CONVEX CXbatch
Programmer's Reference

Document No. 710-004430-002

First Edition, Rev. 1
April 1990

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX CXbatch Programmer's Reference
Order No. DSW-186
First Edition, Rev. 1

Printed in the United States of America

CXbatch 1.1

Optional Product

CONVEX CXbatch permits users to submit jobs for batch execution in queues. The jobs may be submitted to and executed on either the local machine or a remote machine, depending on the batch system configuration. Different resource quotas may be defined on a queue-by-queue basis. Algorithms are used to schedule jobs for execution on the most lightly loaded processor. CXbatch software relies on two underlying utilities for configuration and control: *qmapmgr* and *qmgr*. Individual commands exist within CXbatch to submit, control, and check the status of job requests.

CONVEX CXbatch documentation:

CONVEX CXbatch Concepts
CONVEX CXbatch Master Index
CONVEX CXbatch Programmer's Reference
CONVEX CXbatch System Management Utilities Reference
CONVEX CXbatch System Manager's Guide
CONVEX CXbatch User's Guide

List of Entries:

batch-acct(5)	*pipeclient*(8)	*qjlist*(1)
qmgr(8)	*qsub*(1)	*ssp*(1)
list(1)	*qdel*(1)	*qlimit*(1)
remove(1)	*submit*(1)	*nqsdaemon*(8)
qmapmgr(8)	*qstat*(1)	*sbatch*(1)

NAME

list – batch queue examination program (obsoleted by qstat)

SYNOPSIS

list [**–n** *queue*]

SPECIAL NOTE

This command was part of an older batch system. It is not part of CXbatch. There is, however, a skeleton program shipped with the first release of CXbatch that accepts this command and converts it to the equivalent CXbatch command, *qstat*.

This command will not be shipped with future releases of CONVEX UNIX or CXbatch. You should learn the CXbatch commands as soon as possible.

DESCRIPTION

list examines the queuing area used by CXbatch for executing jobs and reports the status of the specified jobs or all jobs associated with a user. Invoked without any arguments, *list* reports on any jobs currently in the queue. A **–n** flag may be used to specify a particular queue, otherwise the default queue is used. In the old *list* command, if a + argument was supplied, *list* would display the batch queue until it emptied. Supplying a number immediately after the + sign indicated that *list* should sleep *n* seconds in between scans of the queue. This skeleton program silently ignores the + option. All other arguments supplied were interpreted as user names or job numbers to select only those jobs of interest. This skeleton program silently ignores user names and job numbers.

For each job submitted (i.e., invocation of *submit*(1)), *list* reports the user's name, current rank in the queue, the name of the job, the job identifier (a number which may be supplied to *remove*(1) for removing a specific job), and other information. This output is different under CXbatch. See *qstat*(1) for more information. Jobs may be executed by any machine in the CXbatch system; the *list* skeleton program only shows jobs on the local host.

Job ordering is dependent on the algorithm used to scan the queuing directory; it is no longer FIFO (First In First Out).

SEE ALSO

remove(1), sbatch(1), ssp(1), submit(1) qdel(1), qjlist(1), qlimit(1), qstat(1), qsub(1)

NOTES

CXbatch is an optional product; for more information, 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-id* is 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, then the specified signal is sent instead of the SIGKILL signal to any running request whose *request-id* is listed on the command line. *signo* can be either the signal number or the signal name. The signal names are as given in */usr/include/signal.h*, stripped of the common SIG prefix. In the absence of the **-k** and *-signo* flags, *qdel* will not delete a *running* CXbatch request.

To delete or signal a CXbatch request, the invoking user must be the owner (the submitter of the request) or the superuser. The only exception to this rule occurs when the invoking user has CXbatch operator privileges as defined in the CXbatch manager database. Under these conditions, the invoker may specify the **-u** *username* flag which 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, then 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.

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 weren't 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), 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_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.

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,

then 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-id* is listed on the command line. To list the commands in a CXbatch request, the invoking user must be the owner (the submitter of the request). The only exception to this rule occurs when the invoking user is the *superuser* or has CXbatch operator privileges as defined in the CXbatch manager database. Under these conditions, the invoker may specify any batch request.

A CXbatch request is always uniquely identified by its *request-id*. A *request-id* is always of the form *seqno* or *seqno.hostname*, where *hostname* identifies the machine from whence the request was originally submitted, and *seqno* identifies the sequence number assigned to the request on the originating host. If the *hostname* portion of a *request-id* is omitted, then the local host is always assumed. The local host is searched for each given *request-id*, unless a different host 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 weren't 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), 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_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, then the information displayed is only relevant to 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, then 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.

