CONVEX

CXbatch User's Guide

Fourth Edition

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

CONVEX
CXbatch User's Guide

Order No. DSW-183

Fourth Edition
February 1994

CONVEX Press
Richardson, Texas
United States of America

CONVEX
CXbatch User's Guide

Order No. DSW-183

Printed in the United States of America

Revision information for
CONVEX
CXbatch User's Guide

Edition	Document No.	Description
Fourth	710-002730-207	Released with CONVEX CXbatch V3.0, February, 1994. Added information for JOBS.
Third	710-002730-206	Released with CONVEX CXbatch, V2.1, January 1992. Added conceptual information and explanation of fields shown in output.
Second	710-002730-205	Released with CONVEX CXbatch, V2.0, September 1990, contains information about integration with CONVEX Share Scheduler and Checkpoint Restart, new qsub options, and enhanced accounting features.
First, Rev.1	710-002730-203	Released with CONVEX CXbatch V1.1, April 1990.
First	710-002730-200	Released with CONVEX CXbatch V1.0, February 1989.

Contents

Figures

Tables

Using this guide

Purpose and audience

The *CONVEX CXbatch User's Guide* describes basic CXbatch concepts and how to use the features available to the general user.

How to use this guide

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

For information on viewing queue and queue request information, read Chapter 2.

For information on how to submit a job to the batch network, read Chapter 3.

For information on manipulating your own jobs submitted to a batch queue, read Chapter 4.

For information on how to checkpoint and restart batch requests, read Chapter 5.

For information on messages put out by CXbatch, read Appendix A and Appendix B.

Notational conventions

This section discusses notational conventions used in this book.

General conventions

The following conventions are used in this guide:

- *Italics*
  - Designate user-supplied variables in a command-line example
  - Indicate document titles
- `Constant-width font` designates input that must be typed exactly as it appears and output displayed on the terminal screen. This includes:
  - Command names and options
  - Directives, program statements, display examples, printout examples, and error messages returned.
- Horizontal ellipsis (...) shows repetition of the preceding item(s).
- Words and abbreviations that indicate keyboard keys you press are identified in a distinctive bold type. For example, **RETURN** refers to the carriage return key. Words separated by a hyphen indicate two keys that you must press simultaneously. For example, **CTRL-x** indicates that you must press and hold down the **CTRL** key and then press the **x** key.
- The word "enter" in a phrase such as "enter `ls`" means that you type the command and then press **RETURN**.
- References to the *ConvexOS Programmer's Reference* appear in the form adb(1), where the name of the man page is followed by its section number enclosed in parentheses.

Command syntax

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

`COmmand` *input_file* `[`*input_file* `...] {a|b} [`*output_file*`]`

① ② ③ ④ ⑤

1. `Constant-width font` indicates that you must type the characters exactly as they appear (uppercase and lowercase are identical). The letters that appear in uppercase indicate

the minimum amount you must type to make the command unique. However, they do not have to be typed in uppercase.

2. *Italics* indicate a variable that must be supplied by the user. In this case, the user must supply the name of an *input_file*.
3. Square brackets [ ] indicate optional data. Horizontal ellipsis (...) shows repetition of the preceding item(s). In this case, the user can optionally specify more than one *input_file* on the command line.
4. Curly brackets { } indicate a choice. The choices available are shown inside the curly brackets and separated by the pipe (|) sign. In this case, the user can enter either a or b.
5. [*output_file*] indicates the user can optionally specify an output file name with the command.

Associated documents

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

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

- *CONVEX COVUEbatch Guide* (DSW-151). This book describes how to use and maintain the COVUEbatch batch processing software.
- *ConvexOS Extensions User's Guide* (DSW-053). This book describes ConvexOS utilities from the user's perspective.

Ordering documentation

To order the current edition of this or any other CONVEX document, send requests to:

CONVEX Computer Corporation
Customer Service
P.O. Box 833851
Richardson TX 75083-3851 USA

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

Technical assistance

If you have questions that are not answered in this book, contact the CONVEX Technical Assistance Center (TAC).

- Within the continental U.S., call 1(800)952-0379.
- From Canada, call 1(800)345-2384
- Outside continental U.S., contact local CONVEX office.

CXbatch processing 1

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

Users can submit batch jobs for execution on either the local machine or a remote machine configured with CXbatch or another version of Network Queueing System (NQS). There is one CXbatch daemon per machine, and each machine has its own set of queues.

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

There are two types of CXbatch queues:

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

Managing queues and queue requests

Commands available through the CXbatch qmgr utility allow a user to control submitted requests, track requests through the system, and with the right privileges, control CXbatch queues. The qmgr commands available to each user depend on their user type. CXbatch distinguishes three user types:

- General users
- CXbatch operators
- CXbatch managers

The following sections list and describe the commands available with each user type. The rest of this document describes the syntax and usage of each command available to the general user.

General users

General CXbatch users can track and control their own batch requests using a small subset of qmgr commands:

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

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

CXbatch operators

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

Managing queue requests

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

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

Managing queues

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

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

Managing CXbatch

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

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

CXbatch managers

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

Incompatibilities between NQS and CXbatch

CXbatch is based on NASA's Network Queueing System (NQS) but has many enhancements, including checkpoint restart, load balancing, fair share scheduling for batch jobs, and batch accounting. There are six major differences between the way you can use CONVEX CXbatch and other NQS systems:

- `move my_request` does not exist in other NQS systems and can only be used within CXbatch.
- CXbatch can mount remote files using NFS; other systems cannot.
- CXbatch's `maximum_request_priority` command, if used when submitting a request between CXbatch and another NQS system, causes the priority of previously-submitted requests to be moved to a lower priority. It does not delete previously-submitted requests.
- You cannot directly submit batch requests between CONVEX machines and other machines. You must use CXbatch to submit them.
- Several CXbatch `qsub` options are not applicable if the destination machine is not a CONVEX machine. These options are listed in Chapter 3, "Submitting a request," of the *CXbatch User's Guide*.
- Several CXbatch packet numbers are not recognized by other NQS systems. Table 1 lists packet numbers recognized only by CXbatch.

Table 1 Packet numbers recognized by CXbatch

Packet Number	Type	Use
206	nqs	Queue request from remote `qsub`
207	nqs	Get sequence number
208	nqs	Hold request
209	nqs	Release request
210	nqs	Add queue alias
211	nqs	Delete queue alias
212	nqs	Ping with ack (used for debugging)
213	nqs	Ping with ack (used for debugging)
214	nqs	Set maximum request priority
215	nqs	Set checkpoint directory
216	nqs	Checkpoint a request
218	nqs	Force request to run
219	nqs	Suspend request
220	nqs	Resume request
221	nqs	Set share policy fixed
222	nqs	Set share policy user
223	nqs	Force run request
224	nqs	Set global per-user-run-limit
225	nqs	Set queue per-user-run-limit
226	nqs	Restart request
227	nqs	Checkpoint done
228	nqs	Copy open files on checkpoint
229	nqs	Keep checkpoint files upon apr event
230	nqs	set next sequence number
205	net	Get remote queue and request information

Integration with CONVEX Share Scheduler

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

COVUEbatch considerations

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

There are several items that you must consider if COVUEbatch is installed on your system.

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

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

Integration with Checkpoint Restart

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

Users and operators can checkpoint and restart processes that are running under CXbatch by using Checkpoint Restart features built into CXbatch. See Chapter 5, "Checkpoint Restart features," for details on these features.

Displaying information 2

This chapter describes how to display information about queues and requests in queues using the following CXbatch commands:

- `qstat`—Displays status of CXbatch queues and requests in queues.
- `qwatch`—Displays status of CXbatch queues and requests in queues interactively.
- `qlimit`—Displays queue limits.
- `qps`—Displays information about CXbatch-related processes.
- `qjlist`—Displays the contents of a shell script.

How to use each of these commands is described in the following sections.

Displaying status of queues

You can display information about CXbatch queues and requests in the queues using the `qstat` command. The format for this command is

`qstat` [*option...*] [*queuename*[@*hostname*]...]

where

option controls the type and amount of information displayed. If no options are specified, `qstat` shows only those requests belonging to the user issuing the command. *option* can be one or more of the following:

`-a`	Displays status for all requests in the queue.
`-l`	Displays additional information about queues and queue requests. See the "Displaying additional request information"section in this chapter for a description of the output for this option.
`-m`	Displays the date and time requests will run. See the "Displaying date and time requests will run"section in this chapter for a description of the output for this option.
`-u` *username*	Displays only those requests belonging to the specified *username*.
`-x`	Displays additional information about queues. See the "Displaying additional queue information"section in this chapter for a description of the output for this option.

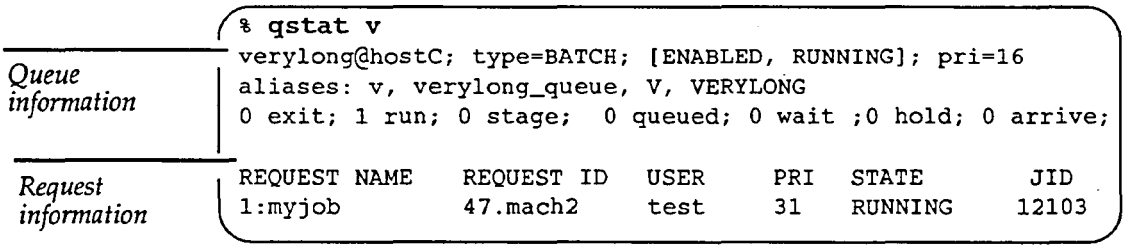
queuename is the name of the queue for which you wish status information. If you do not specify a queue, information for all queues on the requested host is displayed.

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

Displaying standard output

There are two sections to the standard output for the `qstat` command: information on the queues and information on each individual request in the queue. Figure 1 illustrates the standard output for the `qstat` command.

Figure 1 Standard `qstat` output



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

```
verylong@hostC; type=BATCH; [ENABLED, RUNNING]; pri=16
1                2      3       4         5
```

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

   `BATCH` Executes CXbatch requests.

   `PIPE` Routes CXbatch requests to queues that can execute them.
3. Indicates whether or not the queue can accept requests. This can be

   `ENABLED` CXbatch is running on the local machine and the queue is accepting requests.

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

   `CLOSED` CXbatch is not running on the local machine.
4. Indicates whether or not the queue can execute requests. This can be

   `INACTIVE` Requests in the queue are permitted to run; none are running.

   `RUNNING` Requests in the queue are permitted to run; some are running.

   `STOPPING` New requests sent to the queue are not permitted to run, but requests that are currently running are allowed to complete.

   `STOPPED` Requests in the queue are not permitted to run; none are running.

   `SHUTDOWN` CXbatch is not running on the local machine.

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

The second line of the output tallies the number of requests in specific states:

```
0 exit; 0 run; 0 stage; 0 queued; 0 wait; 0 hold; 0 arrive;
  1       2       3         4         5       6         7
```

1. Number of requests in this queue in a state of exiting.
2. Number of requests in this queue in the state of running.
3. Number of requests in this queue in the staged state, which means the request has completed executing and is moving the stdout and stderr files to the appropriate destination directory.
4. Number of requests in this queue in a state of being queued.
5. Number of requests in this queue in a state of waiting.
6. Number of requests in this queue in a state of holding.
7. Number of requests in this queue in a state of arriving.

The rest of the output displays status information about each of the requests in the queue:

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

1. Name assigned to the request.
2. Unique identifier assigned to request when it is submitted to the queue.
3. User submitting the request.
4. Intra-queue priority assigned to request. This priority affects which job in a queue is executed next. This number can be from 0 to 63; 0 is the lowest priority and 63 the highest.
5. State of the request. This can be:

   `ARRIVING` Request is arriving at the queue.

   `CHECKPOINTED` A failed attempt was made to restart the checkpointed request. You can attempt to manually restart the request using the `qrestart` command. Refer to Chapter

5, "Checkpoint Restart features," for details on using this command.

DEPARTING	Request is departing from the queue but has not yet been received by the destination queue.
EXITING	Batch request has completed executing and will exit from the system after the required output files are returned to their intended destinations.
HOLDING	A hold has been placed on the request preventing it from entering any other state.
QUEUED	Request is queued and eligible for running or routing. This is the most common state.
ROUTING	Request has reached the head of a pipe queue and is being routed to another queue.
RUNNING	Request has reached its final destination batch queue and is executing.
WAITING	Request is waiting for a specified amount of time to pass before attempting to execute. This could be because it was submitted with a future date and time specified for running, or because a pipe queue could not route the request and will attempt to route it later.
SUSPENDED	Request is suspended from running. It will remain suspended until manually resumed.

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

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

Displaying additional request information

You can use the `-l` option to display additional information on queue requests. Figure 2 illustrates the output for this option.

Figure 2 Additional queue request information

```
% qstat -l long
Queue for long jobs.
 long@mach2; type=BATCH; [ENABLED, RUNNING]; pri=32
aliases: l, long_queue, L, LONG
 0 exit; 1 run; 0 stage; 0 queued; 0 wait; 0 hold; 0 arrive;

Request 1: Name=STDIN Id=25.mach2
 Owner=test Priority=31 RUNNING Pgrp=4918
```

Additional request information

```
1  Created at Mon Jan 23 16:30:03 CST 1989
2  Mail = [NONE]
3  Mail address = test@mach2
4  Owner user name at originating machine = test
5  Import directory: Yes
6  Per-proc. core file size limit= UNLIMITED <DEFAULT>
7  Per-proc. data size limit= UNLIMITED <DEFAULT>
8  Per-proc. permanent file size limit= UNLIMITED <DEFAULT>
9  Per-proc. memory size limit= UNLIMITED <DEFAULT>
10 Per-req. memory size limit= UNLIMITED <DEFAULT>
11 Per-proc. execution nice priority = 0 <DEFAULT>
12 Per-proc. stack size limit= UNLIMITED <DEFAULT>
13 Per-proc. CPU time limit= [600.0, 600.0]<DEFAULT>
14 Per-req. CPU time limit= UNLIMITED <DEFAULT>
15 Per-proc. working set limit= UNLIMITED <DEFAULT>
16 Standard-error access mode = SPOOL
17 Standard-error name = mach2:/mnt/test/STDIN.e272
18 Standard-output access mode = SPOOL
19 Standard-output name = mach2:/mnt/test/STDIN.o272
20 Shell = DEFAULT
21 Umask = 2
22 Restartable = Yes
23 Checkpointable = Yes
```

1. `Created at` specifies the day, date, and time the request was created.
2. `Mail` specifies whether or not mail is sent when the job starts and when it finishes.
3. `Mail address` specifies where mail generated for the request is sent.
4. `Owner user name at originating machine` defines the user submitting the request.

5. `Import directory` describes whether or not the current working directory is imported for this request.

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

7. `Per-process data size limit` is the maximum size for this request that a data segment for a process can be.

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

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

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

11. `Per-process execution nice value` is the maximum nice value for this request that a process can have. The nice value determines the proportion of CPU time allocated to a process relative to all other processes in the system. The lower the nice value assigned to a process, the higher the proportion of CPU time allocated to that process. A practical range is from -20 to 20.

12. `Per-process stack size limit` is the maximum size for this request that a stack segment for a process can be.

13. `Per-process CPU time limit` is the maximum total CPU time for this request that a process can use. If this limit is exceeded, ConvexOs sends the appropriate signal to the process. If this signal is not caught or ignored, the process exits.

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

15. `Per-process working set limit` is the maximum amount of physical memory for this request that a process

can use. If this limit is reached, the job is targeted for paging.

16. `Standard-error access mode` indicates that the stderr file is written to spooling directories during job execution and copied to a destination at job termination.
17. `Standard-error name` is the name of the file where error output is sent.
18. `Standard-output access mode` indicates that the stdout file is written to spooling directories during job execution and copied to a destination at job termination.
19. `Standard-output name` is the name of the file where standard output is sent.
20. `Shell` is the shell specified for this request to interpret shell scripts.
21. `Umask` defines the default umask for this request.
22. `Restartable` specifies whether or not this request can be restarted.
23. `Checkpointable` specifies whether or not this request is checkpointable.

Displaying date and time requests will run

You can use the `-m` option to display information about the date and time a request is scheduled to run. Only those requests submitted with a specific date and time have this information available. Figure 3 illustrates the output for this option when a request in the queue is submitted to run at a future date and time.

Figure 3 Future run time output

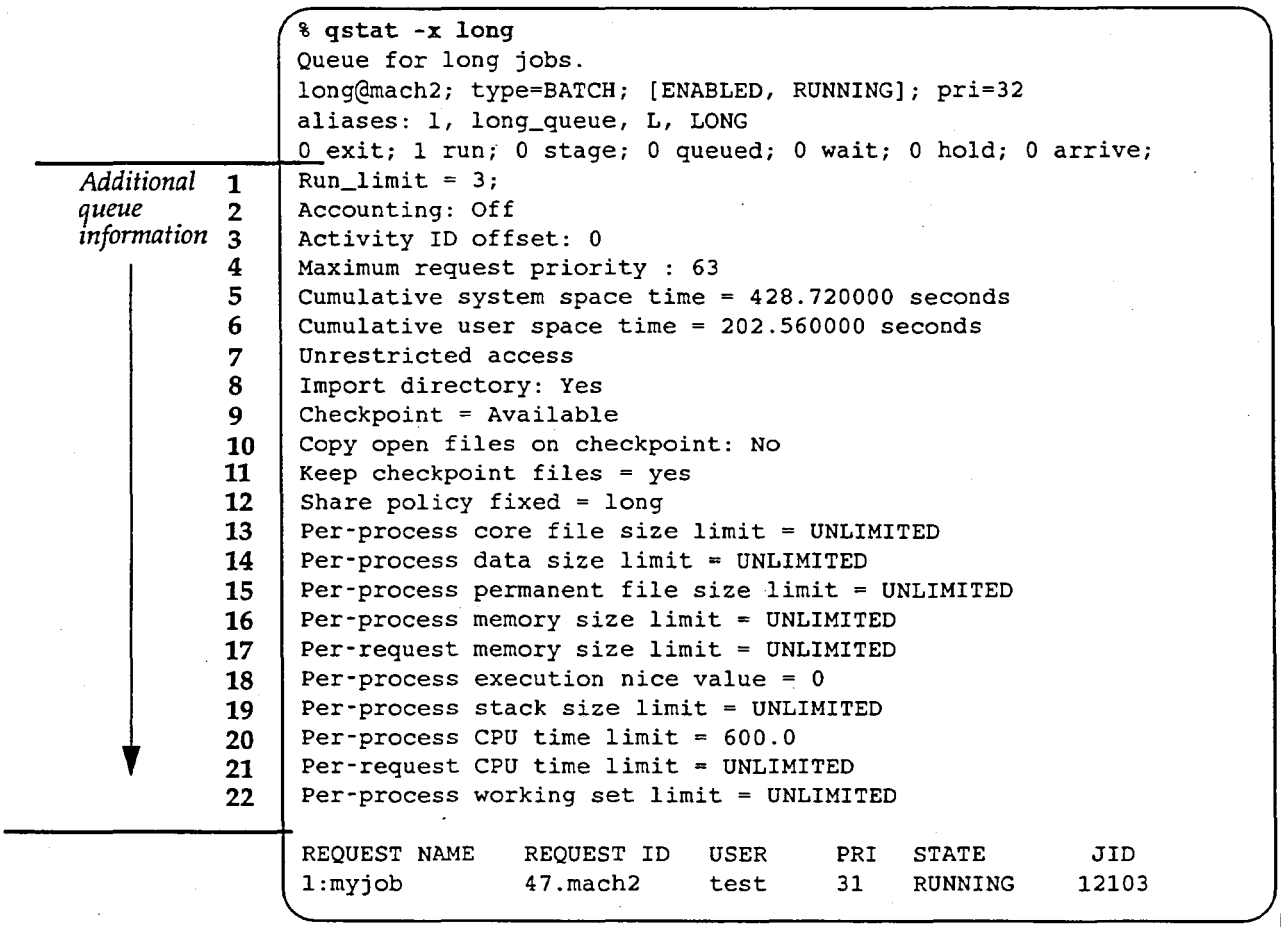
```
% qstat -m long
Queue for long jobs.
long@mach2; type=BATCH; [ENABLED, INACTIVE]; pri=32
aliases: l, long_queue, L, LONG
0 exit;0 run; 0 stage; 0 queued;1 wait; 0 hold; 0arrive;

Request 1: Name=myjob Id=297.mach2
Owner=test Priority=31 WAITING Wed Jan 25 00:00:00 CST 1989
```

The request is shown as `WAITING` and includes the day, date, and time it is scheduled to run.

Displaying additional queue information

You can use the `-x` option to display additional information about the queue without receiving additional information about queue requests. Figure 4 illustrates the output from this option.

Figure 4 Additional queue information



1. `Run_limit` limits the number of requests that can run in a queue at any one time. When the limit is exceeded, requests are queued until the number of jobs running is less than the limit.
2. `Accounting` indicates whether or not batch accounting is activated.
3. `Activity ID offset` is the number added by CXbatch to a job's activity identification (ID) before the request is executed. The activity ID offset is typically an integer from

1 to 9, with 0 reserved for jobs not submitted to a batch queue. The activity ID is used for accounting purposes.

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

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

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

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

`Unrestricted`	Queue can receive any request from any submitter.
`Restricted`	Queue can receive only those requests submitted by a specified group(s) or user(s).
`Pipeonly`	Queue accepts requests only from a pipe queue.

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

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

9. `Checkpointable` specifies whether or not jobs are automatically checkpointed in this queue in the event CXbatch shuts down. This can be:

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

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

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

10. `Copy open files` defines whether or not CXbatch preserves the state of open files within a process when the request is checkpointed.

11. `Keep checkpoint files` defines whether or not CXbatch preserves a request's checkpoint upon an apr(automatic processor recovery) event. This allows the user to restart their job from the last checkpoint time.

12. `Share policy` describes the queue share policy. This can be

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

    `Fixed` Charges CPU usage to a specified account.

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

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

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

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

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

18. `Per-process execution nice value` is the maximum nice value a process can have. The nice value determines the proportion of CPU time allocated to a process relative to all other processes in the system. The lower the nice value assigned to a process, the higher the proportion of CPU time allocated to that process. A practical range is from -20 to 20.

19. `Per-process stack size limit` is the maximum size of a stack segment for a process.

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

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

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

Displaying queue status interactively

You can view the status of queues and requests in queues interactively using the `qwatch` command. This allows you to monitor one or more queues without making repetitive calls to `qstat`.

`qwatch` is a tool designed to be used by system managers in fine-tuning CXbatch queue loads. While it does not interfere with CXbatch daemons, it does use a small amount of CPU resources and should not be used to check on the status of one request. Use `qstat` for checking the status of a request.

The format for the `qwatch` command is

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

where

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

`-c` *count* specifies how many screen updates must occur before updating terminates. If this number is 0, updating continues until manually stopped by pressing **CTRL-c**. The default is 0.

You can display status information about queues either with or without status information about queue requests. When you first invoke `qwatch`, you only receive status information about queues. The first page displays status information on the first five queues on the system. If there are more than five queues, you can display additional pages of queue information by typing the number of the additional page. If there are more than 25 queues, `qwatch` ignores those above 25. Figure 5 on the next page illustrates the initial page output for the `qwatch` command.

Figure 5 Interactive queue status output

```
% qwatch

1  mach1 CXbatch Activity            Wed Jun 6 15:59:54 1991
2                              3 queues
          3              4       5       6        7
   a short@mach1;type=BATCH; [ENABLED,RUNNING];pri=48
8         aliases: s
          0 exit;1 run;0 stage;0 queued;0 wait;0 hold;0 arrive;
           9      10     11      12       13      14        15
   b long@mach1; type=BATCH; [ENABLED,RUNNING];pri=32
          aliases: 1
          0 exit;1 run; 0 stage; 0 queued;0 wait;0 hold;0 arrive;

   c verylong@mach1; type=BATCH; [ENABLED,INACTIVE];pri=16
          aliases: 1
          0 exit;0 run;0 stage;0 queued;0 wait;0 hold;0 arrive;

   Page 1 of 1 Type 'a' - 'c' for queue.
```

1. Name of the host containing the queue and the day, date, and time the status was taken.
2. Number of queues on the host.
3. Name of the queue and what host it is configured on.
4. Type of queue. This can be

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

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

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

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

`INACTIVE`	Requests in the queue are permitted to run; none are running.
`RUNNING`	Requests in the queue are permitted to run; some are running.

STOPPING New requests sent to the queue are not permitted to run, but requests that are currently running are allowed to complete.

STOPPED Requests in the queue are not permitted to run; none are running.

SHUTDOWN CXbatch is not running on the local machine.

7. Specifies the inter-queue priority. This priority affects which queue is looked at first for the next job to run. Priority can be any number from 0 to 63; 0 is the lowest priority and 63 the highest.
8. Alternate names by which the queue can be referenced.
9. Number of requests in this queue in a state of exiting.
10. Number of requests in this queue in the state of running.
11. Number of requests in this queue in the staged state, which means the request has completed executing and is moving the stdout and stderr files to the appropriate destination directory.
12. Number of requests in this queue in a state of being queued.
13. Number of requests in this queue in a state of waiting.
14. Number of requests in this queue in a state of holding.
15. Number of requests in this queue in a state of arriving.

If you wish to display status information about the requests in queues, enter the letter assigned to the queue. The first page displays status information on the first 16 requests in the queue. If there are more than 16 requests, you can display additional pages by typing the number of the additional page. `qwatch` can display up to nine pages or 144 requests. Figure 6 on the next page illustrates the output for selected queues.

Figure 6 Interactive queue request output

```
mach1 CXbatch Activity Thu Aug 23 16:00:44 1990
 long@mach1; type=BATCH; [ENABLED,RUNNING];pri=32
 aliases: 1
 0 exit;1 run;0 stage;2 queued;0 wait;0 hold;0 arrive;
 REQUEST NAME    REQUEST ID    USER   PRI    STATE      JID
 1:STDIN         346.mach1     johndoe 31   RUNNING   16257
 1:STDIN         346.mach1     johndoe 31    QUEUED
 1:STDIN         346.mach1     johndoe 31    QUEUED

Page 1 of 1 Type '-' for queues.
```

Return to the initial screen by typing a dash (-). Exit `qwatch` by pressing **CTRL-c**.

The information provided about each queue request is:

REQUEST NAME	REQUEST ID	USER	PRI	STATE	JID
1:STDIN	346.mach1	johndoe	31	RUNNING	16257
1	**2**	**3**	**4**	**5**	**6**

1. Name assigned to the request.
2. Unique identifier assigned to request when it is submitted to the queue.
3. User submitting the request.
4. Intra-queue priority assigned to request. This priority affects which job in a queue is executed next. This number can be from 0 to 63; 0 is the lowest priority and 63 the highest.
5. State of the request. This can be

   `ARRIVING` Request is arriving at the queue.

   `CHECKPOINTED` A failed attempt was made to restart the checkpointed request. You can attempt to manually restart the request using the qrestart command. Refer to Chapter 5, "Checkpoint Restart features," for details on using this command.

   `DEPARTING` Request is departing from the queue but has not yet been received by the destination queue.

   `EXITING` Batch request has completed executing and will exit from the system after the

required output files are returned to their intended destinations.

HOLDING	A hold has been placed on the request preventing it from entering any other state.
QUEUED	Request is queued and eligible for running or routing. This is the most common state.
ROUTING	Request has reached the head of a pipe queue and is being routed to another queue.
RUNNING	Request has reached its final destination batch queue and is executing.
WAITING	Request is waiting for a specified amount of time to pass before attempting to execute. This could be because it was submitted with a future date and time specified for running, or because a pipe queue could not route the request and will attempt to route it later.
SUSPENDED	Request is suspended from running. It will remained suspended until manually resumed.

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

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

Displaying queue limits

You can display the resource limits of a queue using the `qlimit` command. Queue limits are set by the system manager when the queue is created.

CXbatch supports many types of resource limits, but not all operating systems support all of them. The `qlimit` command displays those limits that CXbatch can directly support under the operating system on the indicated host machine. The format is

`qlimit` [*hostname* ...]

where *hostname* is the desired host machine. If *hostname* is omitted, the local host is assumed.

Figure 7 illustrates use of the `qlimit` command to display the local host limits and shell strategy.

Figure 7 Queue limit output

```
% qlimit
1  Core file size limit (-lc)
2  Data segment size limit (-ld)
3  Per-process permanent file size limit (-lf)
4  Per-process memory size limit (-lm)
5  Per-request memory size limit (-lM)
6  Nice value (-ln)
7  Stack segment size limit (-ls)
8  Per-process cpu time limit (-lt)
9  Per-request cpu time limit (-lT)
10 Working set limit (-lw)

11 Shell strategy = FREE
```

The information displayed is

1. `Per-process core file size limit` is the maximum size a core file for a process can be. If a process attempts to create a core file exceeding this maximum size, the core file is not written.
2. `Per-process data size limit` is the maximum size the data segment for a process can be.
3. `Per-process permanent file size limit` is the maximum size a file created by a process can be. If a process attempts to write to a file larger than this maximum, ConvexOs sends a signal to the process. If nothing is defined in the process to handle that signal, the process is killed.

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

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

6. `Per-process execution nice value` is the maximum nice value a process can have. The nice value determines the proportion of CPU time allocated to a process relative to all other processes in the system. The lower the nice value assigned to a process, the higher the proportion of CPU time allocated to that process. A practical range is from -20 to 20.

7. `Per-process stack size limit` is the maximum size of a stack segment for a process.

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

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

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

11. `Shell strategy` specifies the ConvexOS shell that is used to interpret the script commands submitted through CXbatch if one is not specified when the request is submitted using `qsub -s`. The shell strategy can be

`FIXED`	The system manager specifies the shell that interprets script file commands.
`LOGIN`	CXbatch uses the shell specified in the /etc/passwd file for the user submitting the job. CXbatch does not read the script file for a shell specification.
`FREE`	CXbatch uses the shell specified in the /etc/passwd file for the user submitting the job unless a shell is specified in the script file. When a user submits a batch request, CXbatch supplies the name of the script file

to the login shell as standard input. The login shell reads the first line of the script file. If the first line specifies a shell, that shell is used to interpret the script file commands instead. If there is no shell specified in either of these places, CXbatch uses `sh`.

