CONVEX CXbatch V2.1 Release Notice

Document No. 710-007830-006

October 1991

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX
CXbatch Release Notice

Document No. 710-007830-006

Printed in the United States of America.

Table of Contents

CONVEX CXbatch Release Notice, V2.1

Overview

1

About This Package

CXbatch allows users to submit non-interactive jobs for batch execution on either the local machine or a remote machine in the CXbatch network. CXbatch provides a great deal of flexibility in queue configuration and management. Queues can be configured to enforce resource limits on requests, allow access only to selected groups and users, and route requests to lightly loaded machines.

CXbatch is based on NASA's Network Queueing System (NQS). Many CONVEX enhancements have been added, including checkpoint/restart, Share Scheduler integration, load balancing, batch accounting, automatic importing of files (via NFS), and direct submission to remote machines.

Documentation

The CXbatch Documentation Set included with this release contains the following documents:

- *Convex CXbatch Release Notice*
- *Convex CXbatch Installation Procedure*

Prerequisites

Before you can install this package, your system must already be running these software packages:

- ConvexOS V9.1 or a later release of the operating system.
- ConvexOS V9.1 Utilities or a later release of the system utilities.

If your system is not running these software releases, you must install them before continuing. If you need additional information on ConvexOS releases, contact the CONVEX Technical Assistance Center (TAC).

Corequisites

CXbatch V2.1 will operate with a number of optional CONVEX software products. However, you must have the correct version of those packages in order to guarantee proper operation of all software.

CXbatch V2.1 will operate with the following software packages:

- ConvexOS V9.1 Internet Services or a later release if you want to use the load balancing option, the automatic NFS import option or the COVUE products.
- ConvexOS V9.1 NFS Utilities or a later release if you wish to use the automatic NFS import option or the COVUE products.
- ConvexOS V9.1Share Scheduler or a later release if you wish to use the Share Scheduler integration features.

- ❑ COVUEbatch V2.1 or a later release if you require COVUEbatch functionality.
- ❑ COVUEshell V8.2 or a later release if you require COVUEshell submission capability.

If you install CXbatch V2.1 and you are running older versions of any of the listed packages, please contact your CONVEX sales representative for information on upgrading your software.

Disk Space Requirements

CXbatch V2.1 requires 6.5 megabytes of disk space in the /usr filesystem. If you are upgrading from a previous release of CXbatch, most of the disk space use by the previous installation is freed before the installation checks available disk space. In this case, you should have at least 2 megabytes of free space in /usr before you attempt to install this release.

Installing This Release

Refer to *CONVEX CXbatch Installation Procedures* for special instructions on how to install the software. Refer to the *Impacts* section of this document for a list of changes that will impact users, operators, and managers of CXbatch V2.1.

New Features in CXbatch V2.1

This release of CXbatch has a number of new features including

- ❑ Support for checkpoint/restart
- ❑ Better integration with the Convex Share Scheduler
- ❑ Improved batch accounting, management and monitoring

Many bug fixes are also included in this release. Refer to the *Software Problems and Fixes* section of this document for more details. This section summarizes the new features of CXbatch V2.1.

Checkpoint/restart

A new queue attribute has been added which determines whether or not open regular files are copy to/from the checkpoint directory on checkpoint/restart. This queue attribute is called `copy_open_files`. This attribute is set with the `SEt COPy_open_files` command.

ConvexOS/Secure

CXbatch V2.1 provides interoperability with ConvexOS/Secure V10.0. Additionally, when CXbatch V2.1 is run under ConvexOS/Secure, CXbatch provides auditing and authorization features. For more information, refer to the ConvexOS/Secure V10.0 release notice.

Notes and Cautions

This section contains useful information and words of caution about this release of CXbatch.

Share Scheduler

If the CONVEX Share Scheduler is installed on your system, an appropriate share policy should be set for each queue. The default share policy is `Fixed=<queuename>`. If the default queue configuration is to be installed from the release tape, the pseudo-users *short, long* and *verylong* must exist on your system and have been assigned shares **before** the installation is started. This is normally done using the `nu`(8) utility.

Unsupported Limits

The NQS implementation supports a number of resource limits. CXbatch can only enforce the resource limits supported by the underlying ConvexOS operating system. The actual limits supported by CXbatch, or any other NQS batch implementation, is reported by the `qlimit`(1) command or by the `qmgr`(8) `show limits_supported` command. All the standard NQS resource limits are accepted by the CXbatch `qsub`(1) and will be passed by CXbatch to the destination queue. If a requested resource limit is supported by the destination system, it will be enforced at run time. Otherwise, it will be silently ignored.

Some resource limits in CXbatch are supported only to the extent possible. For example, cpu time limits may be entered down to the milliseconds with `qsub`(1), but because ConvexOS only supports granularity in seconds, milliseconds are ignored.

Background Processes

When a request is selected to run, a shepherd process starts it by executing a top level shell which runs the submitted command script. When this top level shell process exits, the request is considered completed and the shepherd process cleans up by killing any remaining child processes started by the top level shell. This means that requests should not run any background commands without waiting for them to complete using the applicable shell commands.

Note

One side effect of this behavior has been found when using the `mail`(1) or `binmail`(1) commands from within a CXbatch request. If the addressee of a mail message is on the local (execution) system, a sub-process is forked by mail to deliver the mail message and the mail command itself exits. If the CXbatch request completes before the mail has been delivered, the delivery process may be killed by the shepherd process. This causes the mail to disappear without a trace. The workaround for this situation is the use of the -v option to `mail`(1) or sleep for a short period of time to allow the mail to be delivered.

Login Shells

By default, CXbatch runs a job using a non-login shell. This can be overridden with the `-l` option to `qsub`(1). If your job runs under a login shell, the /etc/login (C shell) or /etc/profile (Bourne shell and KornShell) and your `.login` (C shell) or .profile (Bourne shell and KornShell) will be read. A job running under the C shell will also read /etc/logout and .logout.

The string **ENVIRONMENT=BATCH** is added to the environment of the batch request. With this variable, shell start-up scripts (such as .profile, .login and .cshrc) can test for batch request execution and not, for example, perform any setting of terminal characteristics. The C shell always reads your .cshrc file regardless of whether it is running as a login shell, so these precautions should be taken.

When running jobs under the C shell, if a login shell is requested using the `-l` option of `qsub`(1), the message "Warning: no access to tty..." will be included in the job's output file. There is no way CXbatch can suppress this message; it would require a change to the C shell.

Load Balancing Pipe Client

The CXbatch load balancing pipe client, `pipeldav`, works with CXbatch pipe queue destinations, but not with non-CONVEX NQS destinations.

COVUEbatch and COVUEshell

When COVUE is used with CXbatch, certain COVUE commands cannot obtain queue and job information from pipe queue destinations that are running non-CONVEX NQS. There should not be any problem obtaining information with CXbatch destination queues.

CXbatch V2.1 no longer contains `queued`(8), a daemon previously needed by COVUEbatch and COVUEshell to obtain queue information. If COVUEbatch or COVUEshell are to be used with this release of CXbatch, COVUEbatch V2.1 or later, and COVUEshell V8.2 or later must be used.

Unsupported NQS Features

Although CXbatch is based on NQS, some NQS feature not supported include:

- Device queues
- Queue complexes
- `Qmgr`(8) `set global run_limit` command

Import_dir

When the `import_dir` attribute is set for a job request that is to run on a remote machine, the current working directory of the request is made available to the remote system through the use on an NFS mount. These mounts are made onto temporary directories in the /tmp filesystem. System administrators should pay particular attention to any automatic clean-up procedures for the /tmp filesystem to assure that such procedures do not traverse NFS mount points.

The current implementation of CXbatch cannot import a current working directory that is already within an NFS filesystem. Requests which attempt to do this will fail.

Disk Space

Measures should be taken to insure that the filesystem containing /usr/spool/nqs has a reasonable amount of free disk space during regular queue processing. Running out of disk space can cause job status and output to be lost if output files cannot be copied back to a request's current working directory due to disk space problems. Mail notification can also be lost due to the inability to write into /usr/spool/mail due to disk space problems.

System Management

While every effort is made to ensure that batch jobs are controlled and cannot disrupt the operation of the system, just as in any user environment, a persistent user may be able to circumvent some of the controlling mechanisms. System administrators have the responsibility to monitor and prevent system abuse through the use of online protection mechanisms (i.e. disk quotas, resources limits, etc.) and offline human management techniques.

Checkpoint/Restart

CXbatch requests submitted to queues with the checkpoint queue attribute set to `AVAILABLE` are no longer automatically checkpointed on CXbatch shutdown. Such requests may be checkpointed manually and they will be checkpointed automatically if they were submitted with the `qsub -c` option.

Software Problems and Fixes

This section contains a list of software bug reports. The list is divided into known problems and problems that have been fixed since the last release

Known Problems

This section lists the know problems with CXbatch V2.1 as of October 15, 1991. Problems reported after this date are not reflected in this document. Please refer to this list before reporting a problem.

qmgr (PR-09971)

There should be `qmgr` commands that reset the accounting file, and maybe a command that resets the current sequence number.

qdel (PR-09988)

`qdel` needs an option to remove "all my jobs" like '`remove -`'.

qmgr (PR-09995)

It would be nice to have an auto command completion feature in `qmgr`.

netdaemon (PR-10095)

The netdaemon will report that a request's output file was left in the user's home directory on the execution machine if it was unable to return the output file to the primary destination. The mail notifying the user of this fact should include the name of the execution machine.

logfile (PR-11078)

It is desirable to be able to turn on a switch with CXbatch that will cause the commands issued to be printed in the logfile prior to the output from that command. This is very useful when debugging batch command scripts.

doc (PR-11422)

A summary for the syntax of the various `qmgr` commands (the top part of each man page) would be helpful as a quick reference guide instead of flipping pages. Also a cross reference in each man page to its opposite command would be helpful (if it has an opposite).

start/stop batch (PR-11810)

It would be nice to say start queue all or some such thing and have it start them all. The same should work for stop.

qmgr (PR-11954, PR-12109)

Need a `qmgr` command showing the status of running batch requests such as starting time, CPU-time consumed, and name of the actual running program. I don't see a way to find out the starting time of a batch request and with `ps -x` you can only see the CPU-time used by the now running process in the batch request. This feature would also be important for batch operators when they have to decide to shut down the batch system they should have the possibility see how long a job has been running.

CXbatch load average (PR-12009)

In the old batch system the system administrator could operate the queues depending on the load average (th-flag in /etc/batchcap). This is no longer possible with CXbatch. They would have to use pipeldav (with `rwhod`). Not all customers use `rwhod`.

import (PR-12156)

If the working directory of a request is in an NFS filesystem, and the request is submitted to a remote queue with the `import_dir` attribute set to yes, the request will fail because nqsdaemon is unable to import the current working directory.

qsub (PR-12487)

It would be nice if it was possible to specify a dependency between jobs that are submitted to the same queue. This most useful if the queue has a width that is greater than 1.

qmgr (PR-12984)

We would like to be able to do the following two functions.

1- A submitted job should be able to get the name of the queue it is running from. A suitable mechanism may be the setting of an environment variable QSUB_QUEUE.

2- A utility like `qstat` should be able to get information on `request-id`. For our purposes, it should at least return the queue in which the job is running and the name of the request, but other attributes may be useful as well.

qsub (PR-13249)

It would be useful to be able to pass parameters to submitted batch jobs. For example: `qsub -av arg1 arg2 arg3`

qsub (PR-13531)

I would like an option to batch queues so that jobs are run in order of shortest to longest. If I had this capability, I could probably replace 2 or 3 queues with one. The only purpose is to insure that the shorter jobs get run before the long ones.

qdel (PR-13814)

A batch job can be deleted during execution by the `qdel -k` command, or the COVUE DELETE/ENTRY command on the VAX. If this is done, the batch account record for the job does not include the resources consumed by the process that was active when the delete signal was received. For a site with very long jobs (which is typical for Convex customers), that charges its users based on the information found in the batch accounting file (/usr/adm/batch-acct) this can mean losing a lot of money.

qsub (PR-13893)

If a user with "/bin/false" as the login shell attempts to run a job in the CXbatch system, it will stop the queue. This problem occurs when using COVUEnet 2.0 and COVUEbatch 2.0. COVUEnet requires the default access account to be created with /bin/false as a the login shell. This account may be used for COVUEbatch requests connecting by the default account. If the shell strategy is not fixed, CXbatch has trouble.

nqsdaemon (PR-10095)

The CXbatch remote file staging failure message should state the name of the execution machine.

nqsdaemon (PR-14022)

Jobs submitted from non-CONVEX NQS clients cannot use "-l" as an embedded option when submitted to CXbatch.

qsub (PR-16405)

The `qsub -mu` option does not allow more than 15 characters in a mail address.

CXbatch (PR-17240)

When a process executing in a CXbatch queue exceeds the per-process CPU limit, it is sent a SIGXCPU(24). Processes may ignore this signal and circumvent the limit.

CXbatch (PR-12984)

CXbatch requests should be able to find out the name of the queue they are running in. qstat should accept request ids to get info about CXbatch requests.

qsub.1 (PR-20792)

The `qsub`(1) man page needs to clarify that a batch request is only run in a login shell if the -l option is specified with qsub. Otherwise initialization scripts (/etc/login, .login, etc...) are not executed at the beginning of a batch request.

qsub (PR-14022)

The `qsub -l` flag cannot be transmitted to CXbatch if the request comes to CXbatch through a pipe queue from a non-CXbatch NQS client.

CXbatch (PR-11951 PR-6615 PR-14476 PR-7249 PR-16300 PR-8278 PR-7575)

Per request resource limits would be useful.

CXbatch (PR-12984)

When a job is submitted with `qsub -b` group.activity, the group used for billing is not recorded for accounting purposes. The gid that is charged in accounting is the user's primary group from the password file.

nqsdaemon (PR-20603)

Currently, if a job is submitted as non-restartable (`-nr`), but it is also checkpointable (`-c`), the `-nr` takes precedence and the job can never be restarted. Request that these two options be decoupled such that a checkpointed job may be restarted, but a non-checkpointed job will not be restarted.

qdel (PR-20800)

When deleting a job from a remote queue (`qdel reqid@hostname`), `qdel` will never return the proper error code. In case of an error, it will always say: Unexpected completion code from CXbatch daemon.

qsub (PR-21491)

`qsub`(1) cannot be run from NFS mounted directories whose parent(s) are not world readable (and root is not granted privilege on the remote server).

qstat (PR-21051)

If you are not the owner of a suspended batch job and do a qstat queue, the suspended job will show up as holding. If you do a `qstat -a` it will show up as suspended. Both variants should show suspended.

qmapmgr (PR-11228 PR-16951 PR-17355)

Occasionally, aliases added to the `qmapmgr` database will not show up when a listing is made with the `show` command.

Fixed Problems

This section lists the fixes that have been made to CXbatch since the V2.0 release.

nqsdaemon (PR-17060 PR-17050)

Problem: CXbatch checkpointing does not work if an mid is greater than 99999.

Resolution: CXbatch now correctly handles mids which exceed 99999.

smg (PR-17907)

Problem: In the *CXbatch System Manager's Guide* page 5-3 there is a reference to the "set share_policy user" command, it should reference the command "set share_policy fixed".

Resolution: Fixed in *CXbatch System Manager's Guide*, 3rd edition.

smur (PR-17907 PR-18091)

Problem: In the *CXbatch System Manager's Utilities Reference* page 2-39, the explanations for "Parameters:" reference parameters for import_dir not set chkpntable. On page 2-64, the "Parameters" sections references "username" it should be "queuename". On page 5-4, there is a section describing the way to specify user names. This needs to be removed.