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*, then 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 that is supported on the execution machine, then 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 as 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, then 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

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, then 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 this information 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, then 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, then 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, then the queue is said to be *stopped*. If the same situation exists with the difference that at least one request is running, then 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, then the queue is said to be *running*. If the same circumstances prevail with the exception that no requests are presently running in the queue, then 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*, then the queue is said to be *shutdown*. The queue states describing the second principal property of a queue are therefore respectively displayed as **STOPPED**, **STOPPING**, **RUNNING**, **INACTIVE**, and **SHUTDOWN**.

REQUEST STATE

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

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

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

DIAGNOSTICS

qstat returns an exit status describing what it did. If there were no errors, the exit status is zero. If one or more of the queues weren't 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), 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_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, then 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, then 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
–e – direct *stderr* output to stated destination
–eo – direct *stderr* output to the *stdout* destination
–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
–lc – establish per-process core file size limit
–ld – establish per-process data-segment size limits
–lf – establish per-process permanent-file size limits
–lF – establish per-request permanent-file space limits
–lm – establish per-process memory size limits
–lM – establish per-request memory space limits
–ln – establish per-process nice execution value limit
–ls – establish per-process stack-segment size limits
–lt – establish per-process CPU time limits
–lT – establish per-request CPU time limits
–lv – establish per-process temporary-file size limits
–lV – establish per-request temporary-file space limits
–lw – establish per-process working set 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
–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
–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 aren't equivalenced in the */etc/hosts.equiv* file, then 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 ":", then goto step 7.
3. If the first non-whitespace character(s) of the current line is not "#" or "$!", then goto step 8.
4. If the second non-whitespace character in the current line is *not* the "@" character, or the character immediately following the second non-whitespace character in the current line is *not* a "$", then goto step 7.
5. If no "-" is present as the character *immediately* following the "@$" sequence, then goto step 8.
6. Process the *embedded* flag, stopping the parsing process upon reaching the end of the line, or upon reaching the first unquoted "#" or "$!" character(s).
7. Read the next *script file* line. Goto step 2.
8. End. No more *embedded* flags will be 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-request 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, then 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, then 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 timezone 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 timezone specification, the local timezone 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

–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, then 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 "/", then 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, then 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.

–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, then 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, and so the **–ke** flag can be specified which instructs CXbatch to leave any *stderr* output file produced by the request on the machine that actually *executed* the batch request.

This flag is meaningless if the **–eo** flag is specified and cannot be specified if an

explicit *machine* destination is given for the *stderr* parameter of the **–e** flag.

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

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

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

–lc *per-process corefile size limit*

Set a *per-process* maximum *core file size limit* for all processes that constitute the running batch request. If any process in the running request attempts to exit creating a core file whose size would exceed the maximum *per-process core file size limit* for the request, then the core file image of the aborting process is reduced to the necessary size by an algorithm dependent upon the underlying UNIX implementation.

Not all UNIX implementations support *per-process corefile size limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

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 *per-process corefile size limit*.

–ld *per-process data-segment size limit* [, *warn-limit*]

Set a *per-process* maximum and an optional warning *data-segment size limit* for all processes that constitute the running batch request. If any process in the running request exceeds the maximum *per-process data-segment size-limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process data-segment warning size limits*. When a warning limit is exceeded, a signal determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is composed of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes or otherwise escaped such that *qsub* and the shell will interpret the entire specification as a single-character string token. This caveat also applies when an embedded default **–ld** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-process data-segment size limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is

simply ignored.

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 *per-process data-segment size limit*.

–lf *per-process permanent-file size limit* [, *warn-limit*]
Set a *per-process* maximum and an optional warning *permanent-file size limit* for all processes that constitute the running batch request. If any process in the running request attempts to write to a permanent file such that the file size would increase beyond the maximum *per-process permanent-file size limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process warning permanent-file size limits*. When a warning limit is exceeded, a signal determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is composed of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes or otherwise escaped such that *qsub* and the shell will interpret the entire specification as a single-character string token. This caveat also applies when an embedded default **–lf** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-process permanent-file size limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

At the time of this writing, the author was unaware of any UNIX implementation that made a distinction at the **kernel** level, between *permanent* and *temporary* files. While it is certainly possible to construct a *pseudo-temporary* file by first creating it and then unlinking its pathname, the disk space allocated for such a file will be allocated from the same pool of disk space that all other UNIX files are allocated from. Furthermore, such a file will be subject to the same quota enforcement mechanisms, if any are provided by the underlying UNIX implementation, that all other UNIX files are created under.

For all UNIX implementations that do not support a distinction between *permanent*, and *temporary* files at the **kernel** level, this limit is interpreted as a *per-process file size limit* with the word *permanent* removed from the definition.

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 *per-process permanent-file size limit*.

–lF *per-request permanent-file space limit* [, *warn-limit*]
Set a *per-request* maximum and an optional warning cumulative *permanent-file space limit* for all processes that constitute the running batch request. If any process in the running request attempts to write to a permanent file such that the file space consumed by all permanent files opened for writing by all of the processes in the batch request would increase beyond the maximum *per-request permanent-file space limit* for the request, then all of the processes in the request will be terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-request warning permanent-file space limits*. When such a

warning limit is exceeded, a signal is delivered to one or more of the processes comprising the running request, depending upon the underlying UNIX implementation.