For further information, refer to the qlimit(1) man page.

Displaying status of related processes

You can display the status of CXbatch-related processes from batch queues on the local system using the `qps` command. Pipe queues and remote queues are not significant because they do not have processes that affect the local machine.

The format for the `qps` command is

`qps` [*option*][*process-id*]

where

option controls which CXbatch-related processes are reported. If no options are given, `qps` displays information about all CXbatch-related processes, including the daemon processes. *option* can be one of the following:

- `-r` displays information on a specific request. Only the processes related to the specified request are displayed.
- `-p` determines if a specific process is running under CXbatch. If it is, the following message displays showing the queue and request to which the process is related:

  ```
  Process ID 508 is being executed by
  Request ID 3782 in the short queue.
  ```

  If it is not running, the following message displays:

  ```
  Process ID 508 is not being executed
  by CXbatch.
  ```

  With `qps`, the top-level daemons are not considered to be running under the CXbatch system, but CXbatch shepherd processes are.
- `-q` is a silent version of the `-p` option. Nothing is printed to the screen, but the exit status of `qps` is set appropriately.

process-id is the identifying number of the process for which you wish to see the status.

Figure 8 illustrates the output for the qps command.

Figure 8 Status of CXbatch-related processes

```
% qps
  1          2          3          4     5      6
QUEUE      REQ        PID        STAT TIME   COMMAND
<daemon>              21052 S           0:00   CXbatch logdaemon
<daemon>              21053 S           0:00   CXbatch nqsdaemon
<daemon>              21054 S           0:00   CXbatch netdaemon
<daemon>              210571 S          0:00   CXbatch queued
long       508.mach   112535 S          0:00   CXbatch shepherd
long       508.mach   112536 S     R    0:00   /bin/make -k ...
long       508.mach   112539 S     T    0:00   /bin/make -k ...
long       508.mach   112535 S     D    0:00   tsch
long       508.mach   112535 S     R    0:00   tsch
```

The information provided is:

1. QUEUE is the queue that contains the request related to the process.
2. REQ is the identification number assigned to request initiating process and machine where it originated.
3. PID is the unique identification number assigned to the process.
4. STAT is the status of the process. This can be

R	Process is runnable.
T	Process is stopped.
P	Process is in page wait.
D	Process is in disk wait or other short term wait.
S	Process is active (sleeping for less than 20 seconds).
I	Process is idle (sleeping for more than 20 seconds).
M	Process is waiting for a migration daemon.
Z	Process is trying to exit but the parent of the process is not waiting for it (zombie).

5. TIME is the amount of CPU time used so far.
6. COMMAND is the command that started the process.

Displaying contents of shell scripts

You can view the contents of a shell script used in a batch request using the `qjlist` command. To display a shell script, you must either:

- Own the request
- Have superuser privileges
- Be designated as a batch operator or manager

The format for the `qjlist` command is

`qjlist` *request_id*.[@*hostname*]

where

request_id is the number assigned to the request when it is submitted to CXbatch. By default, CXbatch displays the *request_id* of the request at the submitter's terminal when a batch request is successfully submitted. You can also get this number using the `qstat` command. Refer to the "Displaying status of queues" section in this chapter for details on using `qstat`.

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

Figure 9 illustrates the contents of the 52.mach2 shell script using the `qjlist` command.

Figure 9 Contents of a shell script

```
% qjlist 52.mach2
sleep 20
echo "It is time to go home"
CTRL-d
```

For further information, refer to the qjlist(1) man page.

Submitting a request

3

This chapter describes how to submit a request to a CXbatch queue using the `qsub` command. It describes

- Three methods for submitting batch requests
- Format of the `qsub` command
- Controlling how, when, and where a request is submitted and executed
- How to redirect output and error information
- How to specify per-process and per-request resource limits
- How to send notification that a request is starting or ending
- How to embed `qsub` command options in a shell script

Three methods for submitting batch requests

A batch request consists of one or more commands submitted by a user or a user program to CXbatch. You can submit a request for batch execution in one of three ways:

- With interactive commands
- In a script file
- In a compiled program

Each of these methods is described in the following sections. See the "qsub command options" section in this chapter for details on using `qsub` command options.

Using interactive commands

You can submit batch requests interactively by issuing the `qsub` command and following it with the commands you want to execute. You complete the transaction by pressing **CTRL-d**. Figure 10 illustrates an interactive session.

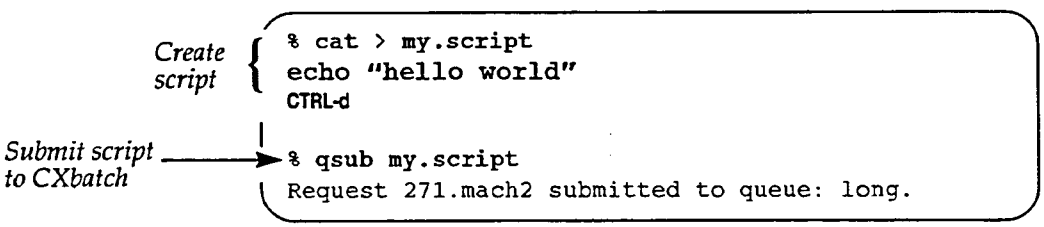
Figure 10 Submitting a batch job interactively

```
% qsub
sleep 20
echo "It is time to go home"
CTRL-d
```

Using a script file

You can submit a batch request using a script file. To do this, create a file that contains all the commands you want to execute. Then, issue the `qsub` command and include the name of the script file on the command line. Figure 11 illustrates how to create a script file named *my.script* and submit it to CXbatch. A script file is run by a shell, so the script file must contain commands the shell can recognize.

Figure 11 Submitting batch job using a script file



When you submit a script file, you must include the name of the file on the command line. Make sure that script files submitted to CXbatch are completely self-contained so that when the script

exits, no background child processes remain. If a request sends mail to a local user, the mail program runs a background process to deliver the mail and does not wait for it to complete.

If you send mail at the end of a request's script, have the request sleep for a few seconds just before exiting to allow the delivery process to complete. If the request exits before the mail delivery process completes, the shepherd process aborts the delivery and the mail disappears. This does not occur if you send mail to a remote machine or if you pass the `-v` (verbose) option to the mail program.

Using a compiled program

You can submit a batch request using a compiled program (binary code). To do this, include the name of the program in a script file or enter the name as standard input following the `qsub` command line. Figure 12 illustrates how to submit the compiled program *test* by entering the program name as standard input.

Figure 12 Submitting a batch job using a compiled program

```
% qsub
test
CTRL-d
```

Figure 13 illustrates how to submit the compiled program *test* by creating a shell script file and submitting that script file using the `qsub` command.

Figure 13 Submitting a compiled program using a script file

```
% cat > myjob
test
CTRL-d
% qsub myjob
```

Figure 14 illustrates an incorrect way to submit the compiled program *test* as a batch request.

Figure 14 Incorrect way to submit compiled program

```
% qsub test
Line length exceeds 1025 characters.
Script file line 2.
```

This command is incorrect because `qsub` is expecting either the name of a shell script or input from the terminal.

qsub command options

Use the `qsub` command to submit a request to a batch queue. The format for the `qsub` command is

`qsub` *option* [*option* ...]

where *option* can be one or more values that allow you to

- Control how, when, and where the request is run
- Redirect output files and error messages
- Specify resource limits
- Request status updates

Unlike ConvexOS command format, which allows you to string several options together after a single dash, CXbatch command format requires that you list each option separately. For example, to execute the `qsub` command with two options, you must enter

`qsub` *-option1 -option2*

The following sections describe the options available with the `qsub` command, except for those related to checkpointing. Refer to Chapter 5, "Checkpoint Restart features" for information on options related to checkpointing.

Controlling run requests

The `qsub` command includes options to control how, when, and where a request is run. Table 2 lists each option and its meaning.

Table 2 `qsub` options that control run requests

Option	Definition
`-a` [*date*] [*time*]	Run request after a stated time
`-b` *bill_account*	Run request under specified billing account
`-h`	Place request on hold
`-i`	Import current working directory
`-l`	Run request under login shell
`-ni`	Do not import current directory
`-p` *priority*	Set intra-queue priority
`-q` *queue*	Specify queue to submit request
`-r` *name*	Assign name to request
`-s` *shell*	Specify shell to interpret script files
`-x`	Export value of all environment variables
`-z`	Submit request silently

The following sections explain these options in detail and give examples of their use.

Specifying a future run time

Unless you specify otherwise, a batch request is eligible for execution immediately after you issue the `qsub` command. You can use the `-a` option to specify a future run time. The format for this option is

```
qsub -a [date] [time]
```

where

date specifies the date the job should run. If no date is indicated, the current month, day, and year are assumed.

If you specify a date, the year is optional. If it is omitted, the current year is assumed. You can abbreviate the names of the month and day by using the first three (or more) characters of the word. For example, you can indicate a day of the week (like, Mon, Tue). A period following the abbreviation is optional. Uppercase and lowercase letters are equivalent, so that "WeD" is treated the same as "WED" or "wed."

You can use the words "today" or "tomorrow" rather than specifying a month, day, and year.

time specifies the time of day the job should run. A 24-hour clock is the default, so if 1:00 is specified without "am" or "pm," CXbatch interprets it as 01:00 or 1:00 a.m.

You can use a 12-hour clock as the basis for the time by including an "am" or "pm" designation with the time. 1:00 p.m. is shown as 1:00pm or 1pm. If 1:00 p.m. is specified, it is interpreted as 13:00 on the 24-hour clock.

"12am" refers to 0:00:00, "12n" refers to noon, and "12 pm" refers to 24:00:00. You can optionally use the words "midnight" or "noon."

You can include a time-zone designation, as in "13:01-PDT." CXbatch understands European time zones. If a time-zone designation is omitted, the local time zone is assumed, with daylight savings time included when appropriate.

The order of *date* and *time* is irrelevant. The following formats are equivalent:

```
qsub -a [date] [time]
qsub -a [time] [date]
```

If you use *date* and *time*, enclose the entire string in double quotes. You must also enclose either *date* or *time* in double quotes if you include spaces within either option.

```
"January 1, 1989, 12:31"
```

shows the correct way to indicate combined *date* and *time* as well as the correct way to use spaces. This rule applies when this option is included on the command line or in the shell script. Refer to the last section in this chapter, "Embedded Options," for information on embedding options in shell scripts.

Some valid examples of *date* and *time* are:

```
"01-Jan-1988 13:00"
tomorrow
"today 5pm"
```

For example, the following command executes the script file *test* on July 4, 2026, at 12:31 Eastern Daylight Time:

```
qsub -a "July 4, 2026 12:31-EDT" test
```

Specifying a billing account

If accounting is set up for the queue, you can use the `-b` option to specify that a request run under a specific billing account. For example, the following command bills the script file *my_job* to the *activity1* billing account:

```
qsub -b activity1 my_job
```

Putting a request on hold at submission

You can use the `-h` option to place a request on hold at the time you submit the job. For example, the following command places the script file *my_job* on hold when it is submitted:

```
qsub -h my_job
```

Controlling importation of the current directory

Often, batch requests require access to all files in all subdirectories of your current working directory. Because of this, you may need to make the files in your current directory available to batch requests. Granting this access is called *importing* the current working directory.

The system manager defines the ability to import directories on a per-queue basis. You can use the `qstat` command to view whether or not the queue you are using allows importation of current directories. See Chapter 2, "Displaying information," for details on using the `qstat` command.

If you need to import files and the queue you are using does not allow this, see your system manager. If you do not want importing or it has been denied by the system manager, CXbatch expects to find any files required to run the job in your home directory.

You can use the `-i` option to import the current working directory before executing the batch request. If the execution queue is on the same machine as the current working directory, CXbatch changes directories before it starts. If the execution queue is on another machine, CXbatch imports the directories and subdirectories with NFS by making temporary mount points in the /tmp directory on the originating machine. Be aware of any local automatic cleanup facilities of /tmp that might affect these NFS mount points.

For example, the following command imports the current working directory for the script file *my_job*:

```
qsub -i my_job
```

CXbatch cannot import a current working directory that is already within an NFS file system. For example, if you are on a machine called sleepy in a file system imported from happy, and you submit a job with the `-i` option to a queue on grumpy, the file system cannot be imported from happy to grumpy.

You can use the `-ni` option to prevent the importation of the current working directory.

For example, the following command does not import the current working directory for the script file *my_job*:

```
qsub -ni my_job
```

For further information on the import option, refer to the *CONVEX CXbatch System Manager's Guide* and the qsub (1) man page.

Specifying use of login shell

During configuration of the CXbatch system, the system manager defines a shell strategy for each queue. A shell strategy specifies the ConvexOS shell that is used to interpret the script commands submitted through CXbatch if one is not specified by the user when the request is submitted. The shell strategy can be

- **Fixed**—The system manager specifies the shell that interprets script file commands.
- **Login**—CXbatch uses the shell specified in the /etc/passwd file for the user submitting the job. CXbatch does not read the script file for a shell specification.
- **Free** —CXbatch uses the shell specified in the /etc/passwd file for the user submitting the job unless a shell is specified in the script file. When a user submits a batch request, CXbatch supplies the name of the script file to the login shell as standard input. The login shell reads the first line of the script file. If the first line specifies a shell, that shell is used to interpret the script file commands instead. If there is no shell specified in either of these places, CXbatch uses `sh`.

You can use the `qlimit` command to discover which strategy is in effect for your system. See Chapter 2, "Displaying information" for details on using the `qlimit` command.

Unless otherwise specified, CXbatch runs jobs using a noninteractive shell. You can use the `-l` option to specify using an interactive login shell.

If you specify use of a login shell, your default login shell (determined by the shell strategy specified for this queue) is used unless you specify otherwise. You can use the `-s` option to specify an alternate shell. The format for this option is

`qsub -s` *shell-name*

where *shell-name* indicates the absolute path name on the executing host of the shell to be used.

If you use a C shell, CXbatch gets environment variables from /etc/login and the .login file in your home directory. If you use the Bourne or Korn shell, CXbatch gets environment variables from /etc/profile and the .profile file in your home directory.

To prevent `stty`, `tset`, and `msgs` from running during batch jobs, you can add the string ENVIRONMENT=BATCH to your .login, .profile, or .cshrc file so shell scripts can test for batch request execution. Then place an `if` statement in the .login, .profile, or .cshrc file.

If your login shell is a C shell, the following `if` statement in your .login file prevents `stty`, `tset`, and `msgs` from running during batch jobs. C shell always reads your .cshrc file regardless of whether or not it is running as a login shell.

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

If your login shell is the Bourne shell or Korn shell, the following `if` statement in your .profile file has the same effect of preventing `stty`, `tset`, and `msgs` from running during batch jobs.

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

As an alternative, you might use

```
if ($?ENVIRONMENT) exit
```

Setting priority on batch jobs

You can use the `-p` option to assign an intraqueue priority to a job request. This priority does not determine execution priority of the request; it is used to determine only relative ordering of requests within a queue. The ConvexOS nice value determines execution priority of requests. Refer to the nice(1) man page for details.