Resolution: Fixed in *CXbatch System Manager's Guide,* 3rd edition.

smur (PR-17907)

Problem: In the *CXbatch System Manager's Utilities Reference* page 2-64, the "Parameters" sections references "username" it should be "queuename".

Resolution: Fixed in *CXbatch System Manager's Guide,* 3rd edition.

qsub.1 (PR-17896)

Problem: 'Suppressed' is misspelled.

Resolution: The typo has been corrected.

logdaemon (PR-19585 PR-18375)

Problem: logdaemon dumps core when attempting to send mail to batch managers on notification of a fatal error.

Resolution: logdaemon now correctly delivers mail messages on fatal errors.

nqsdaemon (PR-18351)

Problem: CXbatch continues to attempt to checkpoint jobs which have been submitted through a pipe queue with periodic checkpointing enabled, after they have been removed.

Resolution: CXbatch now stops periodic checkpointing when such a job is removed.

qsub (PR-19929)

Problem: The summary report produced by the `qsub -y` option incorrectly uses the submission time of a job as the start time.

Resolution: The summary report now correctly states the start time of the job.

qmgr (PR-13306, PR-19575)

Problem: `qmgr` does not document how limits should be set to "UNLIMITED". The all upper-case value is displayed, but only "unlimited" is accepted.

Resolution: `qmgr` now accepts "UNLIMITED" or "unlimited" as limit values. In addition, the qmgr(8) man page gives instruction on setting limits to this value.

nqsdaemon (PR-14105 PR-21619)

Problem: If a CXbatch queue is attempting to deliver a request to a machine which is unreachable when the local machine goes down and the next retry time passes before the local machine is rebooted, CXbatch will never try again to reach the remote machine.

Resolution: CXbatch now retries pipe queue destinations on startup. It will then reset the destination retry time to the default if the destination is unreachable.

qmgr (PR-14733)

Problem: `qmgr` should allow users to suspend and resume their own jobs.

Resolution: Users may now suspend and resume their own jobs.

nqsdaemon (PR-16926)

Problem: It would be desirable to be able to start CXbatch with all queues stopped and or disabled. This would allow the system manager to make changes to the system with no jobs running and or no submissions occurring upon startup.

Resolution: Two qualifiers have been added to the `STArt Cxbatch` command. `Disabled` will cause all queue to be disabled on CXbatch startup. `Stopped` will cause all queues to be stopped on CXbatch startup.

nqsdaemon (PR-15966)

Problem: CXbatch should give the name of the queue the request was executing in when sending mail about CXbatch requests.

Resolution: CXbatch now includes the queue name in mail about CXBatch requests.

qmgr (PR-19201)

Problem: Occasionally, quitting the `$PAGER` during qmgr operation will cause qmgr to terminate.

Resolution: `qmgr` no longer terminates when `$PAGER` is quit early.

nqsdaemon (PR-15802)

Problem: CXbatch should inform either the user or the CXbatch manager what the name of the output file is when the output can't be delivered to the user's primary or backup directory.

Resolution: The owner of a CXbatch request whose output could not be returned to the primary or backup directory is informed by mail that the output is held in the CXbatch output directory. A message is also logged that gives the name of the request and the location of the files.

qsub (PR-15341)

Problem: `qsub`(1) will dump core with illegal instruction when executed from a fortran program.

Resolution: `qsub` now operates correctly with ConvexOS V9.0 and later.

nqsdaemon (PR-16123)

Problem: CXbatch grace periods for shutdown and queue aborts do not allow values greater than 600 seconds.

Resolution: Users may now specify grace periods greater than 600 seconds.

CXbatch (PR-17654 PR-20590)

Problem: Request that CXbatch include a facility to copy open files on checkpoint/restart.

Resolution: CXbatch now offers this capability through the `copy_open_files` queue attribute.

nqsdaemon (PR-20908)

Problem: If CXbatch is set to perform periodic checkpointing on a job, and that job is suspended, CXbatch will still attempt to checkpoint the suspended job. When attempting to checkpoint the suspended job, CXbatch may instead checkpoint a running job, overwriting the files of the suspended job.

Resolution: CXbatch no longer attempts to checkpoint a job which has been suspended.

qdel (PR-15271)

Problem: When issuing a `qdel -15 id`, the job does not terminate.

Resolution: qdel sends the specified signal to the shell process of a CXbatch request. By default, shells which are attached to terminals ignore SIGTERM. Therefore, `qdel -15` will not stop a request unless SIGTERM is explicitly set to SIG_DFL.

smg (PR-15729)

Problem: An explanation of how to set up CXbatch without TCP/IP is needed.

Resolution: Added in *CXbatch System Manager's Guide*, 3rd edition.

qsub (PR-17211)

Problem: A If a job is submitted without the `-p` flag of `qsub` (i.e. with the default request priority) the request priority field in the accounting record will incorrectly be set to -1.

Resolution: If the `qsub -p` option is not used, CXbatch now records the correct default priority in the accounting record.

accounting (PR-17231)

Problem: Some parts of CXbatch use the `ru_exutime` field of the `rusage` structure whereas other sections use the `ru_utime` field. All parts of CXbatch should use the same field.

Resolution: CXbatch now consistently uses to `ru_utime` field of the resource structure.

install (PR-17783)

Problem: When the installation is re-configuring the existing queues it complains that it cannot complete rebuild of queues, when in fact the queues have been rebuilt properly. This occurs with CXbatch V2.0 and ConvexOS V9.1.

Resolution: This was caused by a bug in grep(1) from ConvexOS V9.1. The CXbatch V2.1 installation does not exhibit this problem with ConvexOS V9.1.

nqsdaemon (PR-18505)

Problem: The nqsdaemon man page mentions that the logdaemon send messages to syslogd, but does not describe the facilities and levels that it uses.

Resolution: More info on syslog(3) facilities and levels has been added to the nqsdaemon(8) man page.

qsub.1 (PR-20131)

Problem: The `qsub`(1) man page does not mention the "Minutes" unit under the description of the "`-cp`" option.

Resolution: `qsub.1` now contains mention of the Minutes unit in relation to the `-cp` option.

nqsdaemon (PR-20177)

Problem: CXbatch requests in the `HOLDING` state which are released using the `qmgr release request` command remain in the `QUEUED` state and are not run.

Resolution: CXbatch now attempts to run request which are released.

qmgr (PR-21588)

Problem: The `qmgr delete request` command will not accept multiple request ids.

Resolution: The command now accepts multiple request ids.

nqsdaemon (PR-15802)

Problem: A mechanism is needed which notifies the user or the batch manager what the name of the output file is when the output can't be delivered to the user's output directory.

Resolution: The file staging mail message now informs the user that the files are in the CXbatch output directory, and a message is logged to syslog which point directly to these files.

qmgr (PR-21414)

Problem: `qmgr`(1) displays the 'Mgr: 'prompt even when standard input is not connected to a tty.

Resolution: `qmgr` no longer prints a prompt if stdin is not connected to a tty.

qmgr (PR-21504)

Problem: `qmgr`(1) sets the invoking user's manager privileges at startup, so changes to this user's privileges are do not affect this initial setting.

Resolution: `qmgr` privileges are now updated on `set manager` and `delete manager` commands.

nqsdaemon (PR-20018)

Problem: The checkpoint queue attribute setting AVAILABLE does not differ from the setting YES.

Resolution: CXbatch requests running in queues with the checkpoint attribute=AVAILABLE are now only automatically checkpointed if they were submitted with the `qsub -c` flag.

nqsdaemon (PR-21487)

Problem: Under the `FIXED` share policy, CXbatch resets the usage field of the process lnode to 0, thus undermining the CONVEX Share Scheduler's attempts to parcel out CPU time equally.

Resolution: CXbatch no longer resets the usage field of the lnode before starting a CXbatch request under the `FIXED` share policy.

CXbatch Initiator

2

Overview

CONVEX has received several requests to implement alternative scheduling algorithms for CXbatch requests. Each request is usually accompanied by a description of the scheduling practices desired by a particular site. While it became evident that some improvement needed to be done in this area, no single scheduler will meet the needs of all customers.

To address this problem, CONVEX is investigating a concept we refer to as the `initiator` daemon. This daemon runs in conjunction with the `nqsdaemon` and is notified whenever the `nqsdaemon` is about to enter its job scheduling loop. The `initiator` can request information about queues and requests and can issue commands to modify the priority of queued requests based on this information and information obtained through other ConvexOS facilities.

The goal of this project is to provide a set of library routines and documentation that will allow customer sites to develop an initiator daemon that implements their own request scheduling algorithms. The interface would be designed so that an experienced C programmer could prototype and implement various scheduling algorithms.

For this release of CXbatch, CONVEX developed a sample initiator daemon for customer evaluation and feedback on the concept. We wish to determine if the initiator concept is the proper solution to the scheduler problem and if the interface would be used sufficiently to warrant its development.

initiator.share

The sample CXbatch `initiator` daemon is installed in /usr/lib/nqs/initiator share by the CXbatch V2.1 installation. The daemon is activated by renaming it /usr/lib/nqs/initiator and restarting CXbatch. Each time a CXbatch request exits, this daemon is notified and given the opportunity to reorder CXbatch local batch queues based on a scheduling algorithm.

The algorithm employed by the sample CXbatch initiator daemon takes into account both the relative share assigned to the owner of a request and the time a request has been waiting in the queue. A ranking is generated for requests in each queue, and intra-queue request priorities are set accordingly.

Relative priorities are generated by the following formula:

$$rank = share + \frac{ln(age)}{2}$$

where *share* is assigned percentage of machine
age is request age in seconds

Customers are encouraged to activate and observe the operation of the sample CXbatch Initiator daemon and provide suggestions to CONVEX about its usefulness or about alternative solutions.

CXbatch Man Pages

A

This appendix contains printed copies of the CXbatch V2.1 man pages. These are also available online using the *man*(1) utility.

NAME

qchkpnt – checkpoint CXbatch request(s).

SYNOPSIS

qchkpnt [**–e** *freq*] [**-f**] *request-id* **...**

DESCRIPTION

qchkpnt checkpoints the requests whose *request-ids* are listed on the command line. The current state of all the processes comprising the request are saved in a set of checkpoint files. The checkpoint files are stored in the CXbatch *checkpoint directory.* A successfully checkpointed request will be restarted from its checkpointed state instead of being re-run.

Only running batch requests may be checkpointed. To checkpoint a request, the invoking user must be the owner of the request or have CXbatch operator privileges. Under ConvexOS/Secure the user must have the **batch** subsystem authorization in addition to CXbatch operator privileges.

The options have the following meanings:

–e *freq* Normally the request is checkpointed immediately. When the **-e** option is present the request will be checkpointed periodically at intervals of *freq.*

The *freq* is specified as <[*number*] *unit*>, where *number* is a positive integer and *unit* is [*Hours* | *Days* | *Weeks*]. A *number* of zero will cancel periodic checkpointing.

-f Force checkpoint even if one of the processes of the request hold non-checkpointable resources.

DIAGNOSTICS

qchkpnt returns an exit status describing what it did. If there were no errors, the exit status is zero. If one or more of the requests weren't checkpointed, the exit status is the number of requests that weren't checkpointed. If a fatal error occurs and none of the requests are checkpointed (e.g., a syntax error), the exit status is one of the codes defined in <*sysexits.h*>.

EX_USAGE The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.

EX_OSFILE Some batch system file does not exist, cannot be opened, or has an error.

EX_TEMPFAIL Temporary failure; retry the command at a later time.

EX_NOPERM You did not have sufficient permission to perform the operation.

EX_SOFTWARE Too many request-ids were specified.

SEE ALSO

qsub(1), qrestart(1), chkpnt(1), restart(1), qmgr(8)

NOTES

CXbatch is an optional product; for more information, please contact your CONVEX sales representative.

NAME

qdel – delete or signal CXbatch request(s).

SYNOPSIS

qdel [**–k**] [*–signo*] [**–u** *username*] *request-id*[@ *host*] ...

DESCRIPTION

qdel deletes all queued CXbatch requests whose *request-ids* are listed on the command line. Additionally, if the flag **-k** is specified, the default signal of SIGKILL (9) is sent to any running request whose *request-id* is listed on the command line. This causes the receiving request to exit and be deleted. If the flag *–signo* is present, the specified signal is sent instead of the SIGKILL signal to any running request whose *request-id* is listed on the command line. *signo* can be either the signal number or the signal name. The signal names are as given in */usr/include/signal.h*, stripped of the common SIG prefix. In the absence of the **-k** and *–signo* flags, *qdel* will not delete a *running* CXbatch request.

To delete or signal a CXbatch request, the invoking user must be the owner (the submitter of the request) or the superuser. The only exception to this rule occurs when the invoking user has CXbatch operator privileges as defined in the CXbatch manager database. Under ConvexOS/Secure the user must have the **batch** subsystem authorization in addition to CXbatch operator privileges. Under these conditions, the invoker may specify the **–u** *username* flag that allows the invoker to delete or signal requests owned by the user whose account name is *username*. When this form of the command is used, all *request-ids* listed on the command line are presumed to refer to requests owned by the specified user.

A CXbatch request is always uniquely identified by its *request-id*, no matter where it is in the network of the machines. A *request-id* is always of the form *seqno* or *seqno.hostname*, where *hostname* identifies the machine from whence the request was originally submitted, and *seqno* identifies the sequence number assigned to the request on the originating host. If the *hostname* portion of a *request-id* is omitted, the local host is always assumed. The local host is searched for each given *request-id*, unless a different host is specified with @ *host*.

The *request-id* of any CXbatch request is displayed when the request is first submitted (unless the *silent* mode of operation for the given CXbatch command was specified). The user can also obtain the *request-id* of any request through the use of the *qstat*(1) command.

DIAGNOSTICS

qdel returns an exit status describing what it did. If there were no errors, the exit status is zero. If one or more of the requests were not deleted, the exit status is the number of requests that weren't deleted. If a fatal error occurs and none of the requests are deleted (e.g., a syntax error), the exit status is one of the codes defined in *<sysexits.h>*.

EX_USAGE	The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.
EX_NOUSER	The user specified with *-u* did not exist.
EX_NOHOST	The host specified did not exist.
EX_OSFILE	Some batch system file does not exist, cannot be opened, or has an error.
EX_TEMPFAIL	Temporary failure; retry the request at a later time.
EX_NOPERM	You did not have sufficient permission to perform the operation.
EX_SOFTWARE	Too many request-ids were specified.

CAVEATS

When a CXbatch request is signaled by the methods discussed above, the proper signal is sent to *all* processes comprising the CXbatch *request* that are in the same *process group*. Whenever a CXbatch request is spawned, a new *process group* is established for all processes in the request.

However, should one or more processes of the request successfully execute a *setpgrp*() system call, such processes will *not* receive any signals sent by the *qdel*(1) command. This can lead to "rogue" request processes that must be killed by other means such as the *kill*(1) command.

SEE ALSO

qjlist(1), qlimit(1), qstat(1), qsub(1), qmgr(8), kill(2), setpgrp(2), signal(3c)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qjlist – list the commands in a batch job.

SYNOPSIS

qjlist *request-id*[@*host*] ...

DESCRIPTION

qjlist lists the commands in each CXbatch request whose *request-ids* are listed on the command line. To list the commands in a CXbatch request, the invoking user must be the owner (the submitter of the request). The only exception to this rule occurs when the invoking user is the *superuser* or has CXbatch operator privileges as defined in the CXbatch manager database. Under ConvexOS/Secure the user must have the **batch** subsystem authorization in addition to CXbatch operator privileges. Under these conditions, the invoker may specify any batch request.