If a maximum limit (and optional warning limit) specification is composed of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes or otherwise escaped such that *qsub* and the shell will interpret the entire specification as a single-character string token. This caveat also applies when an embedded default **–lF** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-request permanent-file space limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

At the time of this writing, the author was unaware of any UNIX implementation that made a distinction at the **kernel** level, between *permanent*, and *temporary* files. While it is certainly possible to construct a *pseudo-temporary* file by first creating it, and then unlinking its pathname, the disk space allocated for such a file will be allocated from the same pool of disk space that all other UNIX files are allocated from. Furthermore, such a file will be subject to the same quota enforcement mechanisms, if any are provided by the underlying UNIX implementation, that all other UNIX files are created under.

For all UNIX implementations that do not support a distinction between *permanent*, and *temporary* files at the **kernel** level, this limit is interpreted as a *per-request file space limit*, with the word *permanent* removed from the definition.

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 *per-request permanent-file space limit*.

–lm *per-process memory size limit* [, *warn-limit*]

Set a *per-process* maximum and an optional warning *memory size limit* for all processes that constitute the running batch request. If any process in the running request exceeds the maximum *per-process memory size limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process warning memory size limits*. When a warning limit is exceeded, a signal determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is composed of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes or otherwise escaped such that *qsub* and the shell will interpret the entire specification as a single-character string token. This caveat also applies when an embedded default **–lm** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-process memory size limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

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 *per-process*

memory size limit.

–lM *per-request memory space limit* [, *warn-limit*]

Set a *per-request* maximum and an optional warning cumulative *memory space limit* for all processes that constitute the running batch request. If the sum of all memory consumed by all of the processes comprising the running request exceeds the maximum *per-request memory space limit* for the request, then all of the processes in the request will be terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-request warning memory size limits*. When such a warning limit is exceeded, a signal is delivered to one or more of the processes comprising the running request, depending upon the underlying UNIX implementation.

If a maximum limit (and optional warning limit) specification is composed of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes or otherwise escaped such that *qsub* and the shell will interpret the entire specification as a single-character string token. This caveat also applies when an embedded default –lM flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-request memory space limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

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 *per-request memory space limit*.

–ln *per-process nice value limit*

Set a *per-process nice value* for all processes comprising the running batch request.

At present, all UNIX implementations support the use of an integer called the *nice* value, which determines the *execution-time* priority of a process relative to all other processes in the system. By letting the user set a limit on the *nice* value for all processes comprising the running request, a user can cause a request to consume less (or more) of the CPU resource presented by the execution machine.

This is particularly useful when a user wishes to execute a CPU-intensive batch request on a machine running interactive processes. By setting a low *execution-time priority*, a user can make a long running batch request give way to more interactive processes during the daytime, while the coming of the nighttime hours with typically smaller process loads allows such a request to gain more and more of the CPU resource. In this way, long-running batch requests can be polite to their more transient, interactive neighbor processes.

The only quirk associated with this flag results from the peculiar choice of *nice* values, implemented by the standard UNIX implementations. 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*! Thus, a *nice value* limit specification of "*-ln -10*" is greedier than a *nice value* limit specification of "*-ln 0*".

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* value limit to

within the necessary range as appropriate.

Lastly, any *nice* value specified by this flag must be acceptable to the batch queue in which the request is ultimately placed. (See the section entitled **LIMITS** for more information.)

–ls *per-process stack-segment size limit* [, *warn-limit*]

Set a *per-process* maximum and an optional warning *stack-segment size limit* for all processes that constitute the running batch request. If any process in the running request exceeds the maximum *per-process stack-segment size limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process warning stack-segment size limits*. When a warning limit is exceeded, a signal determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is composed of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes or otherwise escaped such that *qsub* and the shell will interpret the entire specification as a single-character string token. This caveat also applies when an embedded default **–ls** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-process stack-segment size limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

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 *per-process stack-segment size limit*.

–lt *per-process CPU time limit* [, *warn-limit*]

Set a *per-process* maximum and an optional warning *CPU time limit* for all processes that constitute the running batch request. If any process comprising the running request exceeds the maximum *per-process CPU time limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process CPU warning time limits*. When a warning limit is exceeded, a signal determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is composed of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes or otherwise escaped such that *qsub* and the shell will interpret the entire specification as a single-character string token. This caveat also applies when an embedded default **–lt** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-process CPU time limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

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 *per-process CPU*

time limit.