When CXbatch adds a request to a queue, it compares the intraqueue priority of the request with the intraqueue priority of all existing requests. It then places the request in the queue ahead of those requests with a lower priority and behind existing requests with a higher priority. 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 you do not specify an intraqueue priority, CXbatch assigns the default intraqueue priority to each request.

The specified priority can be any number from 0 and 63; 0 is the lowest priority and 63 the highest. You cannot assign a priority to a request that is higher than the maximum request priority assigned to the queue.

For example, the following command sets a priority of 48 on the script file *test*:

```
qsub -p 48 test
```

Submitting to a specific queue

Unless otherwise specified, CXbatch determines which queue should receive a request by checking the value of the QSUB_QUEUE environment variable in the .login file of the user submitting the job. If this environment variable is not defined, the request is submitted to the default batch queue defined by the system manager. If no variable is found and there is no default queue defined, an error message is displayed and the request is not accepted.

You can use the `-q` option to specify a queue to run the request. If you do not specify a queue, the job is submitted to the default queue.The format for this option is

`qsub -q` *queuename*[*@hostname*]

where

queuename is the name of the queue.

hostname is the name of the host machine for that queue. If *hostname* is omitted, CXbatch submits the request to the named queue on the local machine. Your system manager can tell you which host machines are running CXbatch.

For example, the following command submits the script file, *test*, to the short queue:

```
qsub -q short
```

Naming a request

Unless otherwise specified, CXbatch assigns a request the same name as the script file (less the path name) given on the command line. If there is no script file, CXbatch assigns the default name STDIN. You can use the `-r` option to assign a name to the request. The format for this option is

```
qsub -r request-name
```

where *request-name* is the name to be assigned. The *request-name* option can have any number of characters, but its display is truncated to fifteen characters. In the actual output file, *request-name* is truncated to seven letters, plus regular extensions. If it begins with a digit, CXbatch prefixes the name with the letter R.

For example, the following command sequence assigns the name *myjob* to the interactive batch session:

```
qsub -r myjob
echo "hello world"
CTRL-d
```

Exporting all environment variables

Unless otherwise specified, CXbatch uses the current value of the following ConvexOS environment variables when a batch request is executed: HOME, SHELL, PATH, USER, MAIL. Figure 15 illustrates the default environment.

Figure 15 Default environment

```
ENVIRONMENT=BATCH
HOME=/mnt/test
MAIL=/usr/spool/mail/test
PATH=:/usr/ucb:/bin:/usr/bin
QSUB_AID=0
QSUB_HOME=/mnt/test
QSUB_HOST=mach2
QSUB_LOGNAME=test
QSUB_PATH=.:/mnt/test/bin:/usr/convex:/bin:/usr/ucb:/usr/bin:/usr/local/bin
QSUB_REQID=278.mach2
QSUB_REQNAME=STDIN
QSUB_SHELL=/bin/csh
QSUB_USER=test
```

You can use the `-x` option to export the value of all other ConvexOS environment variables with a batch request. Figure 16 illustrates sample environment variables exported when the `-x` option is used to export all environment variables.

Figure 16 All environment variables exportation

```
EDITOR=gnumacs
ENVIRONMENT=BATCH
HOME=/mnt/test
MAIL=/usr/spool/mail/test
MCSDIR=/test/work/lib/MCS
NFED=/usr/local/bin/gnumacs
PATH=:/usr/ucb:/bin:/usr/bin
PRINTER=vaxip
QSUB_AID=0
QSUB_HOME=/mnt/test
QSUB_HOST=mach2
QSUB_PATH=.:/mnt/test/bin:/usr/convex:/bin:/usr/ucb:/usr/bin:/usr/local/bi
n
QSUB_REQID=279.mach2
QSUB_REQNAME=STDIN
QSUB_SHELL=/bin/csh
QSUB_USER=test
QSUB_WORKDIR=/mnt/test
SHELL=/bin/csh
TERM=sun
USER=test
```

Submitting a request silently

When a batch request is successfully submitted, CXbatch displays the *request_id* assigned to the request at the user's terminal who submitted the request. You can stop the display of this message by using the `-z` option. This option has no effect on error messages; they are always displayed.

Redirecting output files and error messages

If you submit a request without supplying a request name, output goes to a file named STDIN.o*seq#* on the machine that originated the request where *seq#* is the sequence number assigned to the request in the queue. Errors go to a file named STDIN.e*seq#* on the machine that originated the request, where *seq#* is the sequence number assigned to the request in the queue. The sequence number assigned to the request is displayed on the user's terminal after the request is submitted.

When submitting a request from a machine running an automounting daemon, you must explicitly specify output and error files with full path names. Otherwise, output and error files may not be placed where expected.

If you are near your quota limit on your home file system when you submit a job, you will lose the information normally stored in the standard output file, STDIN.o*seq#*. If the request runs on the same machine as it was submitted, you will receive notification that the output could not be saved. If the request ran on a remote machine, you will not receive notification.

The `qsub` command includes options to control direction of output and error messages. Table 3 lists each option and its meaning.

Table 3 `qsub` options that redirect output and error messages

Option	Definition
`-e`	Redirect standard error output
`-eo`	Send error output to standard output file
`-ke`	Leave standard error output file on executing machine
`-ko`	Leave standard output file on executing machine

The following sections explain each option in detail and give examples of its use.

Redirecting standard error output

You can use the `-e` option to redirect error messages to a specified file name instead of the default file. The format for this option is

`qsub -e [`*hostname*`:][[/][`*pathname*`/]]` *filename*

where

hostname is the host machine to contain the error file. If you omit *hostname*, CXbatch sends error messages to the machine that originated the request. If you also use the `-ke` option, CXbatch keeps the error messages on the machine that executes the request.

If you omit *hostname* and the / preceding the path name, CXbatch sends the error messages to the specified file name in the current working directory, provided that `-ke` is not used. Otherwise, any portion of the path name included on the command line is interpreted in relation to the user's home directory on the standard error destination machine.

pathname is the path name of the designated file.

filename is the file name to contain the error output.

For example, the following command sends error messages for the *test* script file to the file *qsub07.redirected_err* on the CONVEX machine:

```
qsub -e mach2:qsub07.redirected_err test
```

You cannot use the `-e` option if you use the `-eo` option. Refer to the "Redirecting error messages to the standard output file" section below for details on this option.

Redirecting error messages to the standard output file

You can use the `-eo` option to redirect error messages to the standard output file instead of the standard error file. For example, the following command redirects error messages for the script file *test* to the standard output file:

```
qsub -eo test
```

Errors are then directed to the file STDIN.o*seq#*. For example, if the sequence number assigned to this request is *282* on host named *mach2*, errors are directed to a file named STDIN.o282.

You cannot use the `-eo` option if you use the `-e` or the `-ke` options. Refer to the "Redirecting standard error output" section above for details on the `-e` option. Refer to the "Redirecting

error output on the executing machine" section below for details on the -ke option.

Redirecting error output on the executing machine

You can use the -ke option to have the standard error output file created on the machine executing the request rather than on the machine originating the request. For example, the script file *test* is submitted from the host named *mach2* but executes on the host named *liberty*. The following command creates the standard error output file for the script file on the host named *liberty*, the host executing the request:

```
qsub -ke test
```

You cannot use the -ke option if you use the -eo option.

You should use the -e option to specify the path or file name. If you do not specify a path or file name, the file is placed in your home directory. Do not specify a hostname with the -e option, or the -ke option is ignored.

Redirecting standard output on executing machine

You can use the -ko option to have the standard output file created on the machine executing the request rather than on the machine originating the request. For example, the script file *test* is submitted from the host named *mach2* but executes on the host named *liberty*. The following command creates the standard output file for the script file on the host named *liberty*, the host executing the request:

```
qsub -ko test
```

You cannot use the -ko option if you specify a specific *hostname* in conjunction with the -o option. The -ko option is also meaningless if the current directory has been imported by the executing machine.

You should use the -e option to specify the path or file name. If you do not specify a path or file name, the file is placed in your home directory.

Redirecting standard output

You can use the `-o` option to send standard output to a specified file name instead of the default file name. The format for this option is

`qsub -o` [*hostname:*] [[/][*pathname/*] *filename*

where

hostname is the host machine where the output file will be sent. If you omit *hostname*, CXbatch sends the output to the machine that originated the request. If you also use the `-ko` option, CXbatch keeps the output on the machine that executes the request.

If you omit *hostname* and the / preceding the path name, CXbatch sends the output to the specified file name in the current working directory, provided that `-ko` is not used. Otherwise, any portion of the path name included on the command line is interpreted in relation to the current working directory at the time of submission.

pathname is the path name of the designated file.

filename is the file name where the standard output will be sent.

For example, the following command sends standard output for the *test* script file to the file *myjob.redirect_out* on the local machine:

```
qsub -o myjob.redirect_out test
```

Appending accounting information to stdout output file

You can use the `-y` option to append accounting information to your request's standard output file. Accounting must be enabled for the queue in which the job runs to use this option.

This information saved includes queue, host, sequence number, remote host, submission time, completion time, time spent executing in user mode, and time spent executing in the system.

For more information on accounting, please refer to the *CONVEX CXbatch System Manager's Guide*.

Specifying resource limits

`qsub` has options that allow you to set limits on resources consumed by each process within a batch request (per-process limits) and on resources that all processes within a batch request together can consume (per-request limits). If you do not specify limits, the limits set for the queue are used. If no limits are set for the queue, limits do not apply.

The format for these options is

`qsub` *option size-limit* [, *warning-limit*]

where

option	is an option that controls a resource limit. See Table 4 for per-process and per-request limit options.
size-limit	is the desired maximum limit.
, *warning-limit*	is the limit at which a warning signal is delivered to the executing process. If no *warning-limit* is set, no warning signal is delivered.

Table 4 lists each option that controls a per-process or per-request resource limit and the limit it defines.

Table 4 Per-process and per-request option limits

Option	Limit defined	Value
`-lc`	Core file size	A number representing less than 100,000,000 bytes.
`-ld`	Data segment	A number representing less than 100,000,000 bytes.
`-lf`	Permanent file	A number representing less than 100,000,000 bytes.
`-lm`	Process Memory	A number representing less than 100,000,000 bytes.
`-lM`	Request Memory	A number representing less than 100,000,000 bytes.
`-ln`	Nice value	A number between -64 and +64.
`-ls`	Stack segment	A number representing less than 100,000,000 bytes.
`-lt`	Process CPU time	*hours:minutes:seconds:milleseconds.*
`-lT`	Request CPU time	*hours:minutes:seconds:milleseconds.*
`-lv`	Temporary file	A number representing less than 100,000,000 bytes.
`-lw`	Working set	A number representing less than 100,000,000 bytes.

Note

If you specify only a *size-limit* for the process CPU time and request CPU time limits, the *warning-limit* is set to the specified *size-limit*. The *size-limit* is adjusted to 10% greater than the *warning-limit*, with a maximum adjustment of one minute greater than the *warning-limit*. This adjustment occurs only with the CPU process and request time options, `lt` and `lT`.

With the exception of the nice value, not all operating systems support these per-process limits. If you submit a request specifying a limit and the machine that runs the batch request does not enforce the specified limit, the limit is ignored.

Other implementations of NQS may support features that CXbatch does not. For example, CXbatch does not support the temporary file limits. However, if the batch system on the execution machine supports these limits, they are enforced.

For further information on resource limits, refer to *CXbatch System Manager's Guide* and qsub(1) and getrlimit(2) man pages.

CXbatch ignores all per-request limits for batch requests submitted to a CONVEX machine for execution on a CONVEX

machine. Table 5 lists each per-request option that is not applicable if the destination machine is a CONVEX machine.

Table 5 Invalid per-request limits for CONVEX machine

Option	Limit defined
`-lf`	Permanent file
`-lv`	Temporary file

Sending notification of request status

CXbatch sends mail to the user submitting the request if it cannot execute the submitted shell script. Unless otherwise specified, it does not notify the user if the request is accepted and executed. The `qsub` command includes options that allow you to send mail when a request has started or finished executing. Although mail originates on the specified machine, it is routed according to standard mail-forwarding mechanisms. Table 6 lists the options that allow sending of notification and their meanings.

Table 6 `qsub` options that allow sending notification

Option	Definition
`-mb`	Send mail when request begins execution
`-me`	Send mail when request completes execution
`-mu`	Send mail to specific user
`-t`	Signal a process when request completes execution

The following sections explain these options and give examples of their uses.

Sending mail when request begins execution

You can use the `-mb` option to have CXbatch send you mail on the originating machine when the request begins execution. For example, the following command sends mail to the user submitting the request named *myjob* when the job begins execution:

```
qsub -mb myjob
```

Figure 17 illustrates the mail received as a result of using the `-mb` option when submitting the job.

Figure 17 Mail received when job starts executing

```
Request name: myjob
Request owner: johndoe
Mail sent at: Fri Aug 24 10:43:07 CDT 1990
```

Sending mail when request completes execution

You can use the -me option to have CXbatch send you mail on the originating machine when the request finishes execution. For example, the following command sends mail to the user submitting the request named *myjob* when the job finishes execution.

```
qsub -me myjob
```

Mail sent as a result of using the -me option contains a summary of CPU time used by the request. Figure 18 illustrates an excerpt of a mail message received as a result of using -me.

Figure 18 Mail received when job completes executing

```
Request name: STDIN
Request owner: johndoe
Mail sent at: Fri Aug 24 10:47:36 CDT 1990
Request exited normally.
_Exit() value was: 0.
```

Sending mail to specified user

You can use the -mu option to have CXbatch send mail to a particular user or host instead of to yourself. The format for this option is

```
qsub -mu username [@hostname]
```

where

username is the user to receive the mail.

hostname is the machine where *username* is located.

For example, the following command sends mail to user *smith* on host *mach2* when the request begins and completes execution.

```
qsub -mu smith mach2
```

Figure 19 illustrates the mail received by the user *smith* as a result.

Figure 19 Mail sent to another user

```
>From daemon Tue Jan 24 15:39:55 1990
Received: by mach2 (5.51/4.7)
 id AA18211; Tue, 24 Jan90 15:39:53 CST
Date: Tue, 24 Jan90 15:39:53 CST
From: root (Superuser)
Subject: CXbatch request: 2172.mach2 beginning.
Apparently-To: smith
Status: R

Request name: STDIN
Request owner: johndoe
Mail sent at: Tue Jan 24 15:39:52 CST 1990

>From daemon Tue Jan 24 15:40:03 1990
Received: by mach2 (5.51/4.7)
id AA18224; Tue, 24 Jan90 15:40:01 CST
Date: Tue, 24 Jan90 15:40:01 CST
From: root (Superuser)
Subject: CXbatch request: 2172.mach2 ended.
Apparently-To: smith
Status: R

Request name: STDIN
Request owner: johndoe
Mail sent at: Tue Jan 24 15:40:01 CST 1990

Request exited normally.
_Exit() value was: 1.
```

Signalling processes when a request completes executing

You can use the `-t` option to have CXbatch signal a particular process when the request has executed. The format for this option is

```
qsub -t process-id
```

where *process-id* is the process to be signalled.