A CXbatch request is always uniquely identified by its *request-id*. A *request-id* is always of the form *seqno* or *seqno.hostname*, where *hostname* identifies the machine from whence the request was originally submitted, and *seqno* identifies the sequence number assigned to the request on the originating host. If the *hostname* portion of a *request-id* is omitted, the local host is always assumed. The local host is searched for each given *request-id* unless a different host is specified with @*host*.

The *request-id* of any CXbatch request is displayed when the request is first submitted (unless the *silent* mode of operation for the given CXbatch command was specified). The user can also obtain the *request-id* of any request through the use of the *qstat*(1) command.

qjlist returns an exit status describing what it did. If there were no errors, the exit status is zero. If one or more of the requests were not listed, the exit status is the number of requests that weren't listed. If a fatal error occurs and none of the requests are listed (e.g., a syntax error), the exit status is one of the codes defined in *<sysexits.h>*.

EX_USAGE	The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, or bad syntax in a parameter.
EX_NOHOST	The host specified did not exist.
EX_OSFILE	Some batch system file does not exist, cannot be opened, or has an error.
EX_NOPERM	You did not have sufficient permission to perform the operation.

SEE ALSO

qdel(1), qlimit(1), qstat(1), qsub(1), qmgr(8)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qlimit – show supported batch limits, and shell strategy for the named host(s).

SYNOPSIS

qlimit [*host-name ...*]

DESCRIPTION

qlimit displays the set of batch request resource limit types that can be directly enforced on the implied local host or named hosts and also the *batch request shell strategy* defined for the implied local host or named hosts.

If no *host-names* are given, the information displayed is relevant to only the local host. Otherwise, the supported batch request limits, and *batch request shell strategy* for each of the named hosts is displayed.

CXbatch supports many batch request resource limit types that can be applied to a batch request. However, not all UNIX implementations are capable of supporting the rather extensive set of limit types that CXbatch provides.

The set of limits applied to a batch request is always restricted to the set of limits that can be directly supported by the underlying UNIX implementation. If a batch request specifies a limit that cannot be enforced by the underlying UNIX implementation, the limit is ignored, and the batch request operates as though there were no limit (other than the obvious physical maximums) placed upon that resource type.

When an attempt is made to queue a batch request, each *limit-value* specified by the request (that can also be supported by the local UNIX implementation) is compared against the corresponding *limit-value* configured for the destination batch queue. If the corresponding batch queue *limit-value* for all batch request *limit-values* is defined as unlimited, or is greater than or equal to the corresponding batch request *limit-value*, the request can be successfully queued, provided that no other anomalous conditions occur. For request *infinity limit-values*, the corresponding queue *limit-value* must also be configured as infinity.

These resource limit checks are performed irrespective of the batch request arrival mechanism, either by a direct use of the *qsub*(1) command, or by the indirect placement of a batch request into a batch queue via a *pipe* queue. It is impossible for a batch request to be queued in a CXbatch batch queue if *any* of these resource limit checks fail.

Finally, if a request fails to specify a *limit-value* for a resource limit type supported on the execution machine, the corresponding *limit-value*, as configured for the destination queue, becomes the *limit-value* for the unspecified request limit.

Upon the successful queuing of a request in a batch queue, the set of limits under which the request will execute is frozen and will not be modified by subsequent *qmgr*(8) commands that alter the limits of the containing batch queue.

As mentioned above, this command also displays the *shell strategy* configured for the implied local host or named hosts. In the absence of a shell specification for a batch request, CXbatch must choose which shell should be used to execute that batch request. CXbatch supports three different algorithms, or *strategies*, to solve this problem that can be configured for each system by a system administrator, depending on the needs of the user's involved, and upon system performance criterion.

The three possible shell strategies are called:

- *fixed*
- *free*
- *login*

These shell strategies respectively cause the configured *fixed* shell to be exec'd to interpret all batch requests; cause the user's login shell as defined in the password file to be exec'd, which in turn chooses and spawns the appropriate shell for running the batch shell script; or cause only the user's login shell to be exec'd to interpret the script.

A shell strategy of *fixed* means that the same shell chosen by the system administrator is used to execute *all* batch requests (in the absence of a shell specification).

A shell strategy of *free* runs the batch request script *exactly* as would an interactive invocation of the script and is the default CXbatch shell strategy.

The strategies of *fixed* and *login* exist for host systems that are short on available free processes. In these two strategies, a single shell is exec'd, and that same shell is the shell that executes all of the commands in the batch request shell script.

When a shell strategy of *fixed* has been configured for a particular CXbatch system, the "fixed" shell that will be used to run *all* batch requests at that host is displayed.

DIAGNOSTICS

qlimit returns an exit status describing what it did. If there were no errors, the exit status is zero. If one or more of the hosts were invalid or couldn't be reached, the exit status is the number of hosts that were invalid or couldn't be reached. If a fatal error occurs, the exit status is one of the codes defined in *<sysexits.h>*.

EX_OSFILE Some batch system file does not exist, cannot be opened, or has an error.

SEE ALSO

qdel(1), qjlist(1), qstat(1), qsub(1), qmgr(8)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qps – display process status of CXbatch related processes

SYNOPSIS

qps [*queue-name ...*]
qps **–r** *request-id*
qps **–{p|q}** *process-id*

DESCRIPTION

qps prints information about CONVEX CXbatch related processes.

If no queues are specified, information is printed about all CXbatch related processes, including the daemon processes. Otherwise, information is printed for the specified queues only. Only batch queues on the local system are significant; pipe queues and remote queues do not have processes running on the local system.

Information about processes pertaining to a particular request can be obtained by using the **-r** *request-id* option. Only a single request-id can be specified.

For each process, *qps* prints the queue name (QUEUE), the request ID (REQ), the process ID (PID), the state (STAT) of the process, CPU time (TIME) used by the process (including both user and system time) and which command is running (COMMAND). More information about these fields can be found on the *ps(1)* man page.

Using the **-p** *process-id* option, you can inquire whether a particular process is running from within the CXbatch system. If it is, a line is printed stating the queue and request to which the process is related and *qps* exits with a status of 0. Otherwise, you are informed that the process is not running from within the CXbatch system and *qps* exits with a status of 1. For the purposes of this inquiry, the top level daemons are not considered to be running under the CXbatch system, but the CXbatch shepherd processes are.

The **-q** *process-id* option is a silent version of the **-p** option. Nothing is printed, but the exit status of *qps* is set appropriately. Only a single process-id can be specified for either of these options.

DIAGNOSTICS

qps returns an exit status of 0 if no errors occur, unless the the **-p** or **-q** options are specified, in which case the exit status is 0 or 1 as described above. If a fatal error occurs, the exit status is one of the codes defined in *<sysexits.h>*.

EX_USAGE	The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.
EX_OSERR	An internal call to *ps*(1) failed. If this error occurs, first make sure a '/bin/ps laxwg' returns valid output, then check the status of CXbatch.

SEE ALSO

ps(1), qstat(1)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qrestart – restart checkpointed CXbatch request(s).

SYNOPSIS

qrestart [**-f**] [*request-id*]

DESCRIPTION

qrestart restarts the requests whose *request-ids* are listed on the command line.

Only successfully checkpointed requests may be restarted. To restart a request, the invoking user must be the owner of the request or have CXbatch operator privileges. Under ConvexOS/Secure the user must have the **batch** subsystem authorization in addition to CXbatch operator privileges.

qrestart is typically only used if the automatic restart of the request by CXbatch failed.

The options have the following meanings:

-f Force restart of request even if non-restartable conditions exist in any of the processes of the request.

DIAGNOSTICS

qrestart returns an exit status describing what it did. If there were no errors, the exit status is zero. If one or more of the requests were not restarted, the exit status is the number of requests that weren't restarted. If a fatal error occurs and none of the requests are restarted (e.g., a syntax error), the exit status is one of the codes defined in *<sysexits.h>*.

EX_USAGE The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.

EX_OSFILE Some batch system file does not exist, cannot be opened, or has an error.

EX_TEMPFAIL Temporary failure; retry the command at a later time.

EX_SOFTWARE Too many request-ids were specified.

REFERENCES

qsub(1), qchkpnt(1), chkpnt(1), restart(1), qmgr(8)

NOTES

CXbatch is an optional product; for more information, please contact your CONVEX sales representative.

NAME

qstat – display status of CXbatch queue(s)

SYNOPSIS

qstat [**–a**] [**–l**] [**–m**] [**–u** *user-name*] [**–x**] [*queue-name* ...] [*queue-name@host-name* ...]

DESCRIPTION

qstat displays the status of CONVEX CXbatch queues.

If no queues are specified, the current state of each CXbatch queue on the local host is displayed. Otherwise, information is displayed for the specified queues only. Queues may be specified either as *queue-name* or *queue-name@host-name*. In the absence of a *host-name* specifier, the local host is assumed.

For each selected queue, *qstat* displays a *queue header* (information about the queue itself) followed by information about requests in the queue. Ordinarily, *qstat* shows only those requests belonging to the invoker. The following flags are available:

-a	Shows all requests.
-l	Requests are shown in a long format.
-m	Requests are shown in a medium-length format.
-u *user-name*	Shows only those requests belonging to *user-name*.
-x	The queue header is shown in an extended format.

The *queue header* always includes the queue-name, queue type, queue status (see below), an indication of whether or not the queue is *pipeonly* (accepts requests from pipe queues only), and the number of requests in the queue. An extended queue header also displays the priority and run limit of a queue, as well as the access restrictions, cumulative use statistics, server and destinations (if a pipe queue), and resource limits (if a batch queue).

By default, *qstat* displays the following information about a request: the *request-name*, the *request-id*, the owner, the relative request priority, and the current request state (see below). For running requests, the process group is also shown, as soon as it becomes available to the local CXbatch daemon.

qstat -m shows the following additional information: if the request was submitted with the constraint that it not run before a certain time and date, the constraining time and date are also displayed.

qstat -l shows the time at which the request was created, an indication of whether or not mail will be sent, where mail will be sent, and the user name on the originating machine. If a batch queue is being examined, resource limits, planned disposition of *stderr* and *stdout*, any advice concerning the command interpreter, and the umask value are shown.

The relative ordering of requests within a queue does not always determine the order in which the requests are run. The CXbatch request scheduler is allowed to make exceptions to the request ordering for the sake of efficient machine resource usage. However, requests appearing near the beginning of the queue have higher priority than requests appearing later, and are usually run before requests appearing later on in the queue.

QUEUE STATE

The general state of a queue is defined by two principal properties of the queue.

The first property determines whether or not requests can be submitted to the queue. If they can, the queue is said to be *enabled*. Otherwise the queue is said to be *disabled*. One of the words CLOSED, ENABLED, or DISABLED appears in the queue status field to indicate the respective queue states of: enabled (with no local CXbatch daemon), enabled (and local CXbatch daemon is present), and disabled. Requests can only be submitted to the queue if the queue is enabled and the local CXbatch daemon is present.

The second principal property of a queue determines if requests that are ready to run, but are not now presently running, will be allowed to run upon the completion of any currently running requests, and whether any requests are presently running in the queue.

If queued requests not already running are blocked from running, and no requests are presently executing in the queue, the queue is said to be *stopped*. If the same situation exists with the difference that at least one request is running, the queue is said to be *stopping*, where the requests presently executing will be allowed to complete execution, but no new requests will be spawned.

If queued requests ready to run are only prevented from doing so by the CXbatch request scheduler, and one or more requests are presently running in the queue, the queue is said to be *running*. If the same circumstances prevail with the exception that no requests are presently running in the queue, the queue is said to be *inactive*. Finally, if the CXbatch daemon for the local host upon which the queue resides is not running, but the queue would otherwise be in the state of *running* or *inactive*, the queue is said to be *shutdown*. The queue states describing the second principal property of a queue are therefore respectively displayed as STOPPED, STOPPING, RUNNING, INACTIVE, and SHUTDOWN.

REQUEST STATE

The state of a request may be *arriving*, *holding*, *waiting*, *queued*, *routing*, *running*, *departing*, *exiting*, or *checkpointed*.

- *arriving* The request is being enqueued from a remote host.
- *holding* The request is presently prevented from entering any other state (including the *running* state), because a *hold* has been placed on the request.
- *waiting* The request was submitted with the constraint that it not run before a certain date and time, and that date and time have not yet arrived.
- *queued* The request is eligible to proceed (by *routing* or *running*).
- *routing* The request has reached the head of a pipe queue and is receiving service.
- *departing* A request is *departing* from the time the pipe queue turns to other work until the request has arrived intact at its destination.
- *running* The request has reached its final destination queue and is actually executing.
- *exiting* The batch request has completed execution and will exit from the system after the required output files have been returned (to possibly remote machines).
- *checkpointed* A failed attempt was made to restart the checkpointed request. This can occur if CXbatch is shut down, checkpointing running requests, CXbatch is started, and a request fails to restart.

Imagine a batch request originating on a workstation, destined for the batch queue of a computation engine, to be run immediately. That request would first go through the states *queued*, *routing*, and *departing* in a local pipe queue. Then it would disappear from the pipe queue. From the point of view of a queue on the computation engine, the request would first be *arriving*, then *queued*, *running*, and finally *exiting*. Upon completion of the *exiting* phase of execution, the batch request would disappear from the batch queue.

DIAGNOSTICS

qstat returns an exit status describing what it did. If there were no errors, the exit status is zero. If one or more of the queues were not listed, the exit status is the number of queues that weren't listed. If a fatal error occurs and none of the queues are listed (ex, a syntax error), the exit status is one of the codes defined in *<sysexits.h>*.

EX_USAGE The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.

EX_NOUSER The user specified with *-u* did not exist.

EX_OSFILE Some batch system file does not exist, cannot be opened, or has an error.

SEE ALSO

qdel(1), qjlist(1), qlimit(1), qsub(1), qmgr(8)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qsub – submit a CXbatch request.

SYNOPSIS

qsub [*flags*] [*script-file*]

DESCRIPTION

qsub submits a batch request to CONVEX CXbatch.

If no *script-file* is specified, the set of commands to be executed as a batch request is taken directly from the standard input file (*stdin*). In all cases however, the *script file* is spooled, so that later changes will *not* affect previously queued batch requests.

All of the flags that can be specified on the command line can also be specified within the first comment block inside the batch request *script file* as *embedded default flags*. Such flags appearing in the batch request *script file* set default characteristics for the batch request. If the same flag is specified on the command line, the command line flag (and any associated value) takes precedence over the *embedded* flag. See the section entitled **LONG DESCRIPTION** for more information on *embedded default flags*.

What follows is a terse definition of the flags implemented by the *qsub* command (see the section **LONG DESCRIPTION** for the complete definition and syntax used for each of these flags).

–a – run request after stated time
–b – set the billing activity for request
–c – request is checkpointable
–cp – specify periodic checkpoint frequency
–e – direct *stderr* output to stated destination
–eo – direct *stderr* output to the *stdout* destination
–h – put request on hold after submitting
–i – request requires current directory to be imported
–ke – keep *stderr* output on the execution machine
–ko – keep *stdout* output on the execution machine
–l – run request under a login shell
–l*x* – establish a resource limit
–mb – send mail when the request begins execution
–me – send mail when the request ends execution
–mu – send mail for the request to the stated user
–ni – don't import the current directory
–nr – declare that batch request is not restartable
–nc – request is not checkpointable
–o – direct *stdout* output to the stated destination
–p – specify intra-queue request priority
–q – queue request in the stated queue
–r – assign stated request name to the request
–s – specify shell to interpret the batch request script
–t – signal process when the job completes
–x – export all environment variables with request
–y – append accounting data to stdout output file
–z – submit the request silently

If you request that a batch request be run under a login shell, the system and user startup files will be read (*/etc/login* and *~/.login* for C-shell or */etc/profile* and *$HOME/.profile* for Bourne shell). A C-shell login shell will also read */etc/logout* and *~/.logout* while exiting. The *~/.cshrc* file is read by the C-shell regardless of whether or not it is executed as a login shell.