–lT *per-request CPU time limit* [, *warn-limit*]

Set a *per-request* maximum and an optional warning cumulative *CPU time limit* for all of the processes that constitute the running batch request. If the sum of the CPU times consumed by all of the processes in the batch request exceeds the maximum *per-request CPU time limit* for the request, then all of the processes in the request are terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-request CPU warning time limits*. When such a warning limit is exceeded, a signal is delivered to one or more of the processes in the running request, depending upon the underlying UNIX implementation.

If a maximum limit (and optional warning limit) specification is composed of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes or otherwise escaped such that *qsub* and the shell will interpret the entire specification as a single-character string token. This caveat also applies when an embedded default **–lT** flag with its associated limit value(s) appears within the batch request *script file*.

Not all UNIX implementations support *per-request CPU time limits*. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

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 *per-request CPU time limit*.

–lv *per-process temporary file size limit* [, *warn-limit*]

Set a *per-process* maximum and an optional warning *temporary (volatile) file size limit* for all processes that constitute the running batch request. If any process in the running request attempts to write to a *temporary* file such that the file size would increase beyond the maximum *per-process temporary-file size limit* for the request, then that process is terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-process warning temporary-file size limits*. When a warning limit is exceeded, a signal determined by the underlying UNIX implementation is delivered to the offending process.

If a maximum limit (and optional warning limit) specification is composed of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes or otherwise escaped such that *qsub* and the shell will interpret the entire specification as a single-character string token. This caveat also applies when an embedded default **–lv** flag with its associated limit value(s) appears within the batch request *script file*.

At the time of this writing, no UNIX operating system known to the author supported a distinction at the **kernel** level between *permanent* and *temporary files*. Certainly, a *pseudo-temporary* file can be constructed by creating it and then unlinking its pathname. However, the file space allocated for such a file is allocated from the same pool of disk space that all other UNIX files are allocated from.

Until a mechanism is implemented in the **kernel** that knows about *permanent* and *temporary* files, this limit cannot be supported in the sense most useful for batch requests, namely the strict enforcement of disk quotas for *permanent* versus

temporary files.

Until such a time, this limit is simply ignored.

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 *per-process temporary-file size limit*.

–lV *per-request temporary file space limit* [, *warn-limit*]

Set a *per-request* maximum and an optional warning cumulative *temporary (volatile) file space limit* for all processes that constitute the running batch request. If any process in the running request attempts to write to a *temporary* file such that the file space consumed by all *temporary* files opened for writing by all of the processes in the batch request would increase beyond the maximum *per-request temporary-file space limit* for the request, then all of the processes in the request are terminated by a signal chosen by the underlying UNIX implementation.

The ability to specify an optional warning limit exists for those UNIX operating systems that support *per-request warning temporary-file space limits*. When such a warning limit is exceeded, a signal is delivered to one or more of the processes in the running request, depending upon the underlying UNIX implementation.

If a maximum limit (and optional warning limit) specification is composed of two or more tokens separated by whitespace characters, then the specification must be enclosed within double quotes or otherwise escaped such that *qsub* and the shell will interpret the entire specification as a single-character string token. This caveat also applies when an embedded default **–lV** flag with its associated limit value(s) appears within the batch request *script file*.

At the time of this writing, no UNIX operating system known to the author supported a distinction at the **kernel** level between *permanent* and *temporary files*. Certainly, a *pseudo-temporary* file can be constructed by creating it and then unlinking its pathname. However, the file space allocated for such a file is allocated from the same pool of disk space that all other UNIX files are allocated from.

Until a mechanism is implemented in the **kernel** that knows about *permanent* and *temporary* files, this limit cannot be supported in the sense most useful for batch requests, namely the strict enforcement of disk quotas for *permanent* versus *temporary* files.

Until such a time, this limit is simply ignored.

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 *temporary-file space limit*.

–lw *per-process working set size limit*

Set a *per-process* maximum *working set size limit* for all processes that constitute the running batch request.

Not all UNIX implementations support *per-process working set size limits*, and such a limit only makes sense in the context of a paged virtual memory machine. If a batch request specifies this limit, and the machine upon which the batch request is eventually run does not support the enforcement of this limit, then the limit is simply ignored.

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 *per-process working set size limit*.

–mb Send mail to the user on the originating machine when the request begins execution.

If the **–mu** flag is also present, then 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, then 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.

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

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

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

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

For a CXbatch request to be properly restarted after a CXbatch shutdown, the **–nr** flag must not be specified, and the spawned batch request shell must ignore **SIGTERM** signals (which is done by default). The spawned batch request shell also must not exit before the final **SIGKILL** arrives. Because the batch request shell is simply 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, then 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 "/", then 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, then 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 will have 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, then the user's environment variable set is searched for the variable QSUB_QUEUE. If this environment variable is found, then 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, then 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, then the default *request-name* assigned to the request is **STDIN**.

In all cases, if the *request-name* is found to begin with a digit, then 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, then 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.