One of the following signals is sent, depending on how the request completes:

- `SIGTERM` —Request completed normally.
- `SIGUSR1`—Request was aborted while running.
- `SIGUSR2`—Request was deleted before it ran.

Embedded options

You can include qsub options in a script file and submit it as a batch request.The options must be included in the first comment block of the script file for CXbatch to recognize them.

Begin comment lines with either # (ConvexOS) or $! (COVUEshell). CXbatch interprets both the ConvexOS (#) and COVUEshell ($!) comment indicators. (COVUEshell is an optional CONVEX interface to ConvexOS that emulates the VAX/VMS Digital Command Language.) If the next nonblank characters are @$, CXbatch treats the line as an embedded option. Signify the end of the option with either the end of the line or unquoted # or $! characters. Indicate the end of the comment block with any character other than # or $! as the first character on a line.

The options embedded in the script file set the default characteristics for the batch request. Command line options override embedded values in the script file. For example, when you enter the command:

```
qsub -a 10:00am EDT testjob
```

and the option

```
-a "11:30pm EDT"
```

is embedded in the script file *testjob,* the request is run at 10:00 a.m. EDT because the value shown on the command line takes precedence over the embedded option.

Figure 20 illustrates use of embedded options in a script file.

Figure 20 Embedded options in a script file

```
#
# Batch request script example:
#
# @$-a "11:30pm EDT" -lt "21:10, 20:00"
# # Run request after 11:30 EDT by default,
# # and set a maximum per-process CPU time
# # limit of 21 minutes and ten seconds.
# # Send a warning signal when any process
# # of the running batch request consumes
# # more than 20 minutes of CPU time.
# @$-lt 1:45:00
# # Set a maximum per-process CPU time limit
# # of one hour and 45 minutes. (The
# # implementation of CPU time limits is
# # completely dependent upon the ConvexOS
# # 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.
```

Controlling queue requests

4

Once a batch job is submitted to a queue, it becomes a queue request. Once in a queue, the owner can

- Delete a request
- Place a request on hold
- Take a request off hold
- Move a request
- Change a request's priority

CXbatch operators and CXbatch managers can perform these same actions on any user's queue requests. This chapter describes how to perform these actions.

Deleting queue requests

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

Using the qdel command

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

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

where

option can be one of the following:

`-u` *username*	Allows you to delete requests other than your own, where *username* is the name of the user who owns the request. By default, only the user who submitted the request can delete it from a queue. This option allows the superuser, CXbatch manager, or CXbatch operator to delete someone else's request from a queue.
`-k`	Sends s SIGKILL (-9) signal to a request. The request then exits and is deleted. If a request is running, all processes of the request are sent a SIGKILL signal.
sig	Sends the specified signal to an executing request where *sig* can either be the signal number or the signal name found in the /usr/include/signal.h file.

request_id is the number assigned to the request when it is submitted to CXbatch. This number can be for any request, whether or not it is executing. You can specify more than one request.

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

By default, when a batch request is successfully submitted, CXbatch displays the identification number of the request on the terminal of the user submitting the request. You can also use the `qstat` command to find the *request_id*. Refer to Chapter 2, "Displaying information" for details on using `qstat`.

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

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

```
qdel -u smith 291
```

Using the delete request command

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

```
qmgr
```

The format for the `delete request` command is

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

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

By default, when a batch request is successfully submitted, CXbatch displays the identification number of the request on the terminal of the user submitting the request. You can also use the `qstat` command to find the *request_id*. Refer to Chapter 2, "Displaying information," for details on using `qstat`.

Preventing queue requests from executing

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

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

```
qmgr
```

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

Placing a queue request on hold

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

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

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

By default, when a batch request is successfully submitted, CXbatch displays the identification number of the request on the terminal of the user submitting the request. You can also use the `qstat` command to find the *request_id*. Refer to Chapter 2, "Displaying information," for details on using `qstat`.

Removing the hold on a queue request

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

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

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

By default, when a batch request is successfully submitted, CXbatch displays the identification number of the request on the terminal of the user submitting the request. You can also use the `qstat` command to find the *request_id*. Refer to Chapter 2, "Displaying information," for details on using `qstat`.

Moving a request to another queue

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

```
qmgr
```

The format for the `move my_request` command is

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

where

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

By default, when a batch request is successfully submitted, CXbatch displays the identification number of the request on the terminal of the user submitting the request. You can also use the `qstat` command to find the *request_id*. Refer to Chapter 2, "Displaying information," for details on using `qstat`.

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

Changing the queue request priority

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

```
qmgr
```

The format for the `modify request` command is

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

where

`priority =` *value* specifies the new priority of the queue request, where *value* is a number between 0 and 63; 0 is the lowest priority and 63 is the highest. A user can only decrease the priority of a request. A CXbatch manager or operator can raise a request's priority.

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

By default, when a batch request is successfully submitted, CXbatch displays the identification number of the request on the terminal of the user submitting the request. You can also use the `qstat` command to find the *request_id*. Refer to Chapter 2, "Displaying information," for details on using `qstat`.

Checkpoint Restart features 5

Checkpoint Restart is a standard feature of ConvexOS useful for application programs that must run for long lengths of time and that—once halted—cannot be started from the beginning without wasting significant time and resources. Such applications can be saved to files (*checkpointed*) either by operator intervention or by a periodically-executed script that includes Checkpoint Restart commands. If the application is then halted—either by a system crash, an apr (automatic processor recovery) event, by a scheduled shutdown, or by an operator—it can be restarted as it was when it was last checkpointed.

You can checkpoint and restart processes, or groups of processes, running under CXbatch using Checkpoint Restart features built into CXbatch. These are:

- `qsub`—Allows you to specify checkpoint capabilities when you submit the request to CXbatch.
- `qchkpnt`—Allows you to checkpoint any checkpointable request currently running.
- `qrestart`—Allows you to restart any request that meets restart requirements.

This chapter describes how to use these commands. For information on other Checkpoint Restart commands, refer to the *ConvexOS Extensions User's Guide*.

Controlling checkpointing when request is submitted

When a queue is created, the system manager specifies whether or not the queue is checkpointable. If the queue is specified as checkpointable, all jobs that are running are checkpointed if the system shuts down for any reason. If the queue is specified as not checkpointable, all jobs that are running are killed if the system shuts down for any reason. You can use the `qstat` command to find out whether or not the queue you wish to submit a job to is checkpointable or not. Refer to Chapter 2, "Displaying information," for details on using the `qstat` command.

You can override the checkpoint specification on a per job basis using options available with the `qsub` command. These options and their descriptions are listed in Table 7.

Table 7 `qsub` options related to Checkpoint Restart

Option	Effect
`-c`	Request is checkpointable
`-nc`	Request is not checkpointable
`-cp` *period*	Checkpoints request every *period* where *period* is the interval of time that must elapse between checkpoints
`-nr`	Request is not restartable
`-kc`	Request will keep its checkpoint files following an apr event
`-nkc`	Request will not keep its checkpoint files following an apr event

Each of these options is described in the following sections.

Overriding not-checkpointable default

You can use the `-c` option when you submit a job to CXbatch to checkpoint the request if it is running when CXbatch shuts down. You only need to use this option if the queue you are using is specified as checkpoint=available and you want the job automatically checkpointed if CXbatch shuts down or if you want to manually checkpoint it while it is running.

For example, the following command checkpoints the request named *my_job* if it is running when CXbatch shuts down:

```
qsub -c my_job
```

Overriding checkpointable default

You can use the -nc option when you submit a job to CXbatch to prevent a request from being checkpointed, if it is running when CXbatch shuts down. You only need to use this option if the queue you are using is specified as checkpoint=yes and you do not want the job automatically checkpointed when CXbatch shuts down.

For example, the following command does not checkpoint the request named *my_job* if it is running when CXbatch shuts down.

```
qsub -nc my_job
```

Periodically checkpointing a request

You can use the -cp option to periodically checkpoint a request. The format is

```
qsub -cp [number]unit
```

where

number is a positive integer specifying the number of *units* that must pass before checkpointing. If you do not specify a number, qsub assumes 1 unit.

unit can be minutes, hours, days, or weeks.

For example, the following command checkpoints the request named *my_job* every 20 minutes.

```
qsub -cp 20 minutes my_job
```

Preventing a checkpointed request from being restarted

You can use the -nr option to prevent a checkpointed request from being automatically restarted when CXbatch is restarted after it has been shut down. For example, the following command prevents the request my_job from being restarted if CXbatch shuts down.

```
qsub -nr my_job
```

Keeping checkpoint files following an apr event

You can use the -kc option when you submit a job to CXbatch to insure the request's checkpoint files (if they exist) are not deleted when the request aborts due to an apr event. You need to use this option only if the queue you are using is specified as keep_checkpoint_files=avail and you want the request checkpoint files preserved following an apr event. Note, this option is not allowed if the queue attribute

keep_checkpoint_files= no. For example, the following command will keep checkpoint files when the request aborts due to an apr event.

```
qsub -kc my_job
```

Preventing retention of checkpoint files following an apr event

You can use the -nkc option when you submit a job to CXbatch to insure the request's checkpoint files (if they exist) are not preserved when the request aborts due to an apr event. You need to use this option only if the queue you are using is specified as keep_checkpoint_files=yes and you want the request checkpoint files deleted following an apr event. For example, the following command will not keep checkpoint files when the request aborts due to an apr event:

```
qsub -nkc my_job
```

For more information, please refer to the qsub(1) man page.

Two methods of checkpointing after request is submitted

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

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

- User requesting the checkpoint must be the owner of the request or must have CXbatch manager or operator privileges.
- Request must be running in a batch queue.
- Request must be checkpointable. This means that the queue is set to checkpoint=available and the job was submitted with the `-c` option to `qsub`, or checkpoint=yes and the job was not submitted with the `-nc` option to `qsub`. Refer to the "Controlling checkpointing when request is submitted" section in this chapter for more details.

Using the qchkpnt command

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

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

where

option can be one of the following:

`-e` *number unit*

specifies to checkpoint the request every *number unit*, where *number* is a positive number specifying the number of *units* that must pass before checkpointing. *unit* can be `minutes`, `hours`, `days`, or `weeks`. Checkpointing begins when the request begins running and ends when the request is terminated.

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

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

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

By default, when a batch request is successfully submitted, CXbatch displays the identification number of the request on the terminal of the user submitting the request. You can also use the `qstat` command to find the *request_id*. Refer to Chapter 2, "Displaying information," for details on using `qstat`.

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

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

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

- If no errors are encountered, the exit status is zero.
- If one or more of the requests are not checkpointed, the exit status is the number of requests that did not get checkpointed.
- If a fatal error occurs and none of the requests are checkpointed, the exit status is one of the codes listed in Table 8.

Table 8 Exit statuses

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

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

Using the chkpnt request command

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

```
qmgr
```

The format for the `chkpnt` command is

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

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

By default, when a batch request is successfully submitted, CXbatch displays the identification number of the request on the terminal of the user submitting the request. You can also use the `qstat` command to find the *request_id*. Refer to Chapter 2, "Displaying information," for details on using `qstat`.

Suspending executing requests

It may sometimes be necessary to suspend a queue request that is currently executing and then later restarting the queue request. The `suspend request` and `resume request` commands allow you to do this.

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

```
qmgr
```

The format for each of these requests are described in the following sections.

Suspending an executing request

You can use the `suspend request` command to temporarily "freeze" execution of a queue request. The request is checkpointed and then execution is terminated. However, the queue request remains in the queue.

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

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

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

By default, when a batch request is successfully submitted, CXbatch displays the identification number of the request on the terminal of the user submitting the request. You can also use the `qstat` command to find the *request_id*. Refer to Chapter 2, "Displaying information," for details on using `qstat`.

Resuming a suspended request

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

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

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

By default, when a batch request is successfully submitted, CXbatch displays the identification number of the request on the terminal of the user submitting the request. You can also use the `qstat` command to find the *request_id*. Refer to Chapter 2, "Displaying information," for details on using `qstat`.

Restarting checkpointed requests

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

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

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

The format for this command is

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

where

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

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

By default, when a batch request is successfully submitted, CXbatch displays the identification number of the request on the terminal of the user submitting the request. You can also use the `qstat` command to find the *request_id*. Refer to Chapter 2, "Displaying information," for details on using `qstat`.

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

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

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

Transaction completion messages

A

This appendix lists common codes that may be returned by any part of the CXbatch system, discusses the meaning of the codes, and describes actions you should take in response to the codes. These codes include error and success codes.

The format for an error message is

TCM*machine_ code, message_text*

where

machine	indicates the type of machine on which the request was executing when the error message was issued. *machine* is either L for local or P for peer (remote) machine.
code	is the mnemonic code for the error message.
message_text	is the text explaining the reason for the error.

The error messages in this appendix are listed alphabetically by the first letter in *code*.

The codes and message text in this appendix are followed by explanatory text detailing what caused the error, what happened to the request, and, in some cases, what action you should take.

Some explanatory text may indicate that "Quota information bits are present." This message indicates that the error involved a violation of a quota limit, and another error message is displayed with additional information about the limit violation. These additional messages are listed at the end of this appendix.

TCML_ACCESSDEN **Access denied at local host.**

The transaction cannot be performed because a queue denies access to a user invoking the `qsub` command.

A request was submitted directly to a queue that can only receive requests from other pipe queues.

A user submitted a request to a queue that does not have the user or the user's group in its access list and does not have unrestricted access.

The transaction failed and should not be retried.

TCMP_ACCESSDEN **Access denied at transaction peer.**

The transaction cannot be performed because a queue denies access to the owner of a request that is being queued remotely.

Tried to submit to *queue@host* where *nqsdaemon@host* was started with the `-r` flag.

A request was submitted directly to *queue@host* (`qsub -q` *queue@host* remotely) that can only receive requests from other pipe queues.

A user submitted a request to a queue that does not have the user or the user's group in its access list and does not have unrestricted access.

The transaction failed and should not be retried.

TCML_ALREADACC **Already has access at local host.**

The transaction specified the addition of a user or group ID to a queue access set, and the user or group ID was already present in the queue access set.

The transaction failed.

TCML_ALREADEXI **Already exists at local host.**

The transaction specified the addition of a set element, and the element was already present in the set.

The transaction failed.

TCML_ALREADYSTART **Starting of CXbatch failed at local host.**

The transaction attempted to start CXbatch and was unable.

TCMP_BADCDTFIL **Corrupt request control/data file(s) detected at transaction peer. Seek staff support.**

The transaction involved an operation on request control file(s) or data file(s), and the request control or data file(s) were found to be corrupt at the peer machine.

The transaction failed and should not be retried.

TCML_CANTSTART **Starting of CXbatch failed at local host.**

The transaction attempted to start CXbatch and was unable.

TCML_CHKPNTFAIL **Checkpoint of request failed at local host.**

The transaction attempted to checkpoint a request and the checkpoint failed. This usually indicates the process is using some noncheckpointable resource, e.g. a socket.

The transaction may be retried.

TCMP_CHKPNTFAIL **Checkpoint of request failed at transaction peer.**

The transaction attempted to checkpoint a request and the checkpoint failed. This usually indicates the process is using some noncheckpointable resource, e.g. a socket.