The environment string ENVIRONMENT=BATCH is added to the environment so that shell scripts (and the user's *.profile* (Bourne shell) or *.cshrc* and *.login* (C-shell) scripts), can test for batch request execution when appropriate, and not (for example) perform any setting of terminal characteristics, because a batch request is not connected to an input terminal. For example, if your login shell is C-shell, the following *.login* file prevents *stty*, *tset*, and *msgs* from being run during batch jobs.

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

If your login shell is Bourne shell, the following *.profile* file has the same effect.

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

When CXbatch is configured on more than one machine, it is possible for users to submit jobs to remote machines in the batch network. However, CXbatch must ensure that the submitter has permission to access the remote machine. This is accomplished with the *user equivalence* mechanism described in *hosts.equiv*(5) (the same method that is used by *rlogin* and *rsh*). If the machines in the batch network are not equivalenced in the */etc/hosts.equiv* file, users must create a *.rhosts* file in their home directories. If this is not done, the submitter will not have access to the remote machine. Read the *hosts.equiv*(5) man page for more information.

LONG DESCRIPTION

As described above, it is possible to specify *default* flags within the batch request *script file* that configure the default behavior of the batch request. The algorithm used to scan for such *embedded default flags* within a batch request script file is:

1. Read the first line of the *script file*.
2. If the current line contains only whitespace characters, or the first non-whitespace character of the line is ":", goto step 7.
3. If the first non-whitespace character(s) of the current line is not "#" or "$!", goto step 8.
4. If the second non-whitespace character in the current line is *not* the "@" character, or the character immediately following the second non-whitespace character in the current line is *not* a "$", goto step 7.
5. If no "-" is present as the character *immediately* following the "@$" sequence, goto step 8.
6. Process the *embedded* flag, stopping the parsing process upon reaching the end of the line, or upon reaching the first unquoted "#" or "$!" character(s).
7. Read the next *script file* line. Goto step 2.
8. End. No more *embedded* flags are recognized.

Here is an example of the use of *embedded* flags within the *script file*.

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

The following paragraphs give detailed descriptions of the *flags* supported by the *qsub* command.

–a *date-time* Do not run the batch request before the specified date and/or time. If a *date-time* specification is composed of two or more tokens separated by whitespace characters, the *date-time* specification must be placed within double quotes as in **–a** *"July, 4, 2026 12:31-EDT"* or otherwise escaped such that *qsub* and the shell will interpret the entire *date-time* specification as a single-character string. This restriction also applies when an embedded default **–a** flag with accompanying *date-time* specification appears within the batch request *script file*.

The syntax accepted for the *date-time* parameter is relatively flexible. Unspecified date and time values default to an appropriate value (e.g., if no date is specified, the current month, day, and year are assumed).

A date may be specified as a month and day (current year assumed), or the year can also be explicitly specified. It is also possible to specify the date as a weekday name (e.g. "*Tues*"), or as one of the strings: "*today*" or "*tomorrow*". Weekday names and month names can be abbreviated by any three-character (or longer) prefix of the actual name. An optional period can follow an abbreviated month or day name.

Time of day specifications can be given using a twenty-four hour clock, or "*am*" and "*pm*" specifications may be used. In the absence of a meridian specification, a twenty-four hour clock is assumed.

It should be noted that the time of day specification is interpreted using the precise meridian definitions whereby "*12am*" refers to the twenty-four hour clock time of 0:00:00, "*12m*" refers to noon, and "*12-pm*" refers to 24:00:00. Alternatively, the phrases "*midnight*" and "*noon*" are accepted as time of day specifications, where "*midnight*" refers to the time of 24:00:00.

A time zone may also appear at any point in the *date-time* specification. Thus, it is legal to say: "*April 1, 1987 13:01-PDT*". In the absence of a time zone specification, the local time zone is assumed, with daylight savings time being inferred when appropriate, based on the date specified.

All alphabetic comparisons are performed in a case insensitive fashion such that both "*WeD*" and "*weD*" refer to the day of Wednesday.

Some valid *date-time* examples are:

01-Jan-1986 12am, PDT
Tuesday, 23:00:00
11pm tues.
tomorrow 23:00-MST

–b [*group.*]*activity*
Set the billing activity for request. The *group* and *activity* arguments refer to entries in the */etc/group* and */etc/activities* files respectively. The *group.activity* combination must correspond to an entry in the */etc/actwho* file. If *group* is omitted, the current primary group of the submitting user is assumed. A request aways runs under the submitting user's default group. The *group* argument to this option is used only in verifying a user's access to the selected *activity*. See the ***bill*(1) man** page for more information.

-c Specify that this request is checkpointable. A request queued with this flag is checkpointed automatically before a CXbatch shutdown, and may be explicitly checkpointed using CXbatch commands.

-cp *period* Declare that this request should be checkpointed periodically by CXbatch at intervals of *period*. The *period* is specified as <[*number*] *unit*>, where *number* is a positive integer and *unit* is [*Minutes* | *Hours* | *Days* | *Weeks*].

–e [*machine:*][[/]*path/*] *stderr-filename*
Direct output generated by the batch request which is sent to the *stderr* file to the named [*machine:*][[/]*path/*] *stderr-filename*

The brackets "[" and "]" enclose optional portions of the *stderr* destination *machine*, *path*, and *stderr-filename*.

If no explicit *machine* destination is specified, the destination machine defaults to the machine that originated the batch request or to the machine that will eventually run the request, depending on the respective absence or presence of the **–ke** flag.

If no *machine* destination is specified, and the path/filename does not begin with a "/", the current working directory is prepended to create a fully qualified path name, provided that the **–ke** (keep *stderr*) flag is also absent. In all other cases, any partial path/filename is interpreted relative to the user's home directory on the *stderr* destination machine.

This flag cannot be specified when the **–eo** flag option is also present.

If the **–eo** and **–e** [*machine:*][[/]*path/*] *stderr-filename* flag options are not present, all *stderr* output for the batch request is sent to the file whose name consists of the first seven characters of the *request-name* followed by the characters: "*.e*", followed by the request sequence number portion of the *request-id* discussed below. In the absence of the **–ke** flag, this default *stderr* output file is placed on the machine that originated the batch request in the current working directory, as defined when the batch request was first submitted. Otherwise, the file is placed in the user's home directory on the execution machine.

–eo Direct all output that would normally be sent to the *stderr* file to the *stdout* file for the batch request. This flag cannot be specified when the **–e** [*machine:*][[/]*path/*] *stderr-filename* flag option is also present.

–h Put request on hold after submitting. The request is put into a HOLD (user hold) rather that a QUEUED state at the time of submittal. The hold can be removed using the *qmgr*(8) 'RELease Request' command.

–i Some jobs may require access to files located in the directory from which a job is

submitted. This option tells CXbatch that the current working directory should be imported before running this job. If the execution queue is on the same machine as the current working directory, CXbatch will change directories before starting the job. However, if the execution queue is on another machine, CXbatch will import the current directory with NFS. **NOTE:** CXbatch will make temporary NFS mounts into the /tmp filesystem. Care should be taken that any automatic clean-up operations on the /tmp filesystem do not traverse NFS mount points.

–ke In the absence of an explicit *machine* destination for the *stderr* file produced by a batch request, the *machine* destination chosen for the *stderr* output file is the machine that originated the batch request. In some cases, however, this behavior may be undesirable, so the **–ke** flag can be specified which instructs CXbatch to leave any *stderr* output file produced by the request on the machine that actually *executed* the batch request.

This flag is meaningless if the **–eo** flag is specified and cannot be specified if an explicit *machine* destination is given for the *stderr* parameter of the **–e** flag.

–ko In the absence of an explicit *machine* destination for the *stdout* file produced by a batch request, the *machine* destination chosen for the *stdout* output file is the machine that originated the batch request. In some cases, however, this behavior may be undesirable, and so the **–ko** flag can be specified which instructs CXbatch to leave any *stdout* output file produced by the request on the machine that actually *executed* the batch request.

This flag cannot be specified if an explicit *machine* destination is given for the *stdout* parameter of the **–o** flag.

–l The submitted request will be run under a login shell. This was the default behavior in the V1.0 release of CXbatch, but is now only done if explicitly requested. See the discussion above regarding shell start-up files and refer to the appropriate shell man page for details on the differences between login and non-login shells. **NOTE:** Running a job request under a login C-shell will cause the C-shell to issue a warning into the request's output file. This is a function of the C-shell that cannot be suppressed by CXbatch.

–l*x* ***limit-argument***

Set a resource limit. Available limits are summarized in the following table.

Resource Limits

Limit option	Limited resource	Enforcement
–lc *size-limit*	per-process corefile size	truncation
–ld *size-limit*[,*warn-limit*]	per-process data-segment	request denied
–lf *size-limit*[,*warn-limit*]	per-process perm-file	SIGXFSZ
–lF *size-limit*[,*warn-limit*]	per-request perm-space	none
–lm *size-limit*[,*warn-limit*]	per-process memory	none
–lM *size-limit*[,*warn-limit*]	per-request memory	none
–ln *nice-value*	per-process nice value	setpriority(2)
–ls *size-limit*[,*warn-limit*]	per-process stack-segment	request denied
–lt *time-limit*[,*warn-limit*]	per-process CPU time	SIGXCPU
–lT *time-limit*[,*warn-limit*]	per-request CPU time	none
–lv *size-limit*[,*warn-limit*]	per-process temp-file	none
–lV *size-limit*[,*warn-limit*]	per-request temp-space	none
–lw *size-limit*	per-process working set	paging

The *per-process corefile size limit* sets the maximum size of a corefile created by a process. The *per-process data-segment size limit* sets the maximum size to which a process's data-segment can grow. The *per-process perm-file size limit* sets the

maximum size to which a permanent file written to by a process can grow. The *per-request perm-file space limit* sets the maximum file space which can be used by permanent files opened for writing by all of the processes in a batch request. The *per-process memory size limit* sets the maximum amount of memory a process can consume. The *per-request memory space limit* sets the maximum amount of memory which can be used by all of the processes in a batch request. The *per-process nice value* sets the nice value for all processes comprising the running batch request. The *per-process stack-segment size limit* sets the maximum size to which a process's stack-segment can grow. The *per-process CPU time limit* sets the maximum amount of CPU time a process can consume. The *per-request CPU time limit* sets the maximum amount of CPU time all of the processes that constitute a batch request can consume. The *per-process temp-file size limit* sets the maximum size to which a temporary file written to by a process can grow. The *per-request temp-file space limit* sets the maximum file space which can be used by temporary files opened for writing by all of the processes in a batch request. The *per-process working set size limit* sets the maximum amount of physical memory each process within a running batch request should use.

Enforcement of limits is done by the underlying UNIX implementation, normally through termination by a signal. Per-process limits are enforced only on the process exceeding the limit; per-request limits are enforced on all the processes which make up a request. The optional warning limits allow warning notification to be made processes where such warnings are supported by the underlying UNIX implementation. Not all UNIX implementations support all the resource limits listed above. The *qsub* command will pass all specified limits on to the destination machine. If a batch request specifies a limit, and the machine upon which the batch request is eventually run does not support the enforcement of that limit, the limit is simply ignored.

The 'Enforcement' column of the table above indicates the ConvexOS specific enforcement policy of each resource limit. At this time, all *warn-limits* are treated identically to the hard limits. ConvexOS makes no distinction between permanent and temporary files; all files are treated as permanent.

See the section entitled **LIMITS** for more information on the implementation of batch request limits and for a description of the precise syntax of a the various limit arguments.

–mb Send mail to the user on the originating machine when the request begins execution. If the **–mu** flag is also present, mail is sent to the user specified for the **–mu** flag instead of to the invoking user.

–me Send mail to the user on the originating machine when the request has ended execution. If the **–mu** flag is also present, mail is sent to the user specified for the **–mu** flag instead of to the invoking user.

–mu *user-name*
Specify that any mail concerning the request should be delivered to the user *user-name*. *user-name* may be formatted either as *user* (containing no '@' characters), or as *user@machine*. In the absence of this flag, any mail concerning the request is sent to the invoker on the originating machine.

–ni A queue can be configured so that it tries to import the originating directory of each job it runs. If this option is specified, CXbatch does not import the current directory.

-nc Advise CXbatch that this request is not checkpointable. A request queued with this option will not be checkpointed at CXbatch shutdown, nor is it possible to

checkpoint the request with any other CXbatch commands.

–nr Declare that the request is non-restartable. If this flag is specified, the request is not restarted by CXbatch upon system boot if the request was running at the time of a CXbatch shutdown or system crash.

Requests submitted with the qsub -nr flag will not be checkpointed on shutdown and will not be restarted on CXbatch startup. The submitting user will be informed by mail.

By default, CXbatch assumes that all requests are restartable, with the caveat that it is the responsibility of the user to ensure that the request will execute correctly if restarted, by use of appropriate programming techniques.

Requests that are not running are always preserved across host crashes and CXbatch shutdowns for later requeuing, with or without this flag.

When CXbatch is shutdown by an operator command to the *qmgr*(8) CXbatch control program, a **SIGTERM** signal is sent to all processes in all running CXbatch requests on the local host, and all queued CXbatch requests are barred from beginning execution. After a finite number of seconds have elapsed (typically sixty, but this value can be overridden by the operator), all remaining processes in all remaining running CXbatch requests are killed by the signal **SIGKILL**.

For a CXbatch request to be properly restarted after a CXbatch shutdown, the **–nr** flag must not be specified, and the spawned batch request shell must ignore **SIGTERM** signals (which is done by default). The spawned batch request shell also must not exit before the final **SIGKILL** arrives. Because the batch request shell is spawning commands and programs, waiting for their completion, this implies that the commands and programs being executed by the batch request shell must also be immune to **SIGTERM** signals, saving state as appropriate before being killed by the final **SIGKILL** signal.

See the **CAVEATS** section below for more discussion about the restartability of batch requests.

–o [*machine:*][[/]*path/*] *stdout-filename*
Direct output generated by the batch request which is sent to the *stdout* file to the named [*machine:*][[/]*path/*] *stdout-filename*

The brackets "[" and "]" enclose optional portions of the *stdout* destination *machine*, *path*, and *stdout-filename*.

If no explicit *machine* destination is specified, the destination machine defaults to the machine that originated the batch request or to the machine that will eventually run the request, depending on the respective absence or presence of the **–ko** flag.

If no *machine* destination is specified, and the path/filename does not begin with a "/", the current working directory is prepended to create a fully-qualified path name, provided that the **–ko** (keep *stdout*) flag is also absent. In all other cases, any partial path/filename is interpreted relative to the user's home directory on the *stdout* destination machine.

If no **–o** [*machine:*][[/]*path/*] *stdout-filename* flag is specified, all *stdout* output for the batch request is sent to the file whose name consists of the first seven characters of the *request-name* followed by the characters: "*.o*", followed by the request sequence number portion of the *request-id* discussed below. In the absence of the **–ko** flag, this default *stdout* output file is placed on the machine that originated the batch request in the current working directory, as defined when the batch request was first submitted. Otherwise, the file is placed in the user's home directory on the execution machine.

-p *priority* Assign an *intra-queue* priority to the request. The specified *priority* must be an integer, and must be in the range [0..63], inclusive. A value of 63 defines the highest *intra-queue* request priority, while a value of 0 defines the lowest. This priority does *not* determine the execution priority of the request. This priority is only used to determine the relative ordering of requests within a queue.

When a request is added to a queue, it is placed at a specific position within the queue such that it appears ahead of all existing requests whose priority is less than the priority of the new request. Similarly, all requests with a higher priority remain ahead of the new request when the queuing process is complete. When the priority of the new request is equal to the priority of an existing request, the existing request takes precedence over the new request.

If no *intra-queue* priority is chosen by the user, CXbatch assigns a default value.

The CXbatch manager can assign a maximum request priority on a per queue basis. The maximum request priority is a ceiling on priorities of requests submitted to that queue. A request that specifies a priority higher than the maximum for that queue has its priority lowered to the maximum.

-q *queue-name*[@*host*]
Specify the queue to which the batch request is to be submitted. If no **-q** *queue-name*[@*host*] specification is given, the user's environment variable set is searched for the variable QSUB_QUEUE. If this environment variable is found, the character string value for QSUB_QUEUE is presumed to name the queue to which the request should be submitted. If the QSUB_QUEUE environment variable is not found, the request is submitted to the default batch request queue, *if* defined by the local system administrator. Otherwise, the request cannot be queued, and an appropriate error message is displayed. The *host* specifies the host where the queue resides. If no *host* is given, the local host is assumed. Not all hosts accept remote submissions.

-r *request-name*
Assign the specified *request-name* to the request. In the absence of an explicit **-r** *request-name* specification, the *request-name* defaults to the name of the *script file* (leading path name removed) given on the command line. If no *script file* was given, the default *request-name* assigned to the request is **STDIN**.

In all cases, if the *request-name* is found to begin with a digit, the character "R" is prepended to prevent a *request-name* from beginning with a digit. All *request-names* are truncated to a maximum length of 15 characters.

-s *shell-name*
Specify the absolute path name of the shell that is used to interpret the batch request script. This flag unconditionally overrides any *shell strategy* configured on the execution machine for selecting which shell to spawn in order to interpret the batch request script.

In the absence of this flag, the CXbatch system at the execution machine uses one of three distinct *shell choice strategies* for the execution of the batch request. Any one of the three strategies can be configured by a system administrator for each CXbatch machine.

The three shell strategies are called:

- *fixed*
- *free*
- *login*

These shell strategies respectively cause the configured *fixed* shell to be exec'd to interpret all batch requests; cause the user's login shell as defined in the password file to be exec'd, which in turn chooses and spawns the appropriate shell for interpreting the batch request script; or cause only the user's login shell to be exec'd to interpret the script.

A shell strategy of *fixed* means that the same shell (as configured by the system administrator) is used to execute all batch requests.

A shell strategy of *free* runs the batch request script ***exactly*** as would an interactive invocation of the script and is the default CXbatch shell strategy.

The strategies of *fixed* and *login* exist for host systems that are short on available free processes. In these two strategies, a single shell is exec'd, and that same shell is the shell that executes all of the commands in the batch request script.

The *shell strategy* configured for a particular CXbatch system can be determined by the *qlimit*(1) command.

–t *process-id* Signal process *process-id* when the job completes execution. The signal sent depends on how the job completed. If the job runs to normal completion, **SIGTERM** is sent. If the job is aborted while running, **SIGUSR1** is sent. If the job is deleted before it starts executing, **SIGUSR2** is sent.

–x Export all environment variables. When a batch request is submitted, the current values of the environment variables HOME, SHELL, PATH, USER, and MAIL are saved for later recreation when the batch request is spawned, as the respective environment variables QSUB_HOME, QSUB_SHELL, QSUB_PATH, QSUB_USER, and QSUB_MAIL. Unless the **–x** flag is specified, no other environment variables are exported from the originating host for the batch request. If the **–x** flag option is specified, all remaining environment variables whose names do not conflict with the automatically exported variables are also exported with the request. These additional environment variables are recreated under the same name when the batch request is spawned.

–y Append accounting data to the stdout output file if accounting is enabled for the queue in which the job runs. This data includes: queue, host, sequence number, remote host, submission time, start time, completion time, time spent executing in user mode, and time spent executing in the system.

–z Submit the batch request silently. If the request is submitted successfully, no messages are displayed indicating this fact. Error messages are, however, always displayed.

If the batch request is successfully submitted, and the **–z** flag has not been specified, the *request-id* of the request is displayed to the user. A *request-id* is always of the form *seqno.hostname*, where *seqno* refers to the sequence number assigned to the request by CXbatch, and *hostname* refers to the name of originating local machine. This identifier is used throughout CXbatch to uniquely identify the request, no matter where it is in the network.

The following events take place in the following order when a CXbatch ***batch*** request is spawned:

1. The process that will become the head of the ***process group*** for all processes comprising the batch request is created by CXbatch.
2. Resource limits are enforced.
3. The real and effective group-id of the process is set to the group-id as defined in the local password file for the request owner.
4. The real and effective user-id of the process is set to the real user-id of the batch request owner.

5. The user file creation mask is set to the value that the user had on the originating machine when the batch request was first submitted.

6. It the user explicitly specified a shell by use of the **–s** flag (discussed above), then that user-specified shell is chosen as the shell that will be used to execute the batch request script. Otherwise, a shell is chosen based upon the *shell strategy* as configured for the local CXbatch system. (See the earlier discussion of the **–s** flag for a description of the possible *shell strategies* that can be configured for a CXbatch system.)

7. The environment variables of **HOME** and **SHELL**, are set from the user's password file entry, as though the user had logged directly into the execution machine.

8. The environment string: **ENVIRONMENT=BATCH** is added to the environment so that shell scripts (and the user's *.profile* (Bourne shell) or *.cshrc* and *.login* (C-shell) scripts) can test for batch request execution when appropriate, and not (for example) perform any setting of terminal characteristics, because a batch request is not connected to an input terminal.

9. The environment variables of **QSUB_WORKDIR**, **QSUB_HOST**, **QSUB_REQNAME**, and **QSUB_REQID** are added to the environment. These environment variables equate to the obvious respective strings of the working directory at the time that the request was submitted, the name of the originating host, the name of the request, and the request *request-id*.

10. All of the remaining environment variables saved for recreation when the batch request is spawned are added at this point to the environment. When a batch request is initially submitted, the current values of the environment variables **HOME**, **SHELL**, **PATH**, **USER**, and **MAIL** are saved for later recreation when the batch request is spawned. When recreated however, these variables are added to the environment under the respective names **QSUB_HOME**, **QSUB_SHELL**, **QSUB_PATH**, **QSUB_USER**, and **QSUB_MAIL** to avoid the obvious conflict with the local version of these environment variables. Additionally, all environment variables exported from the originating host by the **–x** option are added to the environment at this time.

11. The current working directory is then set to the user's home directory on the execution machine, and the chosen shell is exec'd to execute the batch request script with the environment as constructed in the steps outlined above.

Unless the **–l** option is specified, the chosen shell is exec'd as a non-login shell and no start-up files (except the *~/.cshrc* for a C-shell) are read. If the **–l** option is selected, the chosen shell is exec'd as though it were the *login* shell. If the Bourne shell is chosen to execute the script, */etc/profile* and the user's *.profile* file are read. If the C-shell is chosen, */etc/login* and the user's *.cshrc* and *.login* scripts are read and when the job exits, */etc/logout* and the user's *.logout* script are read.

If the user did not specify a shell for the batch request, CXbatch chooses which shell is used to execute the shell script, based on the *shell strategy* as configured by the system administrator. (See the earlier discussion of the *–s* flag.)

In such a case, a *free* shell strategy instructs CXbatch to execute the login shell for the user (as configured in the password file). The login shell is in turn instructed to examine the shell script file and fork another shell *of the appropriate type* to interpret the shell script, behaving *exactly* as an interactive invocation of the script.

Otherwise no additional shell is spawned, and the chosen *fixed* or *login* shell sequentially executes the commands contained in the shell script file until completion of the batch request.

QUEUE TYPES

CXbatch supports three different queue types that serve to provide three very different functions. These three queue types are known as *batch*, *pipe*, and *network*.

The queue type of *batch* can only be used to execute CXbatch *batch requests*. Only CXbatch *batch requests* created by the *qsub*(1) command can be placed in a *batch queue*.

Queues of type *pipe* are used to send CXbatch requests to other *pipe* queues or to request destination queues of type *batch*. In general, *pipe queues*, in combination with *network queues*, act as the mechanism that CXbatch uses to transport *batch* requests to distant queues on remote machines. It is also legal for a *pipe queue* to transport requests to queues on the *same* machine.

When a *pipe queue* is defined, it is given a *destination set* that defines the set of possible destination queues for requests entered in that *pipe queue*. In this manner, it is possible for a *batch* request to pass through many pipe queues on its way to its ultimate destination, which must eventually be a queue of type *batch*.

Each *pipe queue* has an associated *server*. For each request handled by a *pipe queue*, the associated server is spawned which must select a queue destination for the request being handled, based upon the characteristics of the request and upon the characteristics of each queue in the *destination set* defined for the pipe queue.

Because a different server can be configured for each *pipe* queue, and *batch* queues can be endowed with the *pipeonly* attribute that will only admit requests queued via another *pipe queue*, it is possible for respective CXbatch installations to use *pipe queues* as a *request class* mechanism, placing requests that ask for different resource allocations in different queues, each of which can have different associated limits and priorities.

It is also completely possible for a *pipe queue server*, when handling a request, to discover that no *destination queue* will accept the request, for various reasons that can include insufficient resource limits to execute the request or a lack of a corresponding account or privilege for queuing at a remote queue. In such circumstances, the request is deleted, and the user is notified by mail (see *mail*(1)).

The queue type of *network*, as alluded to earlier, is implicitly used by *pipe* queues to pass CXbatch requests between machines and is also used to handle queued file transfer operations.

QUEUE ACCESS

CXbatch supports queue access restrictions. For each queue of queue type other than *network*, access may be either *unrestricted* or *restricted*. If access is *unrestricted*, any request may enter the queue. If access is *restricted*, a request can only enter the queue if the requester or the requester's login group has been given access to that queue (see *qmgr*(8)). Requests submitted by the superuser are an exception; they are always queued, even if the superuser has not explicitly been given access.

Use *qstat*(1) to determine who has access to a particular queue.

LIMITS

CXbatch supports many batch request resource limit types that can be applied to a batch request. The existence of configurable resource limits allows a CXbatch user to set resource limits within which his or her request must execute.

The syntax used to specify a *limit-value* for one of the *limit-flags* (**-l***x*), is quite flexible and describes values for three general limit categories. These three general categories respectively deal with *time limits*, priority (*nice value*) limits and file/memory *size limits*.

All the *limit-flags* expect a single *limit-argument*. If the *limit-argument* consists any whitespace which will cause the it to be passed to qsub as multiple tokens, it should be enclosed within double quotes or otherwise escaped such that is is passed a a single, character-string token. The optional *warn-limit* should also be included as part of the same token. This also applies to *limit-arguments* associated with *limit-flags* embedded within the batch request *script file*.

For *finite* CPU time limits, the *limit-value* is expressed in the reasonably obvious format:

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

Whitespace can appear anywhere between the principal tokens, with the exception that no whitespace can appear around the decimal point. **NOTE:** The *milliseconds* value may be ignored if the system on which the request is run does not support such granularity.

Example time *limit-values are:*

1234 : 58 : 21.29	– 1234 hrs 58 mins 21.290 secs
12345	– 12345 seconds
121.1	– 121.100 seconds
59:01	– 59 minutes and 1 second

Priority limits are expressed as a small, signed integer in the range acceptable for *nice values* on the execution machine. In general, increasingly negative *nice values* cause the relative execution priority of a process to increase, while increasingly positive *nice values* causes the relative priority to decrease. Because different UNIX implementations often support different finite ranges of *nice values*, CXbatch allows the specification of *nice values* that can eventually turn out to be outside the limits for the UNIX implementation running at the execution machine. In such cases, CXbatch simply binds the specified *nice values* limit to within the necessary range as appropriate.

For the *finite* size limits the acceptable syntax is:

.*fraction* [*units*]

or

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

where the *integer* and *fraction* tokens represent strings of up to eight decimal digits, denoting the obvious values. In both cases, the *units* of allocation may also be specified as one of the case insensitive strings:

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

In the absence of any *units* specification, the units of *bytes* are assumed.

For all limit types with the exception of the *nice* limit-value (**–ln**), it is possible to state that no limit should be applied. This is done by specifying a *limit-value* of "unlimited" or any initial substring thereof. Whenever an *infinite limit-value* is specified for a particular resource type, the batch request operates as though no explicit limits have been placed upon the corresponding resource, other than by the limitations of the physical hardware involved.

The complications caused by *batch request* resource limits first show up when queuing a *batch request* in a *batch queue*. This operation is described in the following paragraphs.

If a batch request specifies a limit that cannot be enforced by the underlying UNIX implementation, the limit is simply ignored, and the batch request operates as though there were no limit (other than the obvious physical maximums) placed upon that resource type. (See the *qlimit*(1) command to find out what limits are supported by a given machine.)

For each remaining *finite* limit that can be supported by the underlying UNIX implementation that is *not* a CPU *time-limit* or UNIX *execution-time nice-value-limit*, the *limit-value* is internally converted to the units of *bytes* or *words*, whichever is more appropriate for the underlying machine architecture.

As an example, a *per-process memory size limit value* of 321 megabytes would be interpreted as 321 x 2^20 bytes, provided that the underlying machine architecture was capable of directly addressing single bytes. Thus the original limit *coefficient* of 321 would become 321 x 2^20. On a machine that was only capable of addressing words, the appropriate conversion of 321 x 2^20 *bytes / # of-bytes-per-word* would be performed.

If the result of such a conversion would cause overflow when the coefficient was represented as a *signed-long integer* on the supporting hardware, the coefficient is replaced with the coefficient of 2^*N*-1, where *N* is equal to the number of bits of precision in a signed long integer. For typical 32-bit machines, this *default extreme limit* would therefore be 2^*31*-1 bytes. For word addressable machines in the supercomputer class supporting 64-bit long integers, the *default extreme limit* would be 2^*63*-1 words.

Lastly, some implementations of UNIX reserve coefficients of the form: 2^*N*-1 as synonymous with infinity, meaning no limit is to be applied. For such UNIX implementations, CXbatch further decrements the *default extreme limit* so as not to imply infinity.

The identical internal conversion process as described in the preceding paragraphs is also performed for each *finite limit-value* configured for a particular batch queue using the *qmgr*(8) program.

After all of the applicable *limit-values* have been converted as described above, each resulting *limit-value* is then compared against the corresponding *limit-value* as configured for the destination batch queue. If, for every type of limit, the batch queue *limit-value* is *greater than* or *equal to* the corresponding batch request *limit-value*, the request can be successfully queued, provided that no other anomalous conditions occur. For request *infinity limit-values*, the corresponding queue *limit-value* must also be configured as infinity.

These resource limit checks are performed irrespective of the batch request arrival mechanism, either by a direct use of the *qsub*(1) command or by the indirect placement of a batch request into a batch queue via a *pipe* queue. It is impossible for a batch request to be queued in a batch queue if *any* of these resource limit checks fail.

Finally, if a request fails to specify a *limit-value* for a resource limit type that is supported on the execution machine, the corresponding *limit-value* configured for the destination queue becomes the *limit-value* for the unspecified request limit.