–z Submit the batch request silently. If the request is submitted successfully, then 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, then */etc/profile* and the user's *.profile* file are read. If the C-shell is chosen, then */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, then 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 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, 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. In many instances, smaller limit values can result in a more favorable scheduling policy for a batch request.

The syntax used to specify a *limit-value* for one of the *limit-flags* (*–llimit-letter-type*), is quite flexible and describes values for two general limit categories. These two general categories respectively deal with time related limits, and those limits are not time related.

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

For all other *finite* limits (with the exclusion of the *nice limit-value* –ln), 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, then 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, then 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, then 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*, then 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, then 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, then 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, then 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 merely 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

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

remove, hold, unhold, move – manipulation of batch queue jobs

SYNOPSIS

remove **–n** *queue* [–] [*job #* ...] [*user* ...]

hold **–n** *queue* [–] [*job #* ...] [*user* ...]

unhold **–n** *queue* [–] [*job #* ...] [*user* ...]

move **–n** *src_queue* **–N** *dest_queue* [–] [*job #* ...] [*user* ...]

SPECIAL NOTE

These commands were part of an older batch system. They are not part of CXbatch. There are, however, skeleton programs shipped with the first release of CXbatch that accept these commands and convert them to the equivalent CXbatch commands. The equivalent CXbatch commands are *qdel, qmgr hold request, qmgr release request,* and *qmgr move my_request.*

These commands will not be shipped with future releases of CONVEX Unix or CXbatch. You should learn the CXbatch commands as soon as possible.

DESCRIPTION

remove removes a job, or jobs, from a batch queue, and kills the job if it is executing. Because the queue directories are protected from users, using *remove* is normally the only method by which a user may remove a job.

hold causes a job, or jobs, from a batch queue to be held in that queue indefinitely and never become active.

unhold enables a job, or jobs, from a batch queue that were marked as held in that queue to become active.

move moves a job, or jobs, from the source batch queue to the destination batch queue.

In the old batch system, if no arguments were given, these commands acted on the highest job in the queue owned by the user. In contrast, the skeleton programs require a job number or user name.

If the – flag is specified, *remove*, et. al. acts upon all jobs that a user owns. If the superuser employs this flag, all jobs are affected. The owner is determined by the user's login name.

Specifying a *user* name, or list of *user* names, causes *remove*, et. al. to attempt to act upon any jobs queued belonging to that *user* (or users). This form of invoking *remove* is useful only to the superuser.

A user may specify an individual job by specifying its job number. This number may be obtained from the *list*(1) program.

The commands announce the names of any jobs affected and are silent if there are no jobs in the queue that match the request list.

The **–n** *queue* option must be used to specify from which queue jobs should be affected.

SEE ALSO

list(1), sbatch(1), ssp(1), submit(1) qdel(1), qjlist(1), qlimit(1), qstat(1), qsub(1)

RESTRICTIONS

hold does not work on jobs that have begun execution.

NOTES

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

NAME

sbatch – show batch queue attributes

SYNOPSIS

sbatch [**-n** *queue*]

SPECIAL NOTE

This command was part of an older batch system. It is not part of CXbatch. There is, however, a skeleton program shipped with the first release of CXbatch that accepts this command and converts it to the equivalent CXbatch command, *qstat*.

This command will not be shipped with future releases of CONVEX Unix or CXbatch. You should learn the CXbatch commands as soon as possible.

DESCRIPTION

sbatch displays the resources and attributes associated with the queue name supplied. If no queue name is given, the specifics of the *long* queue are displayed.

The resources displayed represent job resources that may be required by a particular batch job.

SEE ALSO

list(1), remove(1), ssp(1), submit(1) qdel(1), qjlist(1), qlimit(1), qstat(1), qsub(1)

NOTES

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

NAME

ssp – show service point status

SYNOPSIS

/usr/convex/ssp [+[*n*]]

SPECIAL NOTE

This command was part of an older batch system. It is not part of CXbatch. There is, however, a skeleton program shipped with the first release of CXbatch that accepts this command and converts it to the equivalent CXbatch command, *qstat*.

This command will not be shipped with future releases of CONVEX Unix or CXbatch. You should learn the CXbatch commands as soon as possible.

DESCRIPTION

In the old batch system, *ssp* examined the *service point* files created by *spd*(8) and reported the current status for each service point in the distributed batch system. There are no service points in CXbatch. This skeleton program calls *qstat*, which shows the status of the queues on the local machine.

ssp invoked without any arguments reports the status once. If a + argument is supplied, *ssp* displays the status continually until interrupted from the keyboard. Supplying a number *n* immediately after the + indicates that *ssp* should sleep for *n* seconds in between scans.