The transaction may be retried.

TCMP_CLIMIDUNKN **Client machine id unknown at transaction peer. Seek staff support.**

The transaction cannot be performed because no machine ID can be determined on the remote machine for the client's network address.

A user submitted a request from *hostA* to *hostB* while `qmapmgr` on *hostB* did not have an mid entry for *hostA*.

TCML_COMPLETE **Transaction complete at local host.**

The transaction concluded successfully.

This is a generic transaction completion code and can be used as the final result code from the local machine for any transaction when a more specific code does not exist.

The transaction succeeded.

TCMP_COMPLETE **Transaction complete at transaction peer.**

The transaction concluded successfully.

This is a generic transaction completion code and can be used as the final result code from the local machine for any transaction when a more specific code does not exist.

The transaction succeeded.

TCMP_CONNBROKEN **Network connection lost. Retry later.**

The transaction failed because the network connection established for the transaction was severed and the machine went down.

The transaction failed.

The transaction should be retried at a later time.

TCMP_CONNTIMOUT **Network connection timeout. Retry later.**

The transaction failed because of a timeout waiting for the transaction peer to respond, causing the machine to go down.

The transaction failed.

The transaction should be retried at a later time.

TCMP_CONTINUE **Subtransaction complete at transaction peer.**

The transaction began successfully (i.e., access was granted, etc.), or the previous subtransaction of the transaction completed successfully. The server (in the client/server) is ready to perform the next subtransaction, as directed by the client.

The transaction continues.

TCML_EACCESS **File access denied at local host.**

The transaction involved a file operation that could not be performed because the protections set on the local machine denied access to the associated user.

The transaction failed and should not be retried.

TCMP_EACCESS **File access denied at transaction peer.**

The transaction involved a file operation that could not be performed because the protections set on the peer machine denied access to the associated user.

The transaction failed and should not be retried.

TCML_EFBIG **File size limit exceeded at local host.**

The transaction involved a file operation that would have increased the size of the file beyond the maximum supported at the local machine.

The transaction failed and should not be retried.

TCMP_EFBIG **File size limit exceeded at transaction peer.**

The transaction involved a file operation that would have increased the size of the file beyond the maximum supported at the peer machine.

The transaction failed and should not be retried.

TCML_EISDIR **Operation on directory is prohibited at local host.**

The transaction involved an illegal operation on a directory based on the protections set on the local machine.

The transaction failed and should not be retried.

TCMP_EISDIR **Operation on directory is prohibited at transaction peer.**

The transaction involved an illegal operation on a directory based on the protections set on the peer machine.

The transaction failed and should not be retried.

TCML_ELOOP **Symbolic link translation limit exceeded at local host.**

The transaction required the translation of symbolic links on the local machine, and the symbolic link translation limit was exceeded on the local machine.

The transaction failed and should not be retried.

TCMP_ELOOP **Symbolic link translation limit exceeded at transaction peer.**

The transaction required the translation of symbolic links on the peer machine, and the symbolic link translation limit was exceeded on the peer machine.

The transaction failed and should not be retried.

TCML_ENFILE **Insufficient file descriptors at local host. Retry later.**

The transaction cannot be attempted or completed because of a file descriptor shortage on the local machine.

The transaction failed.

The transaction should be retried at a later time.

TCMP_ENFILE **Insufficient file descriptors at transaction peer. Retry later.**

The transaction cannot be attempted or completed because of a file descriptor shortage on the peer machine.

The transaction failed. The transaction should be retried at a later time.

TCML_ENOBUFS **Insufficient buffer space at local host. Retry later.**

The transaction cannot be attempted because there are not enough buffers available on the local machine to establish the network connection required for the transaction at this time.

The transaction failed. The transaction should be retried at a later time.

TCMP_ENOBUFS **Insufficient buffer space at transaction peer. Retry later.**

The transaction cannot be attempted because there are not enough buffers available on the peer machine to establish the network connection required for the transaction at this time.

The transaction failed. The transaction should be retried at a later time.

TCML_ENOENT **Component in file path does not exist at local host.**

The transaction involved a file operation in which some element of the file path name did not exist at the local machine.

The transaction failed and should not be retried.

TCMP_ENOENT **Component in file path does not exist at transaction peer.**

The transaction involved a file operation in which some element of the file path name did not exist at the peer machine.

The transaction failed and should not be retried.

TCML_ENOMEM **Insufficient memory at local host. Retry later.**

The transaction cannot be performed because there is insufficient memory or swap space at the local machine.

The transaction failed. The transaction should be retried at a later time.

TCMP_ENOMEM **Insufficient memory at transaction peer. Retry later.**

The transaction cannot be performed because there is insufficient memory or swap space at the peer machine.

The transaction failed. The transaction should be retried at a later time.

TCML_ENOSPC **File system resource shortage at local host. Retry later.**

The transaction cannot be completed or performed because of a file system resource shortage on the local machine.

The transaction failed. The transaction should be retried at a later time.

TCMP_ENOSPC **File system resource shortage at transaction peer. Retry later.**

The transaction cannot be completed or performed because of a file system resource shortage on the peer machine.

The transaction failed. The transaction should be retried at a later time.

TCML_ENOTDIR **Invalid file path at local host.**

The transaction involved a file operation in which an element in the file path name was not a directory at the local machine.

The `qmgr Set CHEckpoint_directory` *directory-name* command specified a path that was not a directory.

The transaction failed and should not be retried.

TCMP_ENOTDIR **Invalid file path at transaction peer.**

The transaction involved a file operation in which an element in the file path name was not a directory at the peer machine.

The transaction failed and should not be retried.

TCML_EPERM **Permission denied at local host.**

The transaction involved a operation that could not be performed because the protections set on the local machine denied permission to the associated user.

The transaction failed and should not be retried.

TCMP_EPERM **Permission denied at transaction peer.**

The transaction involved a operation that could not be performed because the protections set on the peer machine denied permission to the associated user.

The transaction failed and should not be retried.

TCML_EPIPE **Fifo error at local host. Retry later.**

A transaction has attempted to write on a pipe or socket which has no process attached to read the data. This usually happens because the read process exited prematurely or was killed.

The transaction failed. The transaction should be retried at a later time.

TCMP_EPIPE **Fifo error at transaction peer. Retry later.**

A transaction has attempted to write on a pipe or socket which has no process attached to read the data. This usually happens because the read process exited prematurely or was killed.

The transaction failed. The transaction should be retried at a later time.

TCML_EROFS **Attempt to modify a read-only file system at local host.**

The transaction involved a file operation that would have altered a portion of a read-only file system at the local machine.

The transaction failed and should not be retried.

TCMP_EROFS **Attempt to modify a read-only file system at transaction peer.**

The transaction involved a file operation that would have altered a portion of a read-only file system at the peer machine.

The transaction failed and should not be retried.

TCML_ERRORRETRY **Recoverable transaction failure at local host.**
Retry later.

The transaction cannot be concluded successfully because of an error condition at the local machine that may not occur in the future.

This is a generic transaction completion code and can be used as the final result code from the local machine for any transaction when a more specific code does not exist. Explanatory text is displayed with this message.

The transaction failed. The transaction should be retried at a later time.

TCMP_ERRORRETRY **Recoverable transaction failure at transaction peer.**
Retry later.

The transaction cannot be concluded successfully because of an error condition at the peer machine that might go away in the future.

This is a generic transaction completion code and can be used as the final result code from the local machine for any transaction when a more specific code does not exist. Explanatory text is displayed with this message.

The transaction failed. The transaction should be retried at a later time.

TCML_ETIMEDOUT **Connect(2) timeout at local host. Retry later.**

The transaction cannot be completed. A request for connection on the local machine failed because the connected party did not properly respond after a period of time.

The transaction failed. The transaction should be retried at a later time.

TCML_ETXTBSY **Text file operation denied at local host. Retry later.**

The transaction cannot be completed because the involved file operation dealt with a busy text file. The operation happened in circumstances under which such action is prohibited at the local machine.

The transaction failed. The transaction should be retried at a later time.

TCMP_ETXTBSY **Text file operation denied at transaction peer. Retry later.**

The transaction cannot be completed because the involved file operation dealt with a busy text file in circumstances under which such action is prohibited at the peer machine.

The transaction failed. The transaction should be retried at a later time.

TCML_FATALABORT **Non-recoverable transaction failure at local host.**

The transaction cannot be concluded successfully because of an error condition that will not go away in the foreseeable future at the local machine.

This is a generic transaction completion code and can be used as the final result code from the local machine for any transaction when a more specific code does not exist. Explanatory text is displayed with this message.

The transaction failed and should not be retried.

TCMP_FATALABORT **Non-recoverable transaction failure at transaction peer.**

The transaction cannot be concluded successfully because of an error condition that will not go away in the foreseeable future at the peer machine.

This is a generic transaction completion code and can be used as the final result code from the local machine for any transaction when a more specific code does not exist. Explanatory text is displayed with this message.

The transaction failed and should not be retried.

TCML_FIXBYNQS **Limit set to enforceable value at local host other than originally requested.**

The transaction specified the enforcement of a quota limit value for a batch queue that cannot be enforced by the local machine. The quota limit has instead been adjusted to fall within the limit enforceable at the local machine.

The transaction succeeds.

TCML_FRUNSTATE **Forced running of request failed at local host.**

The transaction transaction attempted to force-run a request and the request was not in an allowable state.

The transaction may be retried.

TCMP_FRUNSTATE **Forced running of request failed at transaction peer.**

The transaction transaction attempted to force-run a request and the request was not in an allowable state.

The transaction may be retried.

TCML_GRANFATHER **Transaction successful at local host but one or more requests were given a grandfather clause.**

The transaction modified a batch queue quota limit to be less than the limit set for one or more previously queued batch request limits.

The transaction succeeds.

TCML_INSHUTDOWN **CXbatch is shutting down at local host.**

The transaction was to test the internal state of CXbatch before continuing, and the local CXbatch daemon on the local machine is in the process of shutting down.

TCML_INSQUESPA **Insufficient space to queue request at local host.**
Retry later.

The transaction failed because of insufficient queue space on the local machine. The transaction failed.

The local `nqsdaemon` either could not allocate memory for a new request or could not allocate a new transaction id.

The transaction should be retried at a later time.

TCMP_INSQUESPA **Insufficient space to queue request at transaction peer.**
Retry later.

The transaction failed because of insufficient queue space on the peer machine.

The transaction failed. The transaction should be retried at a later time.

TCML_INSUFFMEM **Insufficient memory at local host.**

The transaction cannot complete because there is insufficient memory on the local machine.

The transaction failed.

TCML_INSUFFPRV **Insufficient privilege at local host.**

The transaction cannot be performed because CXbatch does not grant the associated user the appropriate privileges.

A `qmgr` command that required operator or manager privileges was attempted by a user without required privileges.

A user who was not a manager attempted to raise the priority of one of his/her requests.

The transaction failed.

TCML_INTERNERR **Internal CXbatch error at local host. Seek staff support.**

The transaction could not be completed because of an internal error at the local machine.

The transaction failed and should not be retried.

TCMP_INTERNERR **Internal CXbatch error at transaction peer. Seek staff support.**

The transaction could not be completed because of an internal error at the peer machine.

The transaction failed and should not be retried.

TCML_INTERRUPT **Transaction interrupted at local host.**

The transaction could not be completed because it was interrupted by the user.

The transaction may be retried.

TCML_INVCHKTDIR **Checkpoint of request failed at local host.**

The transaction attempted to checkpoint a request and the global checkpoint directory has not been set or is invalid.

The transaction may be retried when the checkpoint directory has been set to a valid directory.

TCMP_INVCHKTDIR **Checkpoint of request failed at transaction peer.**

The transaction attempted to checkpoint a request and the global checkpoint directory has not been set or is invalid.

The transaction may be retried when the checkpoint directory has been set to a valid directory.

TCML_LOGFILERR **Cannot open or create new logfile at local host.**

The transaction designated the creation or opening of a new CXbatch log file, and the create or open operation failed.

The transaction failed.

TCMP_MAXNETCONN **Maximum network connection limit reached at transaction peer. Retry later.**

The transaction cannot begin because there are too many ongoing network transactions at the peer machine.

The transaction failed. The transaction should be retried at a later time.

TCML_MAXQDESTS **Maximum destination set cardinality reached at local host.**

The transaction specified the addition of a new queue destination for a pipe queue that would otherwise exceed the maximum number of queues allowed within a destination set for a single pipe queue. The maximum number of queue destinations in a destination set is 100.

The transaction failed.

TCMP_MIDCONFLCT **Machine id conflict between client and peer at transaction peer. Seek staff support.**

The transaction cannot begin because the respective machine ID values of the peer (server) machine and the local (client) machine do not match.

The transaction failed. The remote site should be marked as failed, and the transaction should not be retried.

TCML_NETDBERR **Local network database error at local host. Seek staff support.**

This completion code is returned when an error or an inconsistency is detected while a local client process accesses the network database on the client machine.

The transaction failed and should not be retried.

TCMP_NETDBERR **Local network database error at transaction peer. Seek staff support.**

This completion code is returned when an error or an inconsistency is detected while a server process accesses the network database on the server peer machine.

The transaction failed and should not be retried.

TCML_NETNOTSUPP **Networking not supported by implementation at local host.**

The transaction cannot begin because the local host implementation of CXbatch does not support the networking implementation required to perform the transaction.

The transaction failed and should not be retried.

TCMP_NETPASSWD **Network password verification failure at transaction peer. Seek staff support.**

The transaction cannot begin because the client has failed to supply the proper CXbatch network server password to the remote server peer machine.

The transaction failed and should not be retried.

TCML_NOACCAUTH **No account authorization at local host.**

The transaction cannot be performed because there is no account database authorization at the local machine for the user associated with the transaction.

The transaction failed and should not be retried.

TCMP_NOACCAUTH **No account authorization at transaction peer.**

The transaction cannot be performed because there is no account database authorization at the peer machine for the user associated with the transaction.

The transaction failed and should not be retried.

TCML_NOACCNOW **Does not have access now at local host.**

The transaction specified the deletion of a user or group ID from a queue access set, and the specified user or group ID was not previously present in the set.

The transaction failed.

TCML_NOESTABLSH **Unable to make connection with CXbatch daemon at local host. Retry later.**

The transaction could not be performed because the connection between the client transaction process and the local CXbatch daemon at the local machine could not be established. The nqs daemons/net daemons are not running.

The transaction failed. The transaction should be retried at a later time.

TCMP_NOESTABLSH **Unable to make connection with CXbatch daemon at transaction peer. Retry later.**

The transaction could not be performed because the connection between the transaction server and peer CXbatch daemon at the peer machine could not be established.

The transaction failed. The transaction should be retried at a later time.

TCML_NOLOCALDAE **CXbatch local daemon is not present at local host. Retry later.**

The transaction cannot be performed because the CXbatch daemon at the local machine is not running.

The transaction failed. The transaction should be retried at a later time.

TCMP_NOLOCALDAE **CXbatch local daemon is not present at transaction peer. Retry later.**

The transaction cannot be performed because the CXbatch daemon at the peer machine is not running.