Upon the successful queuing of a request in a batch queue, the set of limits under which the request will execute is frozen and will not be modified by subsequent *qmgr*(8) commands that alter the limits of the containing batch queue.

CAVEATS

When a batch request is spawned, a new *process-group* is established such that all processes of the request exist in the same *process-group*. If the *qdel*(1) command is used to send a signal to a batch request, the signal is sent to all processes of the request in the created *process-group*. However, should one or more processes of the request choose to successfully execute a *setpgrp*(2) system call, such processes will *not* receive any signals sent by the *qdel*(1) command. This can lead to "rogue" requests whose constituent processes must be killed by other means such as the *kill*(1) command.

It is extremely wise for all processes of a CXbatch request to catch any **SIGTERM** signals. By default, the receipt of a **SIGTERM** signal causes the receiving process to die. CXbatch sends a **SIGTERM** signal to all processes in the established *process-group* for a batch request as a notification that the request should be prepared to be killed, either because of an *abort queue* command issued by an operator using the *qmgr*(8) program, or because it is necessary to shutdown CXbatch and all running requests as part of a general shutdown procedure of the local host.

It must be understood that the spawned *shell* ignores **SIGTERM** signals. If the current immediate child of the shell does not ignore or catch **SIGTERM** signals, it will be killed by the receipt of such, and the shell will go on to execute the next command from the script (if there is one). In any case, the shell will not be killed by the **SIGTERM** signal, though the executing command will have been killed.

After receiving a **SIGTERM** signal delivered from CXbatch, a process of a batch request typically has sixty seconds to get its "house in order" before receiving a **SIGKILL** signal (though the sixty second duration can be changed by the operator).

All batch requests terminated because of an operator CXbatch shutdown request that did not specify the **–nr** flag are considered restartable by CXbatch and are requeued (provided that the batch request shell process is still present at the time of the **SIGKILL** signal broadcast as discussed above), so that when CXbatch is rebooted, such batch requests will be respawned to continue execution. It is, however, up to the user to make the request restartable by the appropriate programming techniques. CXbatch simply spawns the request again as though it were being spawned for the first time.

Upon completion of a batch request, a mail message can be sent to the submitter (see the discussion of the **–me** flag above). In many instances, the completion code of the spawned Bourne or C-Shell is displayed. This is the value returned by the shell through the *exit*(2) system call.

Lastly, there is no good way to echo commands executed by unmodified versions of the Bourne and C shells. While the Bourne and C shells can be spawned in such a fashion as to echo the commands they execute, it is often very difficult to tell an echoed command from genuine output produced by the batch request.

Thus, one of the better ways to write the shell script for a batch request is to place appropriate lines in the shell script of the form:

echo "*explanatory-message*"

where the echoed message should be a meaningful message chosen by the user.

DIAGNOSTICS

qsub returns an exit status describing what it did. If there were no errors, the exit status is zero. If a fatal error occurs and the request is not submitted (e.g., a syntax error), the exit status is one of the codes defined in *<sysexits.h>*, or one if none of the *<sysexits.h>* codes are appropriate.

EX_USAGE	The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.
EX_OSFILE	Some batch system file does not exist, cannot be opened, or has an error.
EX_TEMPFAIL	Temporary failure; retry the request at a later time.
EX_NOPERM	You did not have sufficient permission to perform the operation.
EX_NOINPUT	The script file does not exist or is not readable.

SEE ALSO

bill(1), mail(1), qdel(1), qjlist(1), qlimit(1), qstat(1), qmgr(8) kill(2), setpgrp(2), signal(3c)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

batch-acct – CXbatch accounting file

DESCRIPTION

Batch accounting can be performed by CXbatch on a per queue basis. Batch accounting is enabled with the *qmgr* command *set accounting*. The accounting log file is specified with *set acc_logfile*. By default, accounting is off and the log file is */dev/null*.

CONVEX provides qsa(8) for processing and reporting the batch accounting data. The log file is in binary format, and its structure is described in the C include file */usr/include/batch-acct.h*.

FILES

/usr/include/batch-acct.h

SEE ALSO

qmgr(8), qsa(8)

NAME

nqsdaemon, netdaemon, logdaemon – CXbatch daemons

SYNOPSIS

/usr/lib/nqs/nqsdaemon

DESCRIPTION

nqsdaemon, *netdaemon*, and *logdaemon* are CXbatch daemons that are normally started at boot time from the */etc/rc.local* file. *nqsdaemon* automatically starts *netdaemon* and *logdaemon*.

nqsdaemon handles all local transactions, including submits, deletes, job scheduling, and system configuration. *netdaemon* handles all possibly remote transactions, including submits and file staging. *nqsdaemon* and *netdaemon* contact *logdaemon* when they need to print an error message. *logdaemon* sends the message to syslogd and notifies the batch managers if the error is fatal. See *qmgr*(8) for more information on defining CXbatch managers. *logdaemon* logs messages to syslogd under the LOG_BATCH facility. These messages are logged at the following levels: LOG_DEBUG, LOG_ERR, LOG_CRIT, LOG_INFO, and LOG_WARNING.

SEE ALSO

qdel(1), qjlist(1), qlimit(1), qstat(1), qsub(1), pipeclient(8), qmapmgr(8), qmgr(8), syslogd(8).

FILES

/usr/lib/nqs – directory containing CXbatch daemons
/etc/nmap – directory that contains network database

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

pipeclient, pipeldav – CXbatch pipe clients

SYNOPSIS

/usr/lib/nqs/pipeclient
/usr/lib/nqs/pipeldav [**-w** *weight*] [*host scale ...*]

DESCRIPTION

CXbatch supports pipe queues that are responsible for routing and delivering requests to other (possibly remote) queue destinations. With each pipe queue, there is an associated pipe client that is spawned to handle each request released from the queue for routing and delivery. When a pipe client is spawned, it is given complete freedom to route the request to any of the destinations in its destination set. CONVEX supplies two pipe clients, *pipeclient* and *pipeldav*, that use different methods for determining which destination should get the request.

The standard pipe client, *pipeclient*, routes the request to the first destination that will accept the request. Destinations may reject the request due to queue limit violations, lack of account authorization, and many other reasons.

pipeldav sorts the destination list by load factor, and tries destinations with low load factors first. The load factor is calculated as follows:

load factor = (*load avg* + *queue length* * *weight*) / *scale*

The *load average* is the current system load average on the destination machine. The *queue length* is the number of requests in the destination queue. The *weight* and *scale* are specified on the *pipeldav* command line and are the job weight and host scale factor.

See *qmgr*(8) for more information on configuring pipe queues.

RESTRICTIONS

The pipe client *pipeldav* gets load average information from *rstatd*. As a result, *pipeldav* is limited to load balancing over machines running *rstatd*.

SEE ALSO

qmgr(8)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qmapmgr – configures the CONVEX Extended Batch System (CXbatch) network

SYNOPSIS

qmapmgr

DESCRIPTION

The *qmapmgr* command builds the network database that CXbatch uses to direct files and messages. CXbatch needs this network database file no matter how simple the machine configuration.

This network database consists of three primary elements:

mid	A machine ID number that is unique in the CXbatch network being configured. CXbatch uses this *mid* to identify a specific machine. A maximum value of (2**31)-1 is permissible.
principle name	A machine name that is unique in the CXbatch network being configured.
alias	Names, other than the *principle name*, of machines in the network. Aliases are known only to the local CXbatch host.

To gain entry to the CXbatch network mapping manager, enter the *qmapmgr* command. You can enter any of the commands described below at the prompt. You can leave *qmapmgr* using the *exit* or *quit* commands.

COMMANDS

You can abbreviate commands to their minimum unambiguous length. For example, you can abbreviate *change name* to *ch n*. However, *c n* is not acceptable because the command *create* exists. You should enter all commands on a single command line. You can enter the following commands at the prompt.

add mid *mid principal name*
Adds a new machine to the configuration with the specified *mid* and *principle name.*

add name *alias mid*
Adds an *alias* for the machine with the specified *mid.*

change name *mid principle name*
Changes the *principle name* of the machine with the specified *mid.*

create
Creates a new network configuration database file.

delete mid *mid*
Removes the machine with the specified *mid* from the configuration database.

delete name *alias*
Deletes the given *alias* from the configuration database.

exit
Leaves the *qmapmgr* program.

get mid *name*
Displays the id of the machine with the specified *principle name* or *alias.*

get name *mid*
Displays the *principle name* of the machine with the specified *mid.*

help
Displays the available commands.

quit

Leaves the *qmapmgr* program.

show

Displays all *mid* entries with their corresponding *principle name* and *aliases.*

SEE ALSO

qmgr(8)

FILES

/etc/nmap – directory that contains network database

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qmgr – CXbatch queue manager program

SYNOPSIS

qmgr [*command* [*options*]]

DESCRIPTION

qmgr is the primary utility used to configure, administer and operate the CXbatch system. *qmgr* recognizes three classes of users: managers, operators, and users. The superuser is a manager by default. All other managers and operators are assigned using *qmgr* (except under ConvexOS/Secure where the assignment is made through the *authif*(8) utility). Under ConvexOS/Secure the user must have the **batch** subsystem authorization in addition to CXbatch operator or manager privileges. Managers can use any of the *qmgr* commands. Operators can use a subset of the commands. Users who are not granted manager or operator privileges can use only a few of the *qmgr* commands to display status information and manipulate their own requests.

The CXbatch system is made up of batch and pipe queues that transport and run batch requests. A batch queue holds requests for scheduled, perhaps delayed, processing. A pipe queue is a queue that can pass queued requests on to other pipe queues or batch queues. A batch request is a set of commands, or a shell script, that is to be run non-interactively with the results being returned to the user. The running of the request is handled by the CXbatch system rather than directly by the user.

qmgr can be invoked to run a single command given as the command line argument or, if no argument is given, as a command interpreter. As an interpreter, commands will be repeatedly prompted for and executed. The *qmgr* command set is extensive and will be discussed in functional groups below. Command keywords are case insensitive and can be abbreviated to their shortest unique substring. This is indicated in the command descriptions below by capitalizing the required substring. For example, when using the exit command, you must type at least 'ex' to distinguish it from the enable command. Thus the description for this command is:

EXit

Exit from the CXbatch manager subsystem.

INFORMATIONAL COMMANDS

The following commands return information about *qmgr* commands, CXbatch parameters and queue configuration. They are available to all users.

HElp [*command*]

Get help information. **HElp** without an argument displays information about what commands are available. **HElp** with an argument displays information about that command. The command may be partially specified as long as it is unique. A more complete help request yields more detailed information.

The **HElp** command provides information that is often more extensive than the command descriptions in this manual page! Use it.

SHOw All

Display the standard amount of information about *"limits supported"*, *managers*, *parameters*, and *queues*. See below.

SHOw LImits_supported

Display the list of CXbatch resource limit types that are meaningful on this machine. Note that users may request resource limits that are *not* meaningful on the machine where *qsub*(1) is invoked. If the request is to be executed on a remote machine where the limit is meaningful, then CXbatch honors it. Otherwise the unsupported limit is simply ignored.

SHOw LOng Queues [*queue* [*user*]]

Display in long format the status of all CXbatch queues on this host. If a *queue* is specified, output is limited to that queue. If a *user* is specified, output downplays any requests not belonging to that user.

SHOw Managers

Display the list of authorized CXbatch managers.

SHOw Parameters

Display the general CXbatch parameters.

SHOw Queues [*queue* [*user*]]

Display the status of all CXbatch queues on this host. If a *queue* is specified, output is limited to that queue. If a *user* is specified, output downplays any requests not belonging to that user.

QUEUE MANAGEMENT

The following commands are used to define batch and pipe queues. They can be used only by CXbatch managers. For more information on the different types of queues, see the **QUEUE TYPES** section below.

CReate Batch_queue *queue* **PRiority** = *n* [**PIpeonly**]
[**Run_limit** = *n*] [**Import_dir** = {*Yes,No,Available*}]
[**Share_policy Fixed** = *user*] [**Share_policy User**]

Define a batch queue named *queue* with inter-queue priority *n* (0..63). If **PIpeonly** is specified, then requests may enter this *queue* only if their source is a pipe queue. The specification of a **Run_limit** sets a ceiling on the maximum number of requests allowed to run in the batch queue at any given time. The default **Run_limit** is one.

The specification of the **Import_dir** attribute determines the availability of the user's current working directory to a request at runtime. See the **SEt Import_dir** command for more information.

The specification of a **Share_policy** affects how the CPU usage of jobs running in this *queue* is charged. See the **SEt SHAre_policy Fixed** and the **SEt SHAre_policy User** commands for more information.

CReate Pipe_queue *queue* **PRiority** = *n* **Server** = (*server*)
[**Destination** = *destination*]
[**Destination** = (*destination* [, *destination* ...])]
[**PIpeonly**] [**Run_limit** = *n*]

Define a pipe queue named *queue* with inter-queue priority *n* (0..63) and associate it with a *server*. This is done by specifying an absolute path name to the program binary (*server*) and any arguments required by the program. After **Destination** appears a list of one or more *destination* queues that requests from this pipe *queue* may be sent to. If **PIpeonly** is specified, then requests may enter this *queue* only if their source is a pipe queue. **Run_limit** sets a ceiling on the maximum number of requests allowed to run in the pipe queue at any given time. The default **Run_limit** is one.

ADd Alias *alias queue*

Add the specified queue *alias* to *"queue"*. A queue alias is an alternate name for a queue; any CXbatch command that requires a queue name will also work with an alias.

DElete Alias *alias*

Delete the specified queue *"alias"*. A queue alias is an alternate name for a queue; any CXbatch command that requires a queue name will also work with an alias.

DElete Queue *queue*

The *queue* is deleted. To delete a *queue*, no requests may be present in the *queue*, and the *queue* must be disabled. (See **"DIsable Queue"** below.)

SEt DESCription = *(description) queue*

Change the description of the named CXbatch queue.

SEt PRiority = *priority queue*

Specify the inter-queue *priority* of a *queue*.

QUEUE ACCESS

CXbatch supports queue access restrictions. For each queue access may be either *unrestricted* or *restricted*. If access is *unrestricted*, any request may enter the queue. If access is *restricted*, a request can only enter the queue if the requester or the requester's login group has been given access. Requests submitted by the superuser are an exception; they are always queued, even if the superuser has not explicitly been given access.

The following commands are used to grant access to queues. They can be used only by CXbatch managers.

In the **ADd** and **DElete** commands below a *group* or *user* can be specified in one of two ways: *name* or [*id*]. (The square brackets are literal characters used to indicate a gid or uid.)

ADd Groups = *group queue*
ADd Groups = (*group* [, *group* ...]) *queue*

The specified *group(s)* are added to the access list for *queue*.

ADd Users = *user queue*
ADd Users = (*user* [, *user* ...]) *queue*

The specified *user(s)* are added to the access list for *queue*.

DElete Groups = *group queue*
DElete Groups = (*group* [, *group* ...]) *queue*

The specified *group(s)* are deleted from the access list for *queue*.

DElete Users = *user queue*
DElete Users = (*user* [, *user* ...]) *queue*

The specified *user(s)* are deleted from the access list for *queue*.

SEt NO_Access *queue*

Specify that no one can place requests in *queue*.

SEt Unrestricted_access *queue*

Specify that no requests will be turned away from *queue* on the grounds of queue access restrictions.

BATCH QUEUE CONFIGURATION

The following commands are used to set the operating parameters for batch queues. They can only be used by CXbatch managers.

SEt ACCOunting = {*OFf,ON*} *queue*

Turn accounting on/off for a CXbatch batch queue. The queue named as a parameter of the command must already exist.