SEE ALSO

list(1), remove(1), sbatch(1), submit(1) qdel(1), qjlist(1), qlimit(1), qstat(1), qsub(1)

NOTES

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

NAME

submit – submit batch queue jobs

SYNOPSIS

submit [**–n** *queue*] [**–j** *jobname*] [**–s**] [**–o** *path*] [**–m**] [**–r**] [*file*]

SPECIAL NOTE

This command was part of an older batch system. It is not part of CXbatch. There is, however, a skeleton program shipped with the first release of CXbatch that accepts this command and converts it to the equivalent CXbatch command, *qsub*.

This command will not be shipped with future releases of CONVEX Unix or CXbatch. You should learn the CXbatch commands as soon as possible.

DESCRIPTION

submit places commands in a batch queue for background execution. Commands are entered from standard input, or from *file*, if specified. If commands are entered from standard input, they must be terminated by EOF. In the old batch system, input could be terminated by a single "." on a line. This skeleton program does not accept a dot as EOF.

The **–n** option may be used to specify which queue the job should be submitted to. If no queue is named, the default queue is used.

In the old batch system, the default notification was to notify the user asynchronously in a manner similar to *write*(1). This skeleton program does not notify you when your job is complete. Any output sent to *stdout* or *stderr* by the job is written to a file in the user's current working directory. In the old batch system, the output file went to your home directory. The output file name has changed from *xxx.out.jobnum* to *xxx.ojobnum*.

Other options have the following meaning:

–m Send mail upon completion. In the old batch system, the mail included any output from the job. This skeleton program only sends you a message, not the output.

–o Specify an alternate path to place the output file. *submit* recognizes "." as the present directory. In the old batch system, the path could be a directory name. This skeleton program requires a filename.

–s Be silent. Do not notify the user when the job is completed.

–j Specify a name to be associated with the job. This name appears on queue listings and in any mail generated by the queue system.

–r Restart the job on reboot if the system goes down when it is executing.

EXAMPLES

```
% submit -j myjob -n s -m < comfile
% submit -n v comfile
% submit -r
make foobar > makelog
make clean
<EOF>
```

SEE ALSO

at(1), list(1), remove(1), sbatch(1), ssp(1) qdel(1), qjlist(1), qlimit(1), qstat(1), qsub(1)

BUGS

All environment variables are carried with the batch job; aliases and shell variables are not. Therefore, users should be aware that their "environments" can change if their job is executed on a machine other than the one from which it was submitted. Although the current working directory and the user's environment variables are exported with the job, other parts of the user's environment can vary from machine to machine. For instance, if the user has a different *.cshrc* file on each processor, then the job will use whichever resides on the machine where the job is executing. This could result in different non-environment variables and aliases. Also, the "~"

character may be expanded differently from machine to machine. Finally, the user's home directory path name can change from machine to machine. Be aware that, when accessing files, using the full path names (i.e., beginning with "/") or using the "~" to access files can yield different results.

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 does not currently provide any programs for analyzing 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)

NAME

nqsdaemon, netdaemon, logdaemon – CXbatch daemons

SYNOPSIS

/usr/lib/nqs/nqsdaemon

DESCRIPTION

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

nqsdaemon handles all local transactions, including submits, deletes, job scheduling, and system configuration. *netdaemon* handles all possibly remote transactions, including submits and file staging. *nqsdaemon* and *netdaemon* contact *logdaemon* when they need to print an error message. *logdaemon* sends the message to syslogd and notifies the batch managers if the error is fatal. See *qmgr*(8) for more information on defining CXbatch managers.

SEE ALSO

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

FILES

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

NOTES

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

NAME

pipeclient, pipeldav – CXbatch pipe clients

SYNOPSIS

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

DESCRIPTION

CXbatch supports pipe queues, which 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*, which 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 that corresponds to an entry in the */etc/hosts* file.
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 a program used by the system administrator or system operator to control CXbatch requests, queues, and the general CXbatch configuration at the local machine.

Definitions

A CXbatch *request* is a request by a user or user program to perform a function that requires a delay in servicing (e.g., after a certain time). 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 *daemon* is a process that is designed to run continuously, providing some service when needed. (See the QUEUE TYPES section below for more information concerning queues.) Lastly, a CXbatch *manager* identifies a person who is capable of changing any CXbatch characteristic on the local machine. A CXbatch *operator* identifies a person who can execute only the *operator commands* as a proper subset of all the commands provided by the *qmgr*(8) utility.

Commands

The following paragraphs describe the syntax of each *qmgr*(8) command. All command keywords are recognized regardless of uppercase or lowercase usage. Keyword characters shown in uppercase indicate the smallest possible abbreviation of the keyword for the particular command being described.

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.

CXbatch *operator* privileges are required to use this command.

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.

Full CXbatch *manager* privileges are required to use this command.

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

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

Full CXbatch *manager* privileges are required to use this command.

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

The specified *group(s)* are added to the access list for *queue*. There are two ways to specify a group:

group name
[*group id*]

Full CXbatch *manager* privileges are required to use this command.