The transaction failed. The transaction should be retried at a later time.

TCML_NOMOREPROC **Insufficient processes to perform transaction at local host. Retry later.**

The transaction cannot begin because there are not enough processes available on the local host to perform the transaction.

The transaction failed. The transaction should be retried at a later time.

TCMP_NOMOREPROC **Insufficient processes to perform transaction at transaction peer. Retry later.**

The transaction cannot begin because there are not enough processes available on the remote peer machine to perform the transaction.

The transaction failed. The transaction should be retried at a later time.

TCMP_NONETDAE **CXbatch net daemon is not present at transaction peer. Retry later.**

The transaction cannot be performed because the CXbatch netdaemon at the peer machine is not running.

The transaction failed. The transaction should be retried at a later time.

TCMP_NONSECPORT **Non-secure network port verification error at transaction peer. Seek staff support.**

The transaction cannot begin because the client process has connected to the remote CXbatch network daemon process using a nonsecure port.

The transaction failed. The client server should be identified as failed, and the transaction should be retried after the client server program is repaired.

TCML_NOPORTAVAI **No network port available at local host. Retry later.**

The transaction cannot begin because no network port is available with which to perform the network operations required by the transaction at the local machine.

The transaction failed. The transaction should be retried at a later time.

TCML_NOSUCHACC **No such account at local host.**

The transaction referred to a nonexistent account on the local machine, where the account must exist for the transaction to complete successfully.

An invalid account name was given to the `qmgr ADd Manager` or `ADd Users`, command. When using an account name, the account must currently exist in

/etc/passwd on the local machine. If the [uid] syntax is used, the account need not currently exist.

The transaction failed.

TCML_NOSUCHALI **No such queue alias at local host.**

The transaction involved an operation referring to a nonexistent queue alias at the local machine and therefore could not be performed.

The transaction failed and should not be retried.

TCML_NOSUCHDES **No such destination at local host.**

The transaction referred to a nonexistent queue destination for a pipe queue at the local machine, where the destination must exist for the transaction to complete successfully.

The transaction failed.

TCML_NOSUCHGRP **No such group at local host.**

The transaction referred to a nonexistent group at the local machine, where the group must exist for the transaction to complete successfully.

An invalid group name was given to the `qmgr ADd Groups` command. When using a group name, the group must currently exist in the /etc/group. If the `[gid]` syntax is used, the `gid` need not currently exist.

The transaction failed.

TCML_NOSUCHMAC **No such machine.**

The transaction refers to a machine that is not known at the local host. The machine name must be known to the local host for the transaction to complete successfully.

The transaction failed.

TCML_NOSUCHMAN **No such manager/operator at local host.**

The transaction refers to a nonexistent CXbatch manager or operator account. The account must have been previously defined as a CXbatch manager or operator for the request to complete successfully.

An attempt was made to delete a CXbatch manager using the `qmgr DElete Manager` command while the account specified did not currently have manager privileges, but existed on the local machine.

The transaction failed.

TCML_NOSUCHQUE **No such queue at local host.**

The transaction involved an operation referring to a nonexistent queue at the local machine and could not be performed.

The transaction failed and should not be retried.

TCMP_NOSUCHQUE **No such queue at transaction peer.**

The transaction involved an operation referring to a nonexistent queue at the peer machine and could not be performed.

The transaction failed and should not be retried.

TCML_NOSUCHQUO **No such quota supported at local host.**

The transaction specified the setting of a batch queue quota limit that cannot be enforced at the local machine.

The transaction failed.

TCML_NOSUCHREQ **No such request at local host.**

The transaction could not be performed because it references a request that does not exist at the local machine.

The transaction failed and should not be retried.

TCMP_NOSUCHREQ **No such request at transaction peer.**

The transaction could not be performed because it references a request that does not exist at the peer machine.

The transaction failed and should not be retried.

TCML_NOSUCHSIG **No such signal at local host.**

The transaction could not be performed because it referred to a signal that does not exist at the local machine.

The transaction failed and should not be retried.

TCMP_NOSUCHSIG **No such signal at transaction peer.**

The transaction could not be performed because it referred to a signal that does not exist at the peer machine.

The transaction failed and should not be retried.

TCML_NOTREQOWN **Not request owner at local host.**

The transaction could not be performed because it specified an operation on a request when the user associated with the transaction was not the request owner at the local machine.

A user without operator or manager privileges attempted to perform one of the following commands on a request not owned by this user:

```
qmgr MODify Request Priority

qmgr MOVe Request

qmgr CHkpnt Request

qmgr DElete Request
```

The transaction failed and should not be retried.

TCMP_NOTREQOWN **Not request owner at transaction peer.**

The transaction could not be performed because it specified an operation on a request when the user associated with the transaction was not the request owner at the peer machine.

The transaction failed and should not be retried.

TCML_PATHLEN **Resolved output filename for batch request at local host exceeds the maximum supported path length.**

The transaction could not be completed because a local transaction process reserved a queue slot on the local machine for a new batch request, and the resolved standard output or standard error path name of the request exceeds the maximum request path length supported by the CXbatch implementation at the local machine.

The transaction failed and should not be retried. The associated request must be deleted.

TCMP_PATHLEN **Resolved output filename for batch request at transaction peer exceeds the maximum supported path length.**

The transaction could not be completed because a remote transaction process reserved a queue slot on the remote machine for a new batch request, and the resolved standard output or standard error path name of the request exceeds the maximum request path length supported by the CXbatch implementation at the remote machine.

The transaction failed and should not be retried. The associated request must be deleted.

TCML_PEERARRIVE **Request arriving at local host.**

The transaction specified the deletion of a request that is in transit between two machines in the network, and the designated request is presently in the arriving state (to be delivered or is being delivered from a remote machine).

The transaction failed, and the coordinator process attempting the transaction must now retry the delete operation using the networked form of the transaction.

TCML_PEERDEPART **Request departing at local host.**

The transaction specified the deletion of a request that is in transit between two machines in the network, and the designated request is presently in the departing state (to be delivered or is being delivered to a remote machine).

The transaction failed, and the coordinator process attempting the transaction must now retry the delete operation using the networked form of the transaction.

TCML_PROTOFAIL **CXbatch protocol failure at local host. Seek staff support.**

The transaction could not be completed because of a CXbatch message protocol failure on the local machine.

The transaction failed and should not be retried.

TCMP_PROTOFAIL **CXbatch protocol failure at transaction peer. Seek staff support.**

The transaction could not be completed because of a CXbatch message protocol failure between a local client transaction process and a remote server transaction process, or a protocol failure between the remote server process and the network daemon at the remote machine.

The transaction failed and should not be retried.

TCML_QUEUESBUSY **Attempt to reset the job sequence number failed at local host.**

The transaction to reset the job sequence number failed because the system was not idle of all batch requests.

The transaction may be retried.

TCML_QUEDISABL **Queue is disabled at local host. Retry later.**

The transaction involved an operation that could not be completed because the referenced queue was disabled at the local machine.

The transaction failed. The transaction should be retried at a later time.

TCMP_QUEDISABL **Queue is disabled at transaction peer. Retry later.**

The transaction involved an operation that could not be completed because the referenced queue was disabled at the peer machine.

The transaction failed. The transaction should be retried at a later time.

TCML_QUEENABLE **Queue is enabled at local host.**

The transaction specified the deletion of a local queue that is enabled. A queue must be disabled and empty before it can be deleted.

The transaction failed.

TCML_QUEHASREQ **Queue has request(s) at local host.**

The transaction specified the deletion of a local queue that contains requests. A queue must be disabled and empty before it can be deleted.

The transaction failed.

TCML_QUENOTRUN **Queue not running at local host.**

The transaction attempted to force-run a request and the queue was not running.

The transaction should not be retried.

TCMP_QUENOTRUN **Queue not running at transaction peer.**

The transaction attempted to force-run a request and the queue was not running.

The transaction should not be retried.

TCML_QUOTALIMIT **Explicit request quota limits exceed maximums at local host.**

The transaction involved the queuing of a batch request or the modification of a batch request limit. The successful completion of the transaction would have caused the batch request to exceed one or more of the corresponding quota limits for that batch queue at the local machine.

Quota information bits are present.

The transaction failed and should not be retried.

TCMP_QUOTALIMIT **Explicit request quota limits exceed maximums at transaction peer.**

The transaction involved the queuing of a batch request or the modification of a batch request limit. The successful completion of the transaction would have caused the batch request to exceed one or more of the corresponding quota limits for that batch queue at the local machine.

Quota information bits are present.

The transaction failed and should not be retried.

TCML_REQCHKPNTING **Request already being checkpointed at local host.**

The transaction attempted to checkpoint a request and a different asynchronous checkpoint of the same request is still in progress.

The transaction may be retried.

TCML_REQCOLLIDE **Attempt to queue already existing request at local host.**

The transaction involved the queuing of a request that had the same request ID already existing at the local machine.

The transaction failed and should not be retried. The associated request must be deleted.

TCMP_REQCOLLIDE **Attempt to queue already existing request at transaction peer.**

The transaction involved the queuing of a request that had the same request ID already existing at the peer machine.

The transaction failed and should not be retried. The associated request must be deleted.

TCML_REQDELETE **Request deleted at local host.**

The transaction specified the deletion of a specific request at the local machine, and the request was deleted successfully.

The transaction succeeded.

TCMP_REQDELETE **Request deleted at transaction peer.**

The transaction specified the deletion of a specific request at the peer machine, and the request was deleted successfully.

The transaction succeeded.

TCML_REQMOVSTA **Request is not in an allowable state at local host.**

The transaction involved moving a request from one queue to another. The state of the request was not one of the allowable states: queued, holding, or waiting.

The transaction failed and should not be retried.

TCML_REQNOTCHPT **Request is not checkpointed at local host.**

The transaction attempted to checkpoint a request, and the request was queued with the '-nc' option.

The transaction failed and should not be retried.

TCMP_REQNOTCHPT **Request is not checkpointed at transaction peer.**

The transaction attempted to checkpoint a request, and the request was queued with the '-nc' option.

The transaction failed and should not be retried.

TCML_REQNOTHOLD **Request is not being held at local host.**

The transaction involved releasing a held request, but the target request was not in the holding state.

The transaction failed and should not be retried.

TCML_REQNOTSUSP **Restart of request failed at local host.**

The transaction attempted to restart a request, and the request was not in an allowable state.

The transaction may be retried.

TCMP_REQNOTSUSP **Restart of request failed at local host.**

The transaction attempted to restart a request, and the request was not in an allowable state

The transaction may be retried.

TCML_REQNOTQUE **Request is not in the queued state at local host.**

The transaction involved putting a queued request on hold, but the target request was not in the queued state.

The transaction failed and should not be retried.

TCML_REQOPHOLD **Request is being held by operator at local host.**

The transaction involved releasing a held request, but the request was being held by an operator.

The transaction failed and should not be retried.

TCML_REQRESTA **Restart of request failed at local host.**

The transaction attempted to restart a request, and the request was not in the checkpointed state.

The transaction may be retried.

TCMP_REQRESTA **Restart of request failed at transaction peer.**

The transaction attempted to restart a request, and the request was not in the checkpointed state.

The transaction may be retried.

TCML_REQRUNNING **Request is running at local host.**

The transaction specified the deletion of a specific request at the local machine, but the request was already running, and the transaction did not specify a signal. The request is not deleted and continues execution.

The transaction failed and should not be retried.

TCMP_REQRUNNING **Request is running at transaction peer.**

The transaction specified the deletion of a specific request at the peer machine, but the request was already running, and the transaction did not specify a signal. The request is not deleted and continues execution.

The transaction failed and should not be retried.

TCML_REQSIGNAL **Request was running and has been signalled at local host.**

The transaction specified the deletion or signalling of a specific request at the local machine, and the request has been signalled successfully.

The transaction succeeded.

TCMP_REQSIGNAL **Request was running and has been signalled at transaction peer.**

The transaction specified the deletion or signalling of a specific request at the peer machine, and the request has been signalled successfully.

The transaction succeeded.

TCML_ROOTINDEL **Root cannot be deleted at local host.**

The transaction specified the deletion of root as a CXbatch manager at the local machine.

The transaction failed.

TCMP_RRFUNKNMID **Request refers to unknown machines at transaction peer.**

The transaction involved the queuing of a request at a queue destination where the request referred to machine IDs not known to the transaction peer machine.

The transaction failed (the request is not queued) and should not be retried.

TCML_SECSTART **Start of CXbatch failed at local host.**

The transaction to start CXbatch using the qmgr STArt CXbatch command failed because this command is not supported when running ConvexOS/Secure.

TCML_SELMIDUNKN **Local machine id unknown at local host. Seek staff support.**

The transaction could not be performed because the local client transaction process at the local machine was unable to determine the machine ID of the local machine.

The transaction failed and should not be retried. The containing queue should be stopped, and the request requeued, if the transaction is on behalf of a previously queued CXbatch request.

TCMP_SELMIDUNKN **Local machine id unknown at transaction peer. Seek staff support.**

The transaction could not be performed because the local server transaction process at the peer machine was unable to determine the machine ID of the peer machine.

The transaction failed. The remote site should be marked as failed, and the transaction should not be retried.

TCML_SELREFDES **Attempt to create self-referential destination at local host.**

The transaction specified the addition of a pipe queue destination that would have caused a self-referential pipe queue destination set. This means that the destination set for a pipe queue contains the name of the pipe queue.

The transaction failed and should not be retried.

TCML_SHUTERROR **Shutdown error at local host.**

The transaction specified that the local CXbatch daemon at the local host be shut down, and an error occurred in the shutdown process.

The transaction failed.

TCML_SUBMITTED **Request successfully submitted at local host.**

The transaction involved queuing a request at the local machine, and the queuing transaction was successful.

Quota information bits are present.

The transaction succeeds.

TCMP_SUBMITTED **Request successfully submitted at transaction peer.**

The transaction involved queuing a request at the peer machine, and the queuing transaction was successful.

Quota information bits are present.

The transaction succeeds.

TCML_UNAFAILURE **Unanticipated transaction failure at local host.**

An unanticipated and totally unexpected error condition occurred when CXbatch tried to process the transaction at the local machine.

This is a generic transaction completion code and can be used as the code from the local machine for any transaction when a more specific code does not exist.

The transaction failed and should not be retried.

TCMP_UNAFAILURE **Unanticipated transaction failure at transaction peer.**

An unanticipated and totally unexpected error condition occurred when CXbatch tried to process the transaction at the peer machine.

This is a generic transaction completion code and can be used as the final result code from the local machine for any transaction when a more specific code does not exist.

The transaction failed and should not be retried.

TCML_UNRESTR **Access is currently unrestricted.**

This transaction specified that a queue access set be made unrestricted, and the queue access set for the named queue was already unrestricted.

The transaction failed.

TCML_WROQUETYP **Wrong queue type for transaction at local host.**

The transaction involved the queuing of a request, but the target queue on the local machine was of the wrong type for the request (e.g., submitting a batch request to a device queue and vice versa). This code is also seen when an operation that only relates to a pipe queue is attempted on a batch queue.

The transaction failed and should not be retried.

