The
Connection Machine
System

CMOST Version 6.1.1 Release Notes

Version 6.1.1
March 1992

Thinking Machines Corporation
Cambridge, Massachusetts

First printing, March 1992

Thinking Machines Corporation
245 First Street
Cambridge, Massachusetts 02142–1264
(617) 234–1000/876–1111

CMOST Version 6.1.1 Release Notes

1 Overview

These release notes document Version 6.1.1 of CMOST, the CM Operating System, Timeshared, and the system software (utilities, compilers, and libraries) that runs on top of it. CMOST Version 6.1.1 contains software patches to CMOST Version 6.1; these patches consist of additional functionality and bug fixes.

Appendix A of these release notes lists the contents of this patch release.

1.1 Requirements

Recompiling and Relinking

All existing programs should be relinked to run with CMOST Version 6.1.1. Recompiling is not necessary.

Sun OS Version

CMOST Version 6.1.1 requires SunOS Version 4.1.1 or 4.1.2, or VAX ULTRIX Version 4.2.

1.2 Changes to Header Files

The header file `paris.h` has been changed to support the new C* accessor routines `CM_set_error_stream` and `CM_get_error_stream` (see Section 6.1).

2 CMOST

2.1 I/O

This patch tape includes an `fsserver` executable for the DataVault as well as one for the front end. The DataVault executable must be copied to the MicroVAX from the installation directory on the front end.

Enhancements

1. The CMFS fileