CONVEX CXbatch Concepts

Order No. DSW-185

Second Edition
September 1990

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX
CXbatch Concepts

Order No. DSW-185
Document Number 710-006630-003

Printed in the United States of America.

Revision Information for

CONVEX
CXbatch Concepts

Edition	Document No.	Description
Second	710-006630-003	Released with CONVEX CXbatch V2.0, September 1990, contains information about new functionality, changes to existing functionality, new `qsub` command options, new `qmgr` command options.
First Edition, Rev. 1	710-006630-001	Released with CONVEX CXbatch V1.1, April 1990.
First	710-000003-201	Released with CONVEX CXbatch V1.0, February 1989.

Contents

Using this Book

Purpose and Audience

This document describes the overall functioning of the CONVEX CXbatch system. It also defines the concepts and terms relevant to CXbatch. It addresses:

- System managers who install, configure, and maintain CXbatch.
- System programmers who wish to submit jobs for batch execution.
- Anyone with a knowledge of batch processing who wants to use CXbatch.

This document is part of a four-volume set consisting of *CONVEX CXbatch Concepts, CONVEX CXbatch User's Guide, CONVEX CXbatch System Manager's Guide,* and *CONVEX CXbatch System Manager Utilities Reference.*

Organization

This manual is organized as follows:

- Chapter 1 introduces CXbatch and describes its major functions. It explains the importance of defining an optimal configuration and gives an overview of the new CONVEX Share Scheduler and CONVEX Checkpoint Restart utilities for CXbatch users.
- Chapter 2 defines terms used in CXbatch documentation and in the batch process.
- Chapter 3 describes how CXbatch operates. It describes communication flow between segments of CXbatch and discusses user-level job-submission commands.
- Appendix A provides instructions for reporting software or documentation problems to the CONVEX Technical Assistance Center (TAC).

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.

Reader Response

If you have comments or questions about the contents of this book, please notify the CONVEX documentation department by using the Reader's Forum at the end of this document.

Associated Documents

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

You can order the following documents from CONVEX Computer Corporation for more information on the ConvexOS operating system:

- *CONVEX UNIX Primer* (DSW-133) - introduces new users to the ConvexOS operating system.
- *ConvexOS Programmer's Reference* (DSW-332) - standard reference for the ConvexOS operating system.
- *Managing ConvexOS: Configuration Guide* (DSW- 030) - contains information needed to configure a CONVEX supercomputer.
- *Managing ConvexOS: Operations Guide* (DSW- 031) - contains information needed to operate a CONVEX supercomputer.
- *CONVEX Share Scheduler Concepts* (DSW- 261) - contains conceptual information on the CONVEX Share Scheduler.
- *CONVEX Share Scheduler System Mnager's Guide* (DSW- 265) - contains information needed to manage and maintain the Share Scheduler.
- *CONVEX Checkpoint Restart Guide* (DSW-350) - contains detailed information and instructions for using Checkpoint Restart.
- *ConvexOS Utilities User's Guide* (DSW- 005) - provides detailed information on the CONVEX Assembler, Loader, text editors, and `adb` (assembly language debugger.

Ordering Documentation

To order the current edition of this or any other CONVEX document, send requests to:
CONVEX Computer Corporation
Customer Service
PO Box 833851
Richardson, TX 75083-3851
USA

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

Notational Conventions

This section discusses notational conventions used in this guide.

Command Syntax

Consider this example:

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

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

General Conventions

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

- **`Bold constant-width font`** identifies user input in examples.
- *Italics*
  - Designate user-supplied variables in a command-line example.
  - Introduce new and important terms.
  - Identify variables in mathematical equations.
  - Indicate titles of documents.
- `Constant-width font` is used to designate input and output, including:
  - Command names and options.
  - System calls.
  - Data structures and types.
  - Directives, program statements, display examples, printout examples, and error messages returned.
- Horizontal ellipsis (...) shows repetition of the preceding item(s).
- Vertical ellipses show that lines of code have been left out of an example.
- Words and abbreviations that indicate keyboard keys you press are identified in a distinctive bold type. For example, **RETURN** refers to the carriage return key. Words separated by a hyphen indicate two keys that you must press simultaneously. For example, **CTRL-X** indicates that you must press and hold down the **CTRL** key and then press the **X** key.
- The word "enter" in a phrase such as "enter `ls`" means that you type the command and then press **RETURN**.
- References to the *ConvexOS Programmer's Reference* appear in the form adb(1), where the name of the man page is followed by its section number enclosed in parentheses.

Overview 1

Introduction

CONVEX CXbatch permits users to submit jobs for batch execution in queues. Jobs may be submitted and executed on either the local machine or a remote machine, depending on the batch system configuration. Different resource quotas may be defined on a queue-by-queue basis. CXbatch can be configured to use algorithms to schedule jobs for execution on the most lightly-loaded processor.

CXbatch software relies on two underlying utilities: `qmgr` and `qmapmgr`.

System managers use `qmgr` to define and control the action and attributes of CXbatch queues; batch operators use `qmgr` to control the action of CXbatch queues. `qmgr` also contains commands for the user to examine queues and control individual requests.

System managers use `qmapmgr` to configure the network database used by CXbatch to map between CXbatch and each machine in the configuration.

CXbatch has individual commands to submit, control, and check the status of job requests.

Benefits of Optimal Configuration

The goal of CXbatch configuration is the best possible turnaround time for each job, within the constraints of your system. Ideally, this is accomplished when each batch job is processed as soon as it is submitted. This action, however, decreases performance for interactive users. To avoid a bottleneck caused by many jobs being started simultaneously, you can specify a run limit for each queue. System managers can assign both global per-user run limits and per-queue per-user run limits.

A secondary goal of configuration is equal CPU usage for all queues. Jobs that require little CPU time and memory should not have to wait for jobs with much larger requirements. Also, very large jobs should run with a lower priority than small jobs.

Generally, these goals are met with:

- Separate queues for smaller and larger jobs.
- Lower priority for larger jobs.
- Large jobs being shipped to the machine with the lightest load, if more than one machine is available.

Integration with CONVEX Share Scheduler

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

System managers may configure each queue to charge its request's CPU resources to either:

- A fixed uid's share group.
- The submitting user's share group.

Using the latter method may affect a user's interactive performance.

Shares are assigned by system managers. Refer to Chapter 5 of *CXbatch User's Guide,* "Using CXbatch with the Share Scheduler," for detailed Share information.

Also refer to the *CONVEX Share Sceduler Guide,* First Edition.

Integration with Checkpoint Restart

CXbatch V2.0 allows users and batch administrators to use the ConvexOS Checkpoint Restart utility when submitting batch jobs.

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.

Users and operators can checkpoint and restart processes that are running under CXbatch by using Checkpoint Restart features built into CXbatch:

`qchkpnt`	Allows the user to checkpoint any request being run by CXbatch that meets checkpointing requirements.
`qmgr`	Provides options that allow the user to checkpoint processes being run through CXbatch and requeue them so that they may later be restarted.
`qrestart`	Allows the user to restart a request that meets restarting requirements.
`qsub`	Provides options that allow the user to adjust the checkpoint characteristics of a batch request and assign checkpointing at periodic intervals.

Users and managers should investigate the CPU and disk space overhead associated with Checkpoint Restart.

Only the Checkpoint Restart information relevant to CXbatch is presented in the CXbatch documentation set. For more information on Checkpoint Restart, refer to the *ConvexOS Checkpoint Restart Guide* (DSW-350).

Definition of Terms 2

Terms

This chapter defines terms used in CXbatch processing and documentation. Specifically, this chapter defines terms associated with:

- ❑ Batch requests.
- ❑ Queues.
  - types.
  - states.
  - access.
  - attributes.
  - lmits.
- ❑ Request states.
- ❑ Daemon definitions.

Batch Request

A batch request is one or more commands, possibly in the form of a shell script, 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 direct interaction with the user. A typical batch request is a program containing commands that perform large-scale, detailed computations on a static data file.

Queue

A queue is a list of batch requests that are ready and waiting to execute. A user submits a request to CXbatch; CXbatch assigns it to a particular queue. The request waits in that queue until it is selected for execution. CXbatch assigns a priority to each request; the request is executed according to its priority. CXbatch executes that request and routes any output to the specified recipient.

Queue Types

Two types of CXbatch queues route and run batch requests:

- Batch queues — Used only to execute CXbatch batch requests, they hold requests for scheduled, perhaps delayed, processing by subsystems within CXbatch.
- Pipe queues — Routing queues that do not process requests directly, but transmit requests to other pipe queues or batch queues on either the same or a remote machine.

Each pipe queue has a set of destination queues that are possible recipient queues for requests submitted to that pipe queue. A destination queue can be either a batch queue or another pipe queue.

Each request received by a pipe queue spawns an associated server known as a pipe client. 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. The pipe client routes and delivers the request to the selecteddestination queue.

Queue States

Two properties define the state of a queue:

- Ability of the queue to accept requests.
- Ability of the queue to execute requests.

Three states indicate whether a queue can accept requests:

- Closed — CXbatch is not running on the local machine.
- Disabled — CXbatch is running on the local machine, but the queue is not accepting requests.
- Enabled — CXbatch is running on the local machine, and the queue is accepting requests.

Five states indicate the ability of the queue to execute requests:

- 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.

Queue Access

Queue access defines permissions granted to a user to communicate with a queue. There are three designations for access to a batch queue:

- Unrestricted — The queue can receive any request from any submitter.
- Restricted — The queue can receive only those requests submitted by a specified group(s) or user(s).
- Pipeonly — The pipe queue accepts requests only if they have been sent from another pipe queue. This access option is specified when the pipe queue is created.

The groups and users who have access to a queue are defined by the batch manager with the `add groups`, `add users`, `set no_access`, and `set unrestricted_access` commands within the `qmgr` utility. For more information, see the qmgr(8) man page, the *CONVEX CXbatch System Manager's Guide*, and the *CONVEX CXbatch System Manager Utilities Referencel*.

A request submitted by the superuser is an exception to these limitations; superuser requests are always queued.

Queue Attributes

Batch and pipe queues have attributes that indicate properties of the queue. The following attributes apply to both batch and pipe queues:

- Queue priority — Indicates the relative order in which batch queues are searched for eligible requests. This number can be from 0 to 63.
- Cumulative system space time — 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.
- Cumulative user space time — Total amount of user time used by completed batch jobs since the queue was created.
- Access — Permissions granted to a user to communicate with a queue. (Refer to the "Queue Access" section for the definition of access types.)

The following attributes apply only to batch queues:

- Accounting — Indicates whether batch accounting has been activated. Refer to Chapter 3,"Accounting," of the *CXbatch System Manager's Guide* for complete information on accounting procedures.
- Activity ID offset — Number added by CXbatch to a job's activity identification (ID) before the request is executed. The activity ID offset is typically an integer from one to nine, with zero reserved for jobs not submitted to a batch queue. The activity ID is used for accounting purposes.
- Import directory — Indicates whether the files in the submitter's current working directory are to be imported (mounted on the processing machine) before the request is executed. Options for the import directory attribute are:
  - Yes — The queue imports the directory unless told otherwise with `qsub -ni`. (For more information, refer to the qsub(1) man page.)
  - Available — The queue can import the directory, but it will only do so if requested with `qsub -i`.
  - No — The queue cannot import the directory, and it will reject any job that requires input files to be imported.
- Share policy — Indicates whether the share policy is set to User, which charges CPU usage to to the request submitter, or to Fixed, which charges CPU usage to a specified account.

The following screen example illustrates the attributes of batch queue *s*.

```
# qmgr
Mgr: show long q s
Queue for short jobs 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;
Accounting: Off
Activity ID offset: 1
Cumulative system space time = 0.000000 seconds
Cumulative user space time = 0.000000 seconds
Unrestricted access
Import directory: Yes
Checkpoint: Not available
Share policy fixed = s
Per-process core file size limit = UNLIMITED
Per-process data size limit = UNLIMITED
Per-process permanent file size limit = UNLIMITED
Per-process execution nice value = 0 <DEFAULT>
Per-process stack size limit = UNLIMITED
Per-process CPU time limit = 36000.0 <DEFAULT>
```

In addition to attributes, each queue has certain defined limits. These are discussed in the next section.

Queue Limits

CXbatch supports resource limits that can be set for each CXbatch queue. Limits that CXbatch recognizes are:

- Run — Maximum number of requests allowed to run in the batch queue simultaneously.
- Core file size — Maximum size of a core file. If a process attempts to create a core file exceeding this maximum size, the core file is not written.
- Data segment size — Maximum size of the data segment for a process.
- Permanent file size — Maximum size of a file created by a process within the request. 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.
- *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.
- Stack segment size — Maximum size of the stack segment for a process.
- CPU time — Total CPU time that a batch request can use. If this limit is exceeded, CXbatch sends a signal to the process. If nothing is defined in the process to handle that signal, the process is killed.
- Working set — Maximum amount of physical memory that can be used by a process.

A CXbatch system manager can establish resource limits for a specific queue. All requests submitted to the queue must comply with these limits. The set of limits applied to a batch request is always restricted to the set of limits directly supported by ConvexOS. If a batch request specifies a limit that cannot be enforced by ConvexOS, the request ignores the limit and operates as though there were no limit (other than the obvious physical maximums) placed upon that resource type.

For requests submitted through pipe queues, batch request limits are forwarded with the request through a pipe queue to the destination batch queue.

CXbatch compares each request limit with the corresponding batch queue limit. The request can be successfully placed in the queue if:

- The corresponding batch queue limit is defined as unlimited.
- The corresponding batch queue limit is greater than or equal to the specific batch request limit.

CXbatch performs these resource limit checks for requests received either directly from a user with the `qsub` command or indirectly from a pipe queue. It is impossible for a batch request to be queued in a CXbatch batch queue if any resource limit checks fail. After CXbatch has successfully queued a request to a batch queue, it freezes the set of limits under which the request will execute. This request will not be affected by subsequent `qmgr` commands that alter limits of that particular batch queue.

The following screen example illustrates the resource limits of batch queue *s*.

```
# qmgr
Mgr: show long q s
Queue for short jobs 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;
Accounting: Off
Import directory: Yes
Share policy fixed = s
Per-process core file size limit = 1 megabytes <DEFAULT>
Per-process data size limit = 1 megabytes <DEFAULT>
Per-process permanent file size limit = 1 megabytes <DEFAULT>
Per-process execution nice value = 0 <DEFAULT>
Per-process stack size limit = 1 megabytes <DEFAULT>
Per-process CPU time limit = 60.0
```

Request States

A request in a CXbatch queue can be in one of seven states:

- Running — A request is executing.
- Routing — A request is in a pipe queue is not executing but is being routed and delivered to a destination queue.
- Queued — A request is ready for execution and is waiting to enter the next state of either running or routing. This is the most common state.
- Waiting — A request is waiting for a specified amount of time to pass. Typically, this state suggests that a pipe queue could not route the request and will attempt to route the request later.
- Arriving — A request has been sent from a pipe queue to a destination batch queue, and the destination batch queue is receiving the request.
- Departing
  - batch queue — A batch request has finished running, and CXbatch is copying output files back to the user's directory.
  - pipe queue — A batch request has left the pipe queue but has not been received by the destination queue.
- Holding — A hold has been placed on the request. This request cannot enter any other state until the hold is removed.

Daemon Definitions

A daemon is a process that runs constantly and provides a particular service to CXbatch when needed. 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 to syslogd (if defined) and notifies the batch manager if the error is fatal.
- `netserver` — Is executed by `netdaemon` to handle operations that require a substantial amount of time, such as queuing a request.
- `shepherd` — A child of the `nqsdaemon` that watches over batch jobs, it is responsible for setting up the job environment and returning output files.

The Batch Process 3

Introduction

This chapter describes the CXbatch job submission process. It describes communication flow between daemons that make up the CXbatch system and user-level commands that submit and control job requests. Specifically, this chapter provides information about:

- Submitting a request.
- Running a request.
- Routing a job to a pipe queue.
- Balancing loads.
- Receiving notification when a job is completed.
- Controlling a request.
- Deleting a request.
- Displaying the status of requests and queue limits.
- Accounting procedures.

CXbatch has both batch and pipe queues; the flow of requests through each of them is different. A batch queue processes a job request received directly from a submitter or another queue. A pipe queue routes the request to another pipe or batch queue. The flow of a job through each queue type is explained in detail in other sections of this chapter.

Chapter 2 of this book contains full definitions of terms used in this chapter.

For more information on how to configure CXbatch, refer to the *CONVEX CXbatch System Manager's Guide*. For more information on how to use CXbatch, refer to the *CONVEX CXbatch User's Guide*.

Batch Queue Processing

This section describes how a job request submitted to a batch queue is processed.

Submission to a Local Batch Queue

A user submits a job request to CXbatch using the `qsub` command. `qsub` sends the request to the `nqsdaemon`. The `nqsdaemon` checks the `qsub` qualifiers (if the user has included any on the command line) against the 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 and sends a transaction code back to qsub. If the request is accepted, nqsdaemon runs the job when its turn comes in the batch queue.

Submission to a Remote 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 username on the local machine matches the username on the remote machine. If these conditions are true, it accepts the request. The remote nqsdaemon then checks the qsub qualifiers (if the user has included any on the 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 and sends a transaction code back to qsub.

Processing a Batch Queue 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 at the time of submission.

Pipe Queue Processing

This section describes how a job request submitted to a pipe queue is processed.

Submission to a Local Pipe Queue

A user first submits a job request to CXbatch using the qsub command, which 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 and sends a transaction code back to qsub. If the request is accepted, nqsdaemon routes the request when the request moves to the top of the queue.

Submission to a Remote Pipe 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.

Routing of a Pipe Queue Request

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 two possible pipe clients are `pipeclient` and `pipeldav` (pipe load average). Each pipe queue is configured 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.

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`.

Load Balancing

Users can route a job from a pipe queue 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

$$mla = \frac{la + (ql \times w)}{scale}$$

where *mla* is modified load average, *la* is load average, *scale* is processor speed, *ql* is queue length, and *w* is weight.

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 `rwho` software, so that load information for each batch queue host is available for the `pipeldav` queue host. 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, spending 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.

Request Completion Notification

If you want to be notified when a request finishes executing, use the `-me` or `-mu` *user-name* option with the `qsub` command. `-me` causes CXbatch to send mail to the user on the originating machine. `-mu` *user-name* causes CXbatch to send to *user-name*. If the `-me` option is not given, CXbatch does not send mail when a batch request has been processed.

Controlling a Request

This section describes how to put a batch request on hold and remove it from a hold state.

Putting a Request on Hold

The `hold request` command within `qmgr` puts a request on hold, preventing it from executing. Only the owner, an operator, or a system manager can place a hold on a request. If a CXbatch operator or manager places a hold on a job, only an operator or a manager can release it.

Releasing a Request From a Hold

The `release request` command within `qmgr` releases a request previously placed on hold, making it eligible to run. If a CXbatch operator or manager places a hold on a job, only an operator or a manager can release it.

Deleting a Request from a Queue

Two commands delete a request from a batch or pipe queue:

- `qdel` — deletes a request from a queue. It is issued by the submitter, a batch manager, a batch operator, or superuser. Usually, the command applies only to jobs that are in a queue and have not started running; the `-k` and `-signo` qualifiers can be used to kill a running request.
- `delete request` — within `qmgr`, deletes either a running or a queued request. It must be issued by a batch manager, operator, or superuser. If the request is running, the command automatically kills it.

Each batch request has a unique identification assigned to it, and this ID is used in the delete process.

For more information on the `qdel` command, refer to the qdel(1) man page, the *CONVEX CXbatch System Manager's Guide*, the *CONVEX CXbatch User's Guide*, or the *CONVEX CXbatch System Manager Utilities Reference*.

Displaying Queue Status and Limits

You can display the resource limits currently enforced in CXbatch by using the `qlimit` command. You can display the current status of each CXbatch queue by using the `qstat` command or the `show` commands within `qmgr`. For more information, refer to the qlimit(1) and qstat(1) man pages.

Accounting

ConvexOS accounting may provide all the information you require. However, ConvexOS accounting cannot log information unique to CXbatch, such as job and queue priorities. CXbatch accounting handles these specific areas.

ConvexOS

ConvexOS accounting procedures maintain activity identification (AID) and resource information for each ConvexOS process in the default file /usr/adm/acct (this file name can be changed by the system manager). ConvexOS accounting is activated using the command /etc/accton.

System managers may use the CXbatch `qsa` command to access, process, and report the accounting records of the entire batch system, specific users, specific queues, and specific users in specific queues. Constraints to `qsa` allow variations on requested information. Please refer to Chapter 3, "Accounting," of the *CXbatch System Manager's Guide* for complete information on accounting functionality.

The system manager first creates or edits the /etc/activities file to establish AID numbers for each activity. While there are no rules that prescribe the actual numbering scheme, incrementing these numbers by 10 or 100 makes adding later entries easier. The following list shows what the entries in a sample /etc/activities file might look like. In this example, the AIDs are incremented by 100.

```
overhead:0
design:100
develop:200
integration:300
document:400
tooldev:500
test:600
```

Each queue in CXbatch has an activity ID offset, a number that CXbatch adds to a job's activity ID before execution. The activity ID offset is typically an integer from one to nine, with zero reserved for jobs not submitted to a batch queue. The system manager defines these ID offsets using the `set activity_id_offset` command within `qmgr`.

The activity ID of a request is modified with the value set in the activity ID mask. This mask is usually the same value as the spacing between entries in the /etc/activities file. The batch manager establishes this mask value with the `set aid_mask` command within `qmgr`.

For more information on ConvexOS system accounting, refer to the *ConvexOS System Manager's Guide* and the activities(5), sa(8), acct(5), and bill(1) man pages.

CXbatch

CXbatch accounting can be activated and deactivated on a per-queue basis. When it is activated, CXbatch saves batch job information in an accounting log file (if one is specified) and, if requested, sends mail to the user detailing what resources have been used.

CXbatch accounting is activated with the `set accounting` command within `qmgr`. The accounting log file is specified with the `set acc_logfile` command.

Refer to Chapter 3, "Accounting," of the *CXbatch System Manager's Guide* for an illustration of the CXbatch accounting information structure. This structure is defined in /usr/include/batch-acct.h. For more information, refer to the batch-acct(5) man page.

Reporting Problems

A

This appendix introduces the CONVEX Technical Assistance Center (TAC) and the `contact` utility.

The `contact` utility is an online system for reporting problems to the TAC. To use it, enter **`contact`** at the system prompt and answer the questions as they appear on the screen.

This appendix describes:

- Prerequisites for using `contact`
- Tips for using `contact`
- The step-by-step process `contact` takes you through

Technical Assistance Center

The CONVEX Technical Assistance Center (TAC) is staffed by technical specialists who can address diverse questions and problems that arise in a supercomputing environment. If you have a hardware, software, or documentation question, contact the TAC. This group stands ready to solve such problems.

The `contact` Utility

The TAC recommends using the `contact` utility to report a hardware, software, or documentation problem. The `contact` utility is an interactive program that helps the TAC track reports and route them to the CONVEX personnel most qualified to fix a problem.

After you invoke `contact`, it prompts you for information about the problem. When you finish your report, `contact` electronically mails it to the TAC. The TAC notifies you within 48 hours that your report has been received.

To use `contact` requires:

- UNIX-to-UNIX Communication Protocol (UUCP) connection to the TAC
- Full path name of the program or utility in question
- Version number of the program or utility in question

UUCP Connection

Before using `contact`, ask your system administrator if your site has a UUCP connection to the TAC. A UUCP connection allows files to be copied from one UNIX-based system to another. The `uucp` (UNIX-to-UNIX copy) command relies on either a dial-up or hard-wired UUCP communication line.

Finding the Program Path Name

To determine the full path name of the program or utility in question, use the `which` command. Figure A-1 illustrates use of the `which` command to find the full path name of the loader (`ld`) utility.

Figure A-1
Using the which command

```
> which ld
/bin/ld
>
```

In this example, the full path name of the loader is /bin/ld.

If you use the C shell (`csh`), you can also use the `whence` command to find the program path name. The `whence` command works like `which`, but faster.

For more information on the `which` command, refer to the which(1) man page. You can also use the `info` online information system by entering **`info which`** at the system prompt.

Finding the Program Version Number

To determine the version number of the program or utility in question, use the `vers` command. Figure A-7 illustrates use of the `vers` command to find the version number of the loader (`ld`) utility. Enter **`vers`**, then the path name of the program or utility.

Figure A-2
Using the `vers` command

```
> vers /bin/ld
/bin/ld: 7.0
>
```

In this example, the loader version number is 7.0.

For more information on the `vers` command, refer to the vers(1) man page. You can also use the `info` online information system by entering **`info vers`** at the system prompt.

Using contact

The contact utility prompts for the following information:

- Your name, title, phone number, and corporate name
- Name and version of the product
- One-line summary of the problem
- Detailed description of the problem
- Priority of the problem
- Instructions on how to reproduce the problem
- Comments about the problem
- Comments about the documentation relating to the problem
- Files to include in the contact report

Following is a step-by-step discussion of these prompts.

Step 1a

To invoke the contact utility, enter **contact** at the system prompt. The system responds with a welcome message and a series of questions regarding your hardware, software, or documentation question. Figure A-3 illustrates use of the contact command and the resulting system response.

Figure A-3 Beginning a contact session

```
> contact
Welcome to contact version 0.11 ()

Enter your name, title, phone number, and corporate name (^D to terminate)
>
```

Step 1b

If there is a .contact file in your home directory, contact skips the first prompt. (Refer to "Using a .contact File" on page 6 for more information.) Figure A-4 illustrates the contact command and the system response when you have a .contact file in your home directory.

Figure A-4 Beginning a contact session with a .contact file

```
> contact
Welcome to contact version 0.11 ()

Enter the name of the product involved
>
```

Step 2

The `contact` utility prompts for the version number of the product. If you do not know the version number, press **CTRL-Z** to suspend the session.

Use the `which` (or `whence` if you use `csh`) and `vers` commands to find the version number of the product. Use the `fg` command to return to the session, and enter the version number in the form X.X or X.X.X.X.

Step 3

The `contact` utility prompts for a one-line summary of the problem. This summary is the subject header in any further correspondence regarding the problem. Please make this summary as descriptive as possible in one line.

Step 4

The `contact` utility prompts for a detailed description of the problem. Please make this description as complete as possible. Include source code and a stack backtrace when possible. (Refer to the adb(1) or csd(1) man page for information on obtaining a stack backtrace.) The more information you provide, the quicker the TAC can isolate and solve the problem.

Step 5

The `contact` utility prompts for the priority of the problem. Figure A-5 illustrates this prompt and priority levels from which to choose. You must enter a priority number.

Figure A-5 Specifying the priority of a problem

```
Enter a problem priority, based on the following:
1) Critical    - work cannot proceed until the problem is resolved.
2) Serious     - work can proceed around the problem, with difficulty.
3) Necessary   - problem has to be fixed.
4) Annoying    - problem is bothersome.
5) Enhancement - requested enhancement.
6) Informative - for informational purposes only.
>
```

Step 6

The `contact` utility prompts for an explanation of how to reproduce the problem. Please include the command syntax and options you used and anything else you did to make the program run.

Step 7

The `contact` utility prompts for any other pertinent comments. Please include all relevant information.

Step 8

The `contact` utility prompts for suggestions regarding documentation supporting the product. Indicate whether the documentation could be revised to address the problem.

Step 9

The `contact` utility prompts for names of files necessary to reproduce the problem. Figure A-6 illustrates this prompt and sample user response.

Figure A-6 Including files in a `contact` report

```
Are there any files that should be included in this report (yes | no)?
> yes
Please enter the names of the files, one to a line (^D to terminate)
> test.f
> ~/subroutines/sub.f
>
```

Note

"Tilde-Escape Sequences" on page 7 are not recognized in responses to this prompt. In `contact`, a tilde in this section indicates your home directory. This convention is based on use of the tilde for expanding file names in `csh`.

If files specified are small text files, they are automatically included in the `contact` report. If the files are too large to be included in this report, `contact` gives further instructions on how to submit these files.

To specify a directory, combine directory files into a single file using the `tar` command (refer to the tar(1) man page for further information) or enter each file name in the directory on a single line in the `contact` report.

Step 10

The `contact` utility prompts you to review, edit, submit, or abort the report. Figure A-7 illustrates this prompt.

Figure A-7 Prompt to review, edit, submit, or abort report

```
Please select one of the following options:
1) Review the problem report.
2) Edit the problem report.
3) Submit the problem report.
4) Abort the problem report.
>
```

Choose the number of the option you want to select. These options let you do the following:

Review	review the text of the `contact` report. You are then prompted again to select an option.
Edit	edit the text of the `contact` report. If you choose to edit the report, `contact` opens your default text editor.
Submit	sends the report to the CONVEX TAC. The TAC notifies you within 48 hours that your report has been received. Choosing this option exits the `contact` utility and returns you to the shell.
Abort	saves the text of the report in a file named ~/dead.report. Choosing this option exits `contact` and returns you to the shell.

Tips for Using `contact`

The `contact` utility is interactive and easy to use. This section lists tips to help use it efficiently. In particular, this section tells how to

- Use a .contact file
- Abort a `contact` session
- Resubmit an aborted report
- Suspend a `contact` session
- Move within `contact` from one prompt to another
- Use tilde-escape sequences in the `contact` utility

Using a .contact File

When you invoke `contact`, it first prompts for your name, title, phone number, and company name. You can, however, create a .contact file to skip this first prompt.

Follow these steps to create a .contact file.

1. Create a .contact file in your home directory.
2. Enter your name, job title, phone number, and company name, each on a new line.

When you invoke `contact`, it automatically includes the .contact file as input for the first prompt and proceeds to the next prompt.

Aborting the Report

To abort a `contact` report, either press the interrupt key (usually **CTRL-C**) or choose the `abort` option when prompted by the `contact` utility. Using **CTRL-C** to abort does not save the contents of the report. Using the `abort` option saves the contents of the report in a file named ~/dead.report.

Submitting the dead.report File

After you abort a `contact` session, the `contact` utility saves the report in a file named ~/dead.report. Using the `contact` command with the `-r` option automatically merges the contents of the ~/dead.report file into the new `contact` session. Enter

```
contact -r
```

and `contact` finds the ~/dead.report file and merges it into the `contact` report. You can then edit the report. When you end the editing session, `contact` resumes at the final prompt, which asks you to review, edit, submit, or abort the report.

Suspending a Report

Sometimes it is necessary to stop in the middle of a `contact` report and return to the shell (for instance, to suspend the `contact` session to find the program path name or version number). To suspend the `contact` session, press **CTRL-Z**.

To return to the `contact` session, press **`fg`**. Using **CTRL-Z** and the `fg` (foreground) command, you can toggle back and forth between the `contact` utility and the shell. You cannot, however, use **CTRL-Z** and `fg` to toggle back and forth in the Bourne shell (`sh`).

Ending a Response

The `contact` utility prompts for information pertinent to your hardware, software, or documentation question. Some prompts require one-line responses; to move to the next prompt, press **RETURN**. Other prompts require more than a one-line response; to move to the next prompt, press **CTRL-D**.

Tilde-Escape Sequences

The `contact` utility treats input beginning with a tilde (~) as a special sequence. The character following the tilde is considered a request for a special function. You can use the following tilde sequences within `contact`:

~e	start the text editor (defined in the EDITOR environment variable)
~h	display a list of available tilde-escape sequences
~p	print the `contact` report to the terminal screen
~r *filename*	read the contents of *filename* as a response to the current prompt. Some prompts require only a one-line response. This tilde-escape sequence works only for prompts that allow more than a one-line response.
~~	insert a single tilde as the first character in the line

Index

V

W