TCMP_WROQUETYP **Wrong queue type for transaction at transaction peer.**

The transaction involved the queuing of a request, but the target queue on the peer machine was of the wrong type for the request (e.g., submitting a batch request to a device queue by means of a pipe queue and vice versa). This code is also seen when an operation that only relates to a pipe queue is attempted on a batch queue.

The transaction failed and should not be retried.

The following are the quota limit violation information flags:

TCI_PP_CFLEXC	per-process core-file size limit
TCI_PP_CTLEXC	per-process CPU time limit
TCI_PP_DSLEXC	per-process data-segment limit
TCI_PP_MSLEXC	per-process memory size limit
TCI_PP_NELEXC	per-process nice execution priority limit
TCI_PP_PFLEXC	per-process permanent file size limit
TCI_PP_SSLEXC	per-process stack-segment limit
TCI_PP_TFLEXC	per-process temporary file size limit
TCI_PP_WSLEXC	per-process working set quota limit
TCI_PR_CTLEXC	per-request CPU time limit
TCI_PR_MSLEXC	per-request memory space limit

TCI_PR_NCPEXC	per-request CPU quota limit
TCI_PR_PFLEXC	per-request permanent file space limit
TCI_PR_TFLEXC	per-request temporary file space limit
TCI_NOIMPORT	import resource not available

Request completion messages

B

This appendix lists common codes that may be returned by a request and discusses the meaning and actions you should take in response to those codes. These codes include error and success codes.

The format for an error message is

RCM_*code, message_text*

where

code	is the mnemonic code for the error message.
message_text	is the text explaining the reason for the error. Some error messages may also include action that CXbatch takes, such as "Request deleted."

The error messages in this appendix are listed alphabetically by the first letter in *code*.

The error codes and message text in this appendix are followed by explanatory text detailing what caused the error, what happened to the request, and, in some cases, what action you should take.

Some explanatory text may indicate that "Quota information bits are present." This message indicates that the error involved a violation of a quota limit, and another error message is displayed with additional information about the limit violation. These additional messages are listed at the end of this appendix.

RCM_2MANYENVARS — **Too many environment variables to run batch request. Request not executed. Request deleted.**

Too many environment variables existed to execute the batch request.

No output files are returned.

The request is deleted.

RCM_2MANYSVARGS — **Too many server arguments on server execve(). Request requeued.**

Too many server arguments in server execve().

No output files are returned.

The request is requeued, and the queue is stopped.

RCM_ABORTED — **Request aborted via a signal. Request deleted.**

A shell process was terminated by a signal unrelated to CXbatch shutting down (unless something unpredictable happens, such as the request received SIGTERM signals and then killed itself with some signal besides SIGTERM or SIGKILL).

Output files (if any) are queued for return.

The request is deleted.

RCM_BADCDTFIL — **Corrupted request control and/or data files. Request not executed. Request files placed in CXbatch failed directory.**

An invalid request control file and/or data file was detected. No output files are returned. The request is placed in the failed directory on the local machine for later analysis.

RCM_BADSRVARG — **Bad argument passed to request server. Request requeued.**

A bad argument or environment variable was given to the server.

No output files are returned.

The request is requeued, and the queue is stopped.

RCM_DELIVERED **Request has been successfully delivered to destination.**

The associated request, as previously routed by a pipe queue server, has been successfully delivered to its destination.

No output files are returned.

The request remains on the destination queue.

RCM_DELIVEREXP **Request delivery time expired. Request deleted.**

The associated request, as previously routed by a pipe queue server, has not been delivered to its destination. The original completion code from the server is RCM_RETRYLATER, and the delivery expiration time on the request has been reached.

The original completion code is converted to this completion code when the expiration time has elapsed.

No output files are returned.

The request is deleted.

RCM_DELIVERFAI **Request could not be delivered. Request deleted.**

The associated request, as previously routed by a pipe queue server, could not be delivered to its destination. This completion code indicates a disastrous situation in which the request can never be delivered.

The information field of this return code must contain the TCM_ code that caused the failure.

No output files are returned.

The request is deleted.

RCM_DELIVERRETX **Request was not delivered. Retry limit exceeded. Request deleted.**

The associated request, as previously routed by a pipe queue server, has not been delivered to its destination, and the retry limit for such a transaction has been reached.

The information field of this return code must contain the TCM_ code that caused the failure.

No output files are returned.

The request is deleted.

RCM_ENFILERUN **Unable to successfully start request because of a file descriptor shortage. Request requeued.**

The request or transaction cannot be handled because of a file descriptor shortage on the local machine.

No output files are returned.

The request is requeued to run at a later time. All CXbatch queues on the local machine are stopped to conserve any remaining available file descriptors.

RCM_ENOSPCRUN **Unable to successfully start request because of a file system resource shortage. Request requeued.**

The request cannot be handled because of insufficient file system space on the local machine.

No output files are returned.

The request is requeued to run at a later time. All CXbatch queues on the local machine are stopped so as not to place additional file system space demands on the local machine.

RCM_EXECUTING **Request executing.**

This code is used in the internal message protocol between the server and shepherd processes when spawning a CXbatch batch request.

This request completion code must never be returned.

RCM_EXITED **Request exited normally.**

The batch request server exited normally.

Output files are queued for return.

Local information concerning the request is deleted.

RCM_IMPORTFAI **NFS import of directory failed. Request deleted.**

nqsimport was unable to successfully perform the import.

Output files are queued for return. The request is deleted.

RCM_INSUFFMEM **Insufficient memory to start request. Request requeued.**

The server to handle the request could not be spawned because of insufficient memory or swap space.

No output files are returned.

The request is requeued, and all queues are stopped.

RCM_INTERRUPTED **Server transaction processing aborted for CXbatch shutdown. Request requeued.**

The network queue or pipe queue server stopped transaction processing because of a CXbatch shutdown.

No output files are returned.

The request is requeued.

RCM_MIDUNKNOWN **Machine-id of request owner is no longer recognized by the execution machine. Request not executed. Request deleted.**

The local batch shell process, local network queue server, or local pipe server cannot identify the machine from which the request originated.

When the request was originally accepted by the local system, the owner machine ID (MID) was known to the local system. Since that time, however, the local host tables have changed, and the owner MID of the request is no longer recognized.

Output files are queued for return.

The request is deleted.

RCM_NETREQDEL **Request could not be successfully delivered. Previously routed request expired or was deleted at destination. Request deleted.**

The request, as previously routed by a pipe queue server, could not be delivered to its destination because the request expired or was deleted at the destination.

No output files are returned.

The request is deleted.

RCM_NOACCAUTH — **No account authorization on execution machine for mapped request owner user-id. Request not executed. Request deleted.**

The request cannot be handled because the local machine has no password entry for the request owner user ID.

Output files are queued for return.

This can occur if the local password database is changed, i.e., someone deletes an account from the password database after a request owned by the account has been queued.

The request is deleted.

RCM_NOMOREPROC — **Insufficient processes available to spawn request. Request requeued.**

The request cannot be executed because of a process shortage on the local machine.

No output files are returned.

The request is requeued to run at a later time. All CXbatch queues on the local machine are stopped to conserve remaining available processes.

RCM_NONSECPORT — **Server bind() error. Request requeued.**

The pipe queue or network queue server connected to the transaction server on a nonsecure port. The queue server is flawed.

No output files are returned.

The request is requeued, and the containing queue is stopped.

RCM_NORESTART — **Interrupted request prohibits restart on CXbatch rebuild. Request deleted.**

The request was running at the time of a CXbatch shutdown or system crash, and the request was not defined as "restartable" by the request owner. This completion code is returned directly from the local CXbatch daemon and can only occur when CXbatch is being rebooted.

Output files are queued for return.

The request is deleted.

RCM_NOSVRETCODE **Server for request did not return a completion code. Request failed. Request files placed in CXbatch failed directory.**

The network queue server or pipe queue server failed to report a completion code when it exited.

No output files are returned.

The request is placed in the failed directory, and the appropriate device or queue is stopped.

RCM_PATHLEN **Resolved stdout or stderr pathname of batch request at destination exceeds the maximum supported length. Request deleted.**

The standard output or standard error path name of a batch request when resolved by the selected pipe queue destination exceeds the maximum request path length supported by the CXbatch implementation at the receiving machine.

No output files are returned.

The request is deleted.

RCM_PIPREQDEL **Request deleted.**

The routing of the request was interrupted by a delete request at the local machine.

No output files are returned.

The request is deleted.

RCM_REBUILDFAI **CXbatch rebuild failure. Request could not be requeued. Request deleted.**

A request that was not running at the time of an CXbatch shutdown or system crash could not be requeued because of an internal CXbatch error. This completion code is returned directly from the local CXbatch daemon and can only occur when CXbatch is being rebooted.

No output files are returned.

The request is placed in the CXbatch failed directory.

RCM_REQCOLLIDE **Request collided with another previously existing request with the same request-id. The newer request has been deleted. Seek staff support.**

The request being routed collided with a previously existing request on a potential destination machine.

No output files are returned.

The request is deleted.

RCM_RESTARTFAIL **Request failed to restart.**

The restart for this request failed and the request is requeued.

RCM_RETRYLATER **Request transaction failed. Retry scheduled. Request requeued.**

The transaction for this request or subrequest cannot be carried out successfully and will be retried later.

No output files are returned.

RCM_ROUTED **Request successfully routed for delivery to destination.**

The request was successfully routed and queued at one of the remote destinations for the request, as selected by the pipe queue. Quota information bits are present.

No output files are returned.

RCM_ROUTEDLOC **Request sent to local queue destination.**

The request was successfully routed and queued at one of the local destinations for the request, as selected by the pipe queue. Quota information bits are present.

No output files are returned.

RCM_ROUTEEXP **Request routing time expired. Request deleted.**

The request has not been routed to a destination. The original completion code from the server is RCM_RETRYLATER, and the routing expiration time on the request has been reached. The original completion code is converted to this completion code when the expiration time has elapsed.

No output files are returned.

The request is deleted.

RCM_ROUTEFAI **Request could not be routed. Request deleted.**

The request cannot be routed. No destination will accept it. Every destination selected for the request by the pipe queue server has been contacted, and each destination indicated that it will not accept the request. Information bits that define the set of failure conditions are present.

No output files are returned.

The request is deleted.

RCM_ROUTERETX **Request was not routed. Retry limit exceeded. Request deleted.**

The request was not routed to any destination, and the retry limit for this type of transaction has been reached. Failed pipe routing information bits that define the set of failure conditions are present.

No output files are returned.

The request is deleted.

RCM_SEREXEFAI **Request server execve() failed. Request requeued.**

Server execve() operation failed.

No output files are returned.

The request is requeued, and the queue is stopped.

RCM_SERVESIGERR **Server killed by unanticipated signal. Request requeued.**

Network or pipe server was killed by a signal that should not have killed it.

No output files are returned.

The request is requeued, and the device or queue is stopped as appropriate.

RCM_SETJOB **The call to setjob() failed, the request was not run.**

The call to setjob(2) failed and the batch request is deleted.

RCM_SHEXEF2BIG **Too many environment variables to run request. Request deleted.**

One or both of the argv[] or envp[] argument sets to the batch request shell exceeded the maximum size supported by the underlying UNIX implementation.

Output files are queued for return.

The request is deleted.

RCM_SHUTDNABORT **Request aborted for CXbatch shutdown. The request was defined as unrestartable, and so the request has been deleted.**

The server or shell process for the request exited because of a signal sent because of a CXbatch shutdown, and the request is not restartable.

Output files are queued for return.

The request is deleted.

RCM_SHUTDNREQUE **Request aborted for CXbatch shutdown. The request has been requeued for later restart.**

The server or shell process for the request exited because of a signal sent because of a CXbatch shutdown, and the request is restartable.

When CXbatch is restarted, the request is requeued for continued execution.

RCM_SSHEXEFAI **Request shell execve() failed. Request requeued.**

The attempt to execute the shell chosen by the system to interpret the batch request shell script failed.

Output files are queued for return.

The request is requeued, and the queue is stopped.

RCM_STAGEOUT **Output file successfully returned to destination.**

The output file was successfully returned to its intended destination.

The output file is deleted.

RCM_STAGEOUTBAK **Output file could not be returned to primary destination. Output file successfully returned to backup destination in user home directory on the execution machine.**

The output file could not be returned to its intended destination because of circumstances not allowing retry attempts. Instead, the file was successfully returned to its backup destination.

The information field of this return code must contain the TCM_ code that caused the failure at the primary destination.

The output file is deleted.

The request completes.

RCM_STAGEOUTFAI **Output file could not be returned to primary or backup destination.**

The output file could not be returned to its intended or backup destination because of error conditions that prohibit retrying the operation.

The information field of this return code must contain the TCM_ code that caused the failure at the primary destination.

The output file must be deleted.

RCM_UNABLETOEXE **Unable to execute request. Request deleted.**

The request could not be executed for reason(s) identified in the information bit vector of the completion code. Quota information bits are defined.

Output files are queued for return.

The request is deleted.

RCM_UNAFAILURE **Request failed. Request files placed in CXbatch failed directory.**

An unanticipated error condition occurred while CXbatch was trying execute the request or perform a transaction operation for the request.

If CXbatch was attempting to return an output file to the submitter, the associated output file is deleted, and the output file loss is recorded for the batch request.

All other operations result in the request being placed in the failed directory, and any associated output files are deleted.

RCM_USHBRKPNT **Unsupported shell breakpoint encountered. Request deleted.**

The shell chosen by the user to execute the batch request stopped at a breakpoint.

Output files are queued for return.

The request is deleted.

RCM_USHEXEFAI **Request shell execve() failed. Request deleted.**

The attempt to execute the shell chosen by the system to interpret the batch request shell script failed.

Output files are queued for return.

The request is deleted.

The following are the request completion flags:

RCI_ACCESSDEN	Access denied
RCI_CLIMIDUNKN	Client machine ID is unknown at transaction peer
RCI_EFBIG	File size limit exceeded
RCI_FATALABORT	Nonrecoverable transaction failure
RCI_MIDCONFLICT	Machine ID conflict between client and destination
RCI_NETNOTSUPP	Networking not supported at CXbatch site.
RCI_NETPASSWD	Network password verification error
RCI_NOSUCHQUE	No such queue
RCI_PEERINTERR	CXbatch internal error at transaction peer
RCI_PEERMIDUNKN	Local machine ID is unknown at transaction peer
RCI_PEERNETDB	Network database error at transaction peer
RCI_PEERNOACATH	No account authorization at transaction peer
RCI_PROTOFAIL	CXbatch protocol failure
RCI_QUOTALIMIT	Explicit request quota limits exceed maximums
RCI_RRFUNKNMID	Request refers to machine IDs unknown at transaction peer
RCI_WROQUETYP	Wrong queue type for request
RCI_UNAFAILURE	Unanticipated transaction failure

Index

R

S

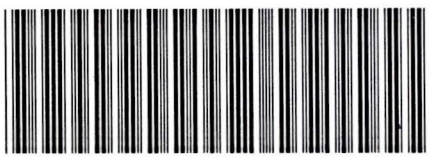
T

U

V

ORDER NUMBER
DSW-183

DOCUMENT NUMBER
710-002730-207



CONVEX PRESS