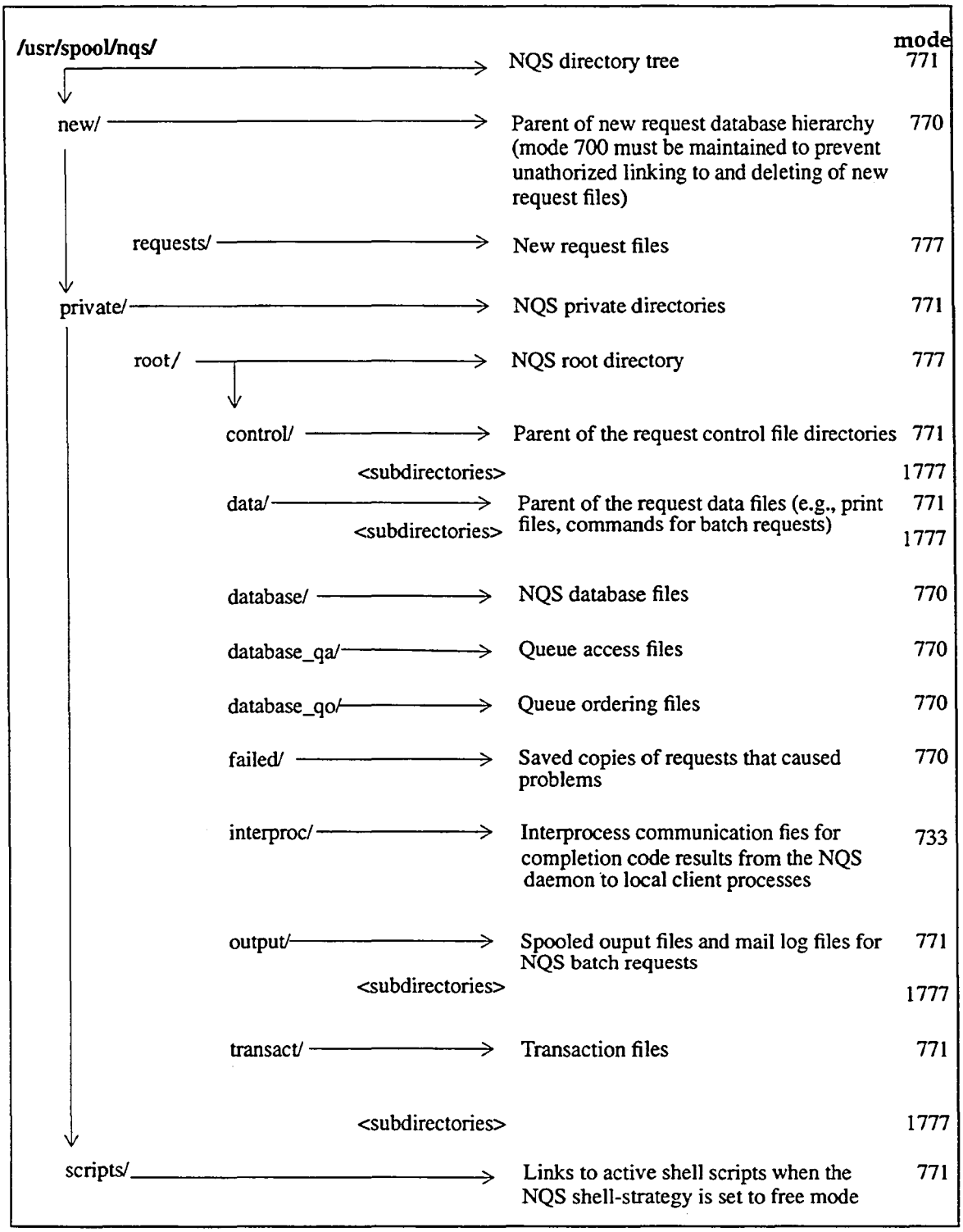
For more information on the contents or access mode of any of these files, please contact the CONVEX Technical Assitance Center (TAC).

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



Index

A

B

C

D

E

H

I

L

N

O

P

Q

R

S

T

U

V

ORDER NUMBER
DSW-182

DOCUMENT NUMBER
710-006730-005