SEt ACTivity_id_offset = *offset queue*

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

SEt CHKpntable = {*Yes,No,Available*} *queue*

Sets the status of the per-queue checkpoint resource. The queue must exist. The queue's checkpoint resource value and the request's flags are used to determine if a request may be checkpointed.

If the checkpoint attribute is set to "Yes", then any request submitted to this queue will be checkpointed at CXbatch shutdown and may be checkpointed by the request owner or CXbatch operator, unless it explicitly requested not to be checkpointed. If this attribute is set to "Available", the request is checkpointable only if it specifically asked to be checkpointed. Requests in a queue with the checkpoint attribute set to "No" cannot be checkpointed by CXbatch.

SEt COPy_open_files = <**Yes**|**No**> **queue**

Sets the status of the copy_open attribute of a CXbatch batch queue. The queue named as a parameter of the command must already exist.

The copy_open attribute for a batch queue determines whether or not the open regular files of a request are copied to and from the checkpoint directory on checkpoint and restart.

SEt Import_dir = {*Yes,No,Available*} *queue*

Changes the **Import_dir** attribute for a *queue*. The queue must already exist. If the **Import_dir** attribute is set to *Yes*, any job submitted to this queue is run in the user's current working directory, unless the job specifically requests not to be imported. If the **Import_dir** attribute is set to *Available*, only jobs that specifically request to be imported are imported. If it is set to *No*, jobs are run in the home directory. When importing, if a job is executed on a remote machine, CXbatch tries to access the local directory using NFS mounts. **NOTE:** CXbatch will make temporary NFS mounts into the /tmp filesystem. Care should be taken that any automatic clean-up operations on the /tmp filesystem do not traverse NFS mount points.

SEt MAXimum Request_priority = *priority queue*

Set a limit on request priorities on a per queue basis. Requests queued with a priority higher than the queue's maximum will have their priority lowered to the maximum.

The following commands are used to set the limits batch queues impose on their requests. Every batch queue on the local host have each of the following limits associated with it at all times. If a request already in the queue has asked for more than a new limit, it is given a grandfather clause and allowed to retain its original limits. A request which specifies a particular limit may only enter a batch queue if the queue's corresponding limit is greater than or equal to the request's limit. See the section on **LIMITS** below for more information on the syntax on the various limit arguments. These commands can only be used by CXbatch managers.

SEt CORefile_limit = (*limit*) *queue*

Set a per-process maximum core file size *limit* for a batch queue against which the per-process maximum core file size limit for a request may be compared.

SEt DAta_limit = (*limit*) *queue*

Set a per-process maximum data segment size *limit* for a batch queue against which the per-process maximum data segment size limit for a request may be compared.

SEt NIce_value_limit = *nice-value queue*

Set the UNIX *nice-value* limit for a batch queue, against which the *nice* value for a request may be compared. A request specifying a *nice-value* may only enter a batch queue if the queue's nice value is numerically less than (more willing to allow access to

the CPU) or equal to the request's *nice* value. *nice-value* is an integer preceded by an optional negative sign.

SEt PER_Process Cpu_limit = (*limit*) *queue*

Set a per-process maximum CPU time *limit* for a batch queue against which the per-process maximum CPU time limit for a request may be compared.

SEt PER_Process Permfile_limit = (*limit*) *queue*

Set a per-process maximum permanent file size *limit* for a batch queue against which the per-process maximum permanent file size limit for a request may be compared.

SEt STack_limit = (*limit*) *queue*

Set a per-process maximum stack segment size *limit* for a batch queue against which the per-process maximum stack segment size limit for a request may be compared.

SEt Working_set_limit = (*limit*) *queue*

Set a per-process maximum working set size *limit* for a batch queue against which the per-process maximum working set size limit for a request may be compared.

PIPE QUEUE CONFIGURATION

The following commands are used to set the operating parameters for pipe queues. They are only available to CXbatch managers.

ADd DEStination = *destination queue*
ADd DEStination = (*destination* [, *destination* ...]) *queue*

The specified *destination(s)* are added as valid destinations for a pipe queue named *"queue"*.

DElete DEStination = *destination queue*
DElete DEStination = (*destination* [, *destination* ...]) *queue*

Delete the mappings from the pipe queue *queue* to the *destination* queues. All requests from the named *queue* being transferred to a deleted *destination* complete normally. If all *destinations* for a pipe *queue* are deleted in this manner, the pipe *queue* is effectively stopped.

SEt DESTination = *destination queue*
SEt DESTination = (*destination* [, *destination* ...]) *queue*

Associate one or more *destination* queues with a particular pipe *queue*.

SEt PIpe_client = (*client*) *queue*

Associate a *pipe client* with a pipe *queue*. *client* should consist of the absolute path name to the program binary followed by any arguments required by the program.

DESIGNATING MANAGERS AND OPERATORS

The following commands are used to specify CXbatch managers and operators. They can only be used by CXbatch managers.

A *manager* specification consists of an account name specification, followed by a colon, followed by either the letter *m* or the letter *o*. There are four ways to specify an account name:

local_account_name
[*local_user_id*]
[*remote_user_id*]@ *remote_machine_name*
[*remote_user_id*]@ [*remote_machine_mid*]

(The square brackets are literal characters used to indicate a uid or mid.) If the account name specification is followed by *:m*, the account is designated as a CXbatch *manager* account, capable of using all *qmgr* commands. If the account name specification is followed by *:o*, the account is

designated as a CXbatch *operator* account, capable of only using those commands appropriate for a CXbatch *operator*. The *root* account always has full privileges.

ADd Managers *manager ...*

The specified *manager(s)* are added to the list of authorized CXbatch managers with privileges as specified.

DElete Managers *manager ...*

The specified *manager(s)* are deleted from the list of authorized CXbatch managers.

SEt MANagers *manager ...*

The list of authorized CXbatch managers is set to the specified *manager(s)*.

GENERAL CXBATCH MANAGEMENT

The following commands are used to set the general operating parameters of CXbatch. Only CXbatch managers can use these commands.

SEt ACC_logfile *filename*

Change the file being used for CXbatch batch accounting.

SEt AId_mask = *mask*

Set the activity ID mask. Generally, this mask is the same as the spacing between aids in */etc/activities*. The following equation is used to determine the activity ID of the CXbatch job:

$$job\_aid = submitter\_aid - (submitter\_aid \ \% \ aid\_mask) + queue\_aid\_offset$$

where % is the modulus (remainder) function.

Warning: As shipped, the aid mask is one. When the mask is one, the above equation simplifies to $job\_aid = submitter\_aid + queue\_aid\_offset$. In this case, if a CXbatch job submits another CXbatch job, the second job has an activity ID of $submitter\_aid + queue1\_aid\_offset + queue2\_aid\_offset$. This is generally not desirable! Setting the activity ID mask to the spacing in */etc/activities* prevents this from happening.

SEt CHEckpoint_directory *directory*

Specify the pathname of the checkpoint directory. All checkpoint files created by CXbatch after this command is issued will be placed in *directory*. Existing checkpoint files will not be moved. The *directory* must exist and be a valid directory.

SEt DEBug *level*

Set the debug *level*. The following values are valid:

0 No debug
1 Minimum debug
2 Maximum debug

SEt DEFault Batch_request Priority *priority*

Set the default intra-batch-queue *priority*. This is *not* the UNIX execution time priority. This is the priority used if the user does not specify an intra-queue priority parameter on the *qsub*(1) command.

SEt DEFault Batch_request Queue *queue*

Set the default batch *queue*. This is the queue used if the user does not specify a queue parameter on the *qsub*(1) command.

SEt DEFault DEStination_retry Time *retry_time*

Set the default number of hours that can elapse during which time a pipe queue destination can be unreachable, before being marked as completely failed.

SEt DEFault DEStination_retry Wait *interval*

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

SEt Global Per_user Run_limit = *run-limit*

Set the *global per-user run-limit* of the local system. The *global per-user run-limit* is the maximum number of requests that any given user can have running on the local system at any given time. A *global per-user run-limit* of 0 turns off the enforcement of this limit.

SEt MAIl *userid*

Specify the *userid* used to send CXbatch mail.

SEt NO_Default Batch_request Queue

Indicate that there is to be no default batch request queue.

SEt SHEll_strategy FIxed = (*shell*)

Specify that *shell* should be used to execute all batch requests. *shell* must be the absolute path name of a command interpreter. (See the **SHELL STRATEGIES** section below for more information.)

SEt SHEll_strategy FRee

Specify that the *free* shell strategy should be used to execute all batch requests. The *free* shell strategy duplicates the shell choice that would have been made if the batch request script had been executed interactively. Under this strategy, the user's *login shell* is allowed to determine the shell to be used to execute the batch request. The user's *login shell* is the shell named within the user's entry in the password file (see *passwd*(4)). (See the SHELL STRATEGIES section below for more information.)

SEt SHEll_strategy Login

Specify that the *login* shell strategy should be used to execute all batch requests. Under the *login* shell strategy, the user's login shell is used to execute the batch request. The login shell is the shell named in the password file (see *passwd*(4)). (See the **SHELL STRATEGIES** section below for more information.)

GENERAL CXBATCH OPERATION

The following commands are used for starting and shutting down CXbatch. They can be used by CXbatch managers or operators.

SHUtdown [**Force**] [*seconds*]

Shutdown CXbatch on the local host.

Each running checkpointable batch request is checkpointed, and, if the checkpoint succeeds, terminated. Successfully checkpointed requests will be restarted from their checkpointed state when CXbatch is rebooted.

After running requests are checkpointed, a **SIGTERM** signal is sent to each process of each request presently running. After the specified number of *seconds* of real time have elapsed, a **SIGKILL** signal is sent to all remaining processes for each request. If a *seconds* value is not specified, the delay is sixty seconds. Unlike **ABort Queue**, **SHUtdown** requeues all of the requests it kills, provided that the initial **SIGTERM** signal is caught or ignored by the running request.

When the optional **force** keyword is present CXbatch ignores any checkpoint errors

incurred during the shutdown.

STArt Cxbatch [**Disabled**] [**Stopped**]

Start CXbatch on the local host. This command will fail if CXbatch is currently running on the local host.

When the optional **disabled** keyword is present, CXbatch is started with all queues disabled. Similarly, when the optional **stopped** keyword is present, CXbatch is started with all queues stopped.

QUEUE OPERATION

The following commands are used in operating queues in CXbatch. They can be used by CXbatch managers or operators.

ABort Queue *queue* [*seconds*]

All requests in the named queue that are currently running are aborted as follows. A SIGTERM signal is sent to each process of each request presently running in the named *queue*. After the specified number of *seconds* of real time have elapsed, a **SIGKILL** signal is sent to all remaining processes for each request running in the named *queue*. If a *seconds* value is not specified, the delay is sixty seconds. All requests aborted by this command are deleted, and all output files associated with the requests are returned to the appropriate destination.

DIsable Queue *queue*

Prevent any more requests from being placed in this queue.

ENable Queue *queue*

If the queue is already enabled, this command has no effect. Otherwise, the queue is enabled to accept new requests.

MOVe Queue *queue1 queue2*

Move all requests currently in *queue1* to *queue2*. The request is moved regardless of any queue limit violations, access restrictions, or attribute violations.

Purge Queue *queue*

All queued requests are dropped from the queue and are irretrievably lost. Running requests in the *queue* are allowed to complete.

STArt Queue *queue*

If the queue is already started, then nothing happens. Otherwise, the queue is started and requests in the queue are eligible for selection.

STOp Queue *queue*

Any requests in the queue that are currently running are allowed to complete. All other requests are "frozen" in the queue. New requests can still be submitted to the queue, but are "frozen" like the other requests in the queue.

QUEUE OPERATING PARAMETERS

The following commands are used to set the operating parameters of CXbatch queues. They can be used by CXbatch managers or operators.

SEt PER_User Run_limit = *run-limit queue*

Change the *per-user run-limit* of a CXbatch batch queue. The *per-user run-limit* determines the maximum number of requests that any given user can have running in the queue at any given time. A *per-user run-limit* of 0 turns off the enforcement of this limit.

SEt Run_limit = *run-limit queue*

Change the *run-limit* of a CXbatch batch or pipe queue. The *run-limit* determines the maximum number of requests that are allowed to run in the queue at any given time.

SEt SHAre_policy Fixed = *user queue*

Change the *share-policy* of a CXbatch batch queue. A queue with a fixed share policy will charge the CPU usage of jobs run in that queue to *account*. There are two ways to specify a user: *username* or [*id*]. (The square brackets are literal characters used to indicate a uid.)

SEt SHAre_policy User *queue*

Change the *share-policy* of a CXbatch batch queue. A queue with a user share policy will charge the CPU usage of jobs run in that queue to the account from which the job was submitted.

REQUEST OPERATIONS

The following commands operate on CXbatch requests. They can only be executed by CXbatch managers or operators.

MOVe Request *requestid ... queue*

Move the request(s) named by the *requestid(s)* to the named queue. The request is moved regardless of any queue limit violations, access restrictions, or attribute violations.

RUn Request *requestid ...*

Force the request(s) named by the *requestid(s)* to begin executing immediately. If running the request would exceed the current run limit of the queue, then the queue's run limit will be increased by one until the request finishes executing.

USER REQUEST OPERATIONS

The following commands operate on CXbatch requests. These commands may be used by non-privileged user in dealing with their own requests. Otherwise, they are limited to use by CXbatch managers or operators.

CHKpnt Request *requestid ...*

Checkpoint the request(s) named by the *requestid(s)*. The state of the named batch request(s) is saved into a set of checkpoint files stored in the *checkpoint directory*.

DElete Request *requestid ...*

Delete the request(s) named by the *requestid(s)*. This command can delete both running and non-running requests. If a request is running, then all processes of the request are sent a SIGKILL signal.

HOLd Request *requestid ...*

Makes the specified request(s), named by *requestid(s)*, ineligible for running. The request(s) must be in the *queued* state prior to being held. The **RELease Request** command removes the effect of **hold**.

If a request is held by an operator, only an operator can release it.

MODify Request *field* = *value requestid ...*

Change the field of the request(s) specified by *requestid(s)* to be *value*.
The legal values for *field* are:

Priority - change the intra-queue request priority. A user can only decrease a request's priority, *operator* privileges are required to raise a request's priority.

MOVe My_request *requestid ... queue*

Move your request(s) named by the *requestid(s)* to the named queue. The request is not moved if any queue limits, access restrictions, or attributes would have prevented the

request from being submitted to the new queue.

RELease Request *requestid* ...

Makes the specified request(s), named by *requestid(s)*, eligible for running. The request(s) must be in the *holding* state prior to being released.S

RESume Request *requestid* ...

Resume execution of suspended requests. Resumed requests start out in the *queued* state. Once the resumed request is about to enter the *running* state, it will be restarted from its checkpointed state instead of being re-run in its entirety.

SUspend Request *requestid* ...

Temporarily freeze execution of the named requests. The requests are checkpointed and terminated. Of course, a request that fails to checkpoint will continue to execute. Only checkpointable requests may be suspended in this manner.

QUEUE TYPES

CXbatch supports two different queue types that provide two very different functions. These two queue types are known as *"batch"* and *"pipe"*.

The queue type of *"batch"* can only be used to execute CXbatch batch requests. Only CXbatch batch requests created by the *qsub*(1) command can be placed in a *batch* queue.

Queues of type *"pipe"* are used to send CXbatch requests to other *"pipe"* queues or *batch* queues. In general, *"pipe"* queues act as the mechanism that CXbatch uses to transport *"batch"* requests to distant queues on other remote machines. It is also perfectly legal for a *"pipe"* queue to transport requests to queues on the same machine.