ADd Managers *manager ...*

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

If the account name specification is followed by *:m*, then the account is designated as a CXbatch *manager* account, capable of using all *qmgr* commands. If the account name specification is followed by *:o*, then the account is designated as a CXbatch *operator* account, capable of only using those commands appropriate for a CXbatch *operator*.

Full CXbatch *manager* privileges are required to use this command.

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

The specified *user(s)* are added to the access list for *queue*. There are two ways to specify a user:

user name
[*user id*]

Full CXbatch *manager* privileges are required to use this command.

Create Batch_queue *queue* **PRiority** = *n* **PIpeonly**]
Run_limit = *n*]
Import_dir = {*Yes,No,Available*}]

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.

If **Import_dir** is set to *Yes*, then 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*, then only jobs that specifically request to be imported are imported. If it is set to *No*, then 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.

(See the QUEUE TYPES section below for more information.)

Full CXbatch *manager* privileges are required to use this command.

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. (See the **QUEUE TYPES** section below for more information.)

Full CXbatch *manager* privileges are required to use this command.

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.

Full CXbatch *manager* privileges are required to use this command.

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, then the pipe *queue* is effectively stopped.

Full CXbatch *manager* privileges are required to use this command.

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

The specified *group(s)* are deleted from the access list for *queue*. There are two ways to specify a group:

group name
[*group id*]

Full CXbatch *manager* privileges are required to use this command.

DElete Managers *manager* ...

The specified *manager(s)* are deleted from the list of authorized 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*]

If the account name specification is followed by *:m*, it is understood that the account is currently permitted to use all *qmgr* commands. If the account name specification is followed by *:o*, it is understood that the account is currently permitted to use only those

commands appropriate for an operator to use. The *root* account always has full privileges.

Full CXbatch *manager* privileges are required to use this command.

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.)

Full CXbatch *manager* privileges are required to use this command.

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.

CXbatch *operator* privileges are required to use this command to delete requests owned by another user.

DElete Users = *user queue*
DElete Users = (*user* , *user ...*]) *queue*
The specified *user(s)* are deleted from the access list for *queue*. There are two ways to specify a user:

user name
[*user id*]

Full CXbatch *manager* privileges are required to use this command.

DIsable Queue *queue*
Prevent any more requests from being placed in this *queue*.

CXbatch *operator* privileges are required to use this command.

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

CXbatch *operator* privileges are required to use this command.

EXit
Exit from the CXbatch manager subsystem.

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.

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**.

CXbatch *operator* privileges are required to hold any request that is not yours. 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) specifed by *requestid(s)* to be *value*.
The legal values for *field* 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.

CXbatch *operator* privileges are required to modify a request that is not yours.

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.

You may only move your own requests with this command, unless you have *operator* privileges.

MOVe Queue *queue1 queue2*

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

CXbatch *operator* privileges are required to use this command.

MOVe Request *requestid ... queue*

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

CXbatch *operator* privileges are required to use this command.

Purge Queue *queue*

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

CXbatch *operator* privileges are required to use this command.

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.

CXbatch *operator* privileges are required to release any request that is not yours. If a request is held by an operator, only an operator can release it.

SEt ACCOunting = {*OFf,ON*} <*queue-name*>

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

Full CXbatch *manager* privileges are required to use this command.

SEt ACC_logfile *<logfile-name>*
Change the file being used for CXbatch batch accounting.

Full CXbatch *manager* privileges are required to use this command.

SEt ACTivity_id_offset = *<offset-value> <queue-name>*
Set the activity ID offset for a CXbatch batch queue. The queue named as a parameter of the command must already exist.

Full CXbatch *manager* privileges are required to use this command.

SEt AId_mask = *<mask-value>*
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.

Full CXbatch *manager* privileges are required to use this command.

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. If the local host does not support per-process core file size limits, then this command reports an error. Otherwise, every batch queue on the local host has a per-process maximum core file size limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it is given a grandfather clause and allowed to retain its original limits. A request specifying a per-process core file size limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full CXbatch *manager* privileges are required to use this command.

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. If the local host does not support per-process data segment size limits, then this command reports an error. Otherwise, every batch queue on the local host has a per-process maximum data segment size limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it is given a grandfather clause and allowed to retain its original limits. A request specifying a per-process maximum data segment size limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full CXbatch *manager* privileges are required to use this command.

SEt DEBug *level*

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

0	No debug
1	Minimum debug
2	Maximum debug

Full CXbatch *manager* privileges are required to use this command.

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.

Full CXbatch *manager* privileges are required to use this 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.

Full CXbatch *manager* privileges are required to use this 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.

Full CXbatch *manager* privileges are required to use this command.

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.

Full CXbatch *manager* privileges are required to use this command.

SEt DESCription = *(<description>) <queue-name>*

Change the description of the named CXbatch queue.

Full CXbatch *manager* privileges are required to use this command.

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

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

Full CXbatch *manager* privileges are required to use this command.