When a *"pipe"* queue is defined, it is given a destination set, that defines the set of possible destination queues for requests entered in that *pipe* queue. In this manner, it is possible for a *"batch"* request to pass through many pipe queues on its way to its ultimate destination, that must eventually be a queue of type *"batch."*

Each *pipe* queue has an associated server. For each request handled by a *pipe* queue, the associated server is spawned which selects a queue destination for the request being handled, based upon the characteristics of the request and upon the characteristics of each queue in the destination set defined for the pipe queue.

Because a different server can be configured for each *pipe* queue, and *"batch"* queues can be endowed with the *"pipeonly"* attribute that will only admit requests queued via another *pipe* queue, it is possible for respective CXbatch installations to use *pipe* queues as a request class mechanism, placing requests that ask for different resource allocations in different queues, each of which can have different associated limits and priorities.

It is also completely possible for a pipe client (pipe queue server), when handling a request, to discover that no destination queue will accept the request, for various reasons that can include insufficient resource limits to execute the request or a lack of a corresponding account or privilege for queuing at a remote queue. In such circumstances, the request is deleted, and the user is notified by mail (see *mail*(1)).

SHELL STRATEGIES

The execution of a batch request requires the creation of a shell process to interpret the shell script that defines the batch request. On many UNIX systems, there is more than one shell available (e.g., */bin/csh*, */bin/ksh*, */bin/sh*). To deal with this problem, CXbatch allows a shell pathname to be specified when a batch request is first submitted (*qsub* option *-s*).

If no particular shell is specified for the execution of the request, CXbatch must have some other means of deciding which shell to use when spawning the request. The solution to this dilemma has been to equip CXbatch with a batch request shell strategy that can be configured as necessary

by the local system administrators.

The batch request shell strategy determines the shell to be used when executing a batch request on the local host that fails to identify any specific shell for its execution. Three such shell strategies can be configured for CXbatch, and they are known by the names of *fixed*, *free*, and *login*.

A shell strategy of *fixed* causes the request to be run by the *fixed shell*, the path name of which is configured by the system administrator. Thus, a particular CXbatch installation may be configured with a *fixed* shell strategy where the default shell used to execute all batch requests is defined as the Bourne shell.

A shell strategy of *free* causes the user's login shell (as defined in the password file), to be executed. This shell is, in turn, given a path name to the batch request shell script, and it is the user's login shell that actually decides which shell should be used to interpret the script. The *free* shell strategy therefore runs the batch request script exactly as would an interactive invocation of the script and is the default CXbatch shell strategy.

The third shell strategy of *login* causes the user's login shell (as defined in the password file) to be the default shell used to interpret the batch request shell script.

The strategies of *fixed* and *login* exist for host systems that are short on available free processes. In these two strategies, a single shell is executed, and that same shell is the shell that executes all of the commands in the batch request script (barring shell exec operations in any user startup files: *.profile*, *.login*, *.cshrc*).

The shell strategy as configured for any particular host can always be determined by the CXbatch *qlimit* command.

LIMITS

CXbatch supports many batch request resource limit types that can be applied to a CXbatch batch queue. The configurability of these limits allows a CXbatch manager to set batch queue-specific resource limits that all batch requests in the queue must adhere to.

The syntax of a *limit* in commands of the form **SEt Some_limit** = (*limit*) *queue* is quite flexible.

For *finite* CPU time limits, the acceptable syntax is as follows:

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

Whitespace can appear anywhere between the principal tokens, with the exception that no whitespace can appear around the decimal point. **NOTE:** The *milliseconds* value may be ignored if the system does not support such granularity.

Example time *"limit-values" are:*

1234 : 58 : 21.29	– 1234 hrs 58 mins 21.290 secs
12345	– 12345 seconds
121.1	– 121.100 seconds
59:01	– 59 minutes and 1 second

For all other *finite* limits (with the exclusion of the *nice-value*), the acceptable syntax is:

.*fraction* [*units*]

or

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

where the *"integer"* and *"fraction"* tokens represent strings of up to eight decimal digits,

denoting the obvious values. In both cases, the *"units"* of allocation may also be specified as one of the case insensitive strings:

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

In the absence of any *"units"* specification, the units of bytes are assumed.

For all limit types with the exception of the ***nice-value***, it is possible to state that no limit should be applied. This is done by specifying a *"limit"* of "unlimited","UNLIMITED", or any initial substring thereof.

The complications caused by batch request resource limits first show up when queuing a batch request in a batch queue. This operation is described in the following paragraphs.

If a batch request specifies a limit that cannot be enforced by the underlying UNIX implementation, the limit is ignored, and the batch request operates as though there were no limit (other than the obvious physical maximums) placed upon that resource type. (See the *"qlimit"*(1) command to find out what limits are supported by a given machine.)

For each remaining finite limit that can be supported by the underlying UNIX implementation that is *not* a CPU *"time-limit"* or UNIX *"nice-value"*, the *"limit-value"* is internally converted to the units of bytes or words, whichever is more appropriate for the underlying machine architecture.

As an example, a working set size limit value of 321 megabytes would be interpreted as 321 x 2^20 bytes, provided that the underlying machine architecture was capable of directly addressing single bytes. Thus the original limit coefficient of 321 would become 321 x 2^20. On a machine that was only capable of addressing words, the appropriate conversion of 321 x 2^20 bytes / *# of-bytes-per-word* would be performed.

If the result of such a conversion would cause overflow when the coefficient was represented as a signed-long integer on the supporting hardware, the coefficient is replaced with the coefficient of: of 2^*N*-1 where *N* is equal to the number of bits of precision in a signed long integer. For typical 32-bit machines, this default extreme limit would therefore be 2^*31*-1 bytes. For word addressable machines in the supercomputer class supporting 64-bit long integers, the default extreme limit would be 2^*63*-1 words.

Lastly, some implementations of UNIX reserve coefficients of the form: 2^*N*-1 as synonymous with infinity, meaning no limit is to be applied. For such UNIX implementations, CXbatch further decrements the default extreme limit so as to not imply infinity.

The identical internal conversion process as described in the preceding paragraphs is also performed for all finite limit-values specified with a particular batch request.

After each applicable request limit has been converted as described above, the resulting limit is then compared against the corresponding limit as configured for the destination batch queue. If the corresponding batch queue limit for all batch request limits is defined as unlimited, or is greater than or equal to the corresponding batch request limit, the request can be successfully queued, provided that no other anomalous conditions occur. For requests that ask for a limit of infinity, the corresponding queue limit must also be configured as infinity.

These resource limit checks are performed irrespective of the batch request arrival mechanism, either by a direct use of the *"qsub"*(1) command, or by the indirect placement of a batch request into a batch queue via a *"pipe"* queue. It is impossible for a batch request to be queued in a CXbatch batch queue if *"any"* of these resource limit checks fail.

Finally, if a request fails to specify a *"limit"* for a resource limit type that is supported on the execution machine, the corresponding *"limit"* as configured for the destination queue becomes the *"limit"* for the request.

Upon the successful queuing of a request in a batch queue, the set of limits under which the request will execute is frozen and will not be modified by subsequent *qmgr*(8) commands that alter the limits of the containing batch queue.

DIAGNOSTICS

qmgr returns an exit status describing what the last *qmgr* command did. If there were no errors, the exit status is zero. If one or more of the operations failed, the exit status is the number of operations that failed. If a fatal error occurs and command completely fails, (ex, a syntax error), the exit status is one of the codes defined in *<sysexits.h>*.

EX_USAGE	The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.
EX_NOHOST	The host specified did not exist.
EX_OSFILE	Some batch system file does not exist, cannot be opened, or has an error.
EX_TEMPFAIL	Temporary failure; retry the request at a later time.

SEE ALSO

qdel(1), qjlist(1), qlimit(1), qstat(1), qsub(1), qmapmgr(8)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qrun – force CXbatch request(s) to run.

SYNOPSIS

qrun *request-id ...*

DESCRIPTION

The *qrun* command forces each CXbatch request whose ***request-id*** is listed on the command line to immediately begin executing.

If spawing a force-run request would cause the run limit of the queue to be exceeded, CXbatch increases the run limit by one while the forced-run request is running.

CXbatch operator privileges are required to use this command. Under ConvexOS/Secure the user must have the **batch** subsystem authorization in addition to CXbatch operator privileges. The target request(s) must reside in batch queues.

DIAGNOSTICS

qrun returns an exit status describing what it did. If there were no errors, the exit status is zero. If one or more of the requests were not run, the exit status is the number of requests that weren't deleted. If a fatal error occurs and none of the requests are run (e.g., a syntax error), the exit status is one of the codes defined in *<sysexits.h>*.

EX_TEMPFAIL Temporary failure; retry the transaction at a later time.

EX_USAGE The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.

EX_OSFILE Some batch system file does not exist, cannot be opened, or has an error.

EX_NOPERM You did not have sufficient permission to perform the operation.

EX_SOFTWARE
Too many request ids were specified.

SEE ALSO

qmgr(8)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qsa – show accounting information on CXbatch requests

SYNOPSIS

qsa -r [**-QU**] [**-q** *queue*] [**-u** *username*] [*acct-file*]
qsa -x [**-QU**] [**-q** *queue*] [**-u** *username*] [*acct-file*]
qsa -a [**-QU**] [**-q** *queue*] [**-u** *username*] [*acct-file*]
qsa -s [**-QU**] [**-q** *queue*] [**-u** *username*] [*acct-file*]

DESCRIPTION

qsa processes CONVEX CXbatch accounting file and outputs data about requests. By default, *qsa* reads from */usr/adm/batchacct*. Another file may be optionally specified on the command line.

qsa operates in one of four modes specified by the following command line flags:

r Raw mode. Accounting records are formatted and written to the standard output.

x Extended mode. Accounting records are processed, formatted and written to the standard output. This mode differs from raw mode in that time spent waiting in queue, time spent executing, and turnaround time are calculated and all time values are converted to ASCII using *ctime(3)*.

a Averaging mode. Accounting records are processed and averages for turnaround time, time waiting in queue, execution time, CPU time in user mode, CPU time spent in system, and I/O operations performed are written.

s Summing mode. Accounting records are processed and totals for turnaround time, time waiting in queue, execution time, CPU time in user mode, CPU time spent in system, and I/O operations performed are written.

At least one of these mode flags must appear.

Constraint flags control which records *qsa* processes. If none are present, *qsa* processes all records in the accounting file. Using these flags, the user may specify records by queue, by user, or by both. The following flags specify constraints:

Q All queues. Accounting records are processed grouped by each queue appearing in the accounting file. This flag may not appear with the **-q** flag.

q *queue*
Specific queues. Accounting records are processed grouped by queues specified in one or more occurrences of this flag. This flag may not appear with the **-Q** flag.

U All users. Accounting records are processed grouped by each user appearing in the accounting file. This flag may not appear with the **-u** flag.

u *username*
Specific users. Accounting records are processed grouped by users specified in one or more occurrences of this flag. This flag may not appear with the **-U** flag.

DIAGNOSTICS

qsa returns an error status describing what it did. If there were no errors, the exit status is zero. If a fatal error occurs then the exit status is one of the codes defined in *<sysexits.h>*.

EX_USAGE One of the following incorrect usages was specified: More that 20 occurrences of the **-q** or **-u** options, respectively, none or more than one of the mode options, **-r**, **-x**, **-a**, **-s**, was specified, the **-Q** option was specified with the **-q** option, the **-U** option was specified with the **-u** option, or an invalid option was specified.

EX_DATAERR The accounting file was not a regular file.

EX_NOINPUT Cannot open accounting file or unable to access CXbatch database files.

SEE ALSO

qmgr(8), ctime(3)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qsnapshot – dump the current CXbatch queue or network configuration

SYNOPSIS

qsnapshot [**-m**]

DESCRIPTION

qsnapshot dumps the current CXbatch queue or networking configuration as a series of *qmgr*(8) or *qmapmgr*(8) commands. The resulting output is suitable as input to a subsequent *qmgr*(8) or *qmapmgr*(8) command. By default, *qsnapshot* dumps the CXbatch queue configuration. The **-m** option causes the *qmapmgr*(8) network database to be dumped instead.

The *qsnapshot* command is useful for copying the CXbatch configuration from one system to another or (by editing the output) duplicating a complex queue(s) on the local system.

Recreating CXbatch Databases

There are no *qmgr*(8) or *qmapmgr*(8) commands which clear out the old configurations from their respective database. If it is necessary to do this, shutdown CXbatch and use one or both of the commands given below. Recreating the qmapmgr database will require you to recreate the qmgr database if any of the MID to machine mappings are changed. Some MIDs may be referenced in the qmgr database. These commands should be executed with caution, as should any recursive *rm*(1) commands containing wildcard characters.

To empty the *qmgr*(8) database:

*rm -f /usr/spool/nqs/private/root/database/**

To empty the *qmapmgr*(8) database:

*rm -f /etc/nmap/**

DIAGNOSTICS

qsnapshot returns an exit status describing what it did. If there were no errors, the exit status is zero. If a fatal error occurs, then the exit status is one of the codes defined in *<sysexits.h>*.

EX_USAGE The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.

EX_OSFILE Some batch system file does not exist, cannot be opened, or has an error.

SEE ALSO

qmgr(8), qmapmgr(8)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.

NAME

qwatch – watch status of CXbatch queue(s)

SYNOPSIS

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

DESCRIPTION

qwatch monitors the status of CONVEX CXbatch and periodically reports statistics about queues and/or requests.

Initially, *qwatch* displays queue headers for the first five queues. If there are more than five queues, additional pages can be displayed by typing the number of the desired page. Only five pages (25 queues) can be display by *qwatch.*

By typing the letter (a-y) corresponding to a queue, you can monitor the activity of that queue. The first page of the requests within the selected queue is displayed along with the queue header. If there are more that 16 requests in the queue, additional pages can be displayed by typing the number of the desired page. Only nine pages (144 requests) can be displayed by *qwatch.*

The command line switches, which may be given in any order, are:

-c *count* If *count* is positive, *qwatch* automatically exits after *count* screen updates have been completed. If *count* is 0, *qwatch* keeps updating until stopped. The default value for *count* is 0.

-i *interval* Specifies how many seconds to wait between each screen update. The default is five seconds.

The following keys are interpreted by *qwatch*:

<SPACE>

<RETURN> Force *qwatch* to update the screen and restart the timer.

<CTRL-L> Replot the screen.

[**1-9**] Select a display page. (Only pages with information on them may be selected.)

[**a-y**] Select a queue. (Only available from queues display page.)

- Return to queues display page from requests display page.

The *SIGINT* signal, usually generated by **<CTRL-C>** or **<DEL>**, is used to exit *qwatch.* Additionally, *qwatch* can be interrupted and later resumed with the *SIGTSTP* signal, which is typically generated by **<CTRL-Z>** from *csh.*

QUEUE STATE INFORMATION

Refer to the qstat(1) manpage for an explanation of the information displayed by *qwatch.*

DIAGNOSTICS

qwatch returns an exit status describing what it did. If there were no errors, the exit status is zero. If a fatal error occurs, then the exit status is one of the codes defined in *<sysexits.h>*.

EX_UNAVAILABLE
The terminal does not support the capabilities necessary for *qwatch* to work properly.

EX_USAGE The command was used incorrectly, e.g., with the wrong number of arguments, a bad flag, a bad syntax in a parameter.

SEE ALSO

qstat(1), syspic(8)

NOTES

CXbatch is an optional product; for more information, contact your CONVEX sales representative.