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

Changes the *Import* attribute for a *queue*. The *queue* must already exist. If the *Import_dir* attribute is set to *Yes*, then 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*, then only jobs that specifically request to be imported are imported. If it is set to *No*, then jobs are ran 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.

Full CXbatch *manager* privileges are required to use this command. can.\f3SEt LOg_file *filename*

SEt MAIl *userid*
Specify the *userid* used to send CXbatch mail.

Full CXbatch *manager* privileges are required to use this command.

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

If the account name specification is followed by *:m*, then the account is designated as a CXbatch *manager* account, capable of using all *qmgr* commands. If the account name specification is followed by *:o*, then 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. Also see **ADd Manager** above.

Full CXbatch *manager* privileges are required to use this command.

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.

Full CXbatch *manager* privileges are required to use this command.

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. If a request already in the queue has asked for treatment more favorable than the new *nice-value*, then it is given a grandfather clause and allowed to retain its original limits. 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.

Full CXbatch *manager* privileges are required to use this command.

SEt NO_Access *queue*
Specify that no one can place requests in *queue*. Superuser is an exception; requests submitted by the superuser are always allowed into a queue, even if the superuser is not explicitly given access.

Full CXbatch *manager* privileges are required to use this command.

SEt NO_Default Batch_request Queue
Indicate that there is to be no default batch request queue.

Full CXbatch *manager* privileges are required to use this command.

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. If the local host does not support per-process CPU time limits, then this command reports an error. Otherwise, every batch queue on the local host has a per-process maximum CPU time limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it is given a grandfather clause and allowed to retain its original limits. A request specifying a per-process maximum CPU time limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full CXbatch *manager* privileges are required to use this command.

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. If the local host does not support per-process permanent file size limits, then this command reports an error. Otherwise, every batch queue on the local host has a per-process maximum permanent file size limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it is given a grandfather clause and allowed to retain its original limits. A request specifying a per-process maximum permanent file size limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full CXbatch *manager* privileges are required to use this command.

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.

Full CXbatch *manager* privileges are required to use this command.

SEt PRiority = *priority queue*
Specify the inter-queue *priority* of a *queue*.

Full CXbatch *manager* privileges are required to use this command.

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.

CXbatch *operator* privileges are required to use this command.

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.)

Full CXbatch *manager* privileges are required to use this command.

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.)

Full CXbatch *manager* privileges are required to use this command.

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.)

Full CXbatch *manager* privileges are required to use this command.

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. If the local host does not support per-process stack segment size limits, then this command reports an error. Otherwise, every batch queue on the local host has a per-process maximum stack segment size limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it is given a grandfather clause and allowed to retain its original limits. A request specifying a per-process maximum stack segment size limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full CXbatch *manager* privileges are required to use this command.

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

Full CXbatch *manager* privileges are required to use this command.

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. If the local host does not support per-process working set size limits, then this command reports an error. Otherwise, every batch queue on the local host has a per-process maximum working set size limit associated with it at all times. If a request already in the queue has asked for more than the new limit, then it is be given a grandfather clause and allowed to retain its original limits. A request specifying a per-process maximum working set size limit may only enter a batch queue if the queue's limit is greater than or equal to the request's limit. For the syntax of *limit*, see the **LIMITS** section below.

Full CXbatch *manager* privileges are required to use this command.

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. If a limit type is meaningful on a machine, then the corresponding *qmgr(8)* commands allow the association of a limit of that type with any batch queue on that machine. Note that users may request resource limits that are *not* meaningful on the machine where *qsub*(1) is invoked. If the 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 Queue *queue-name user-name*]]
Display in long format the status of all CXbatch queues on this host. If a *queue-name* is specified, output is limited to that queue. If a *user-name* 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 Queue *queue-name user-name*]]
Display the status of all CXbatch queues on this host. If a *queue-name* is specified, output is limited to that queue. If a *user-name* is specified, output downplays any requests not belonging to that user.

SHUtdown *seconds*]
Shutdown CXbatch on the local host. 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, then 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.

To restart CXbatch, the superuser must run the command **/usr/lib/nqs/nqsdaemon > /dev/console**.

CXbatch *operator* privileges are required to use this command.

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.

CXbatch *operator* privileges are required to use this command.

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*.

CXbatch *operator* privileges are required to use this command.

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 *batch* queues. In general, *pipe* queues in combination with *network* queues act as the mechanism that CXbatch uses to 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, which 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 client (pipe queue server), when handling a request, to discover that no destination queue will accept the request, for various reasons which 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. Requests submitted by the superuser are an exception; they are always queued, even if the superuser has not explicitly been given access.

SHELL STRATEGIES

The execution of a batch request requires the creation of a shell process to interpret the shell script which 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, then 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* simply 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* simply 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", 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, then 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 *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, then 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, then 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, then 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), 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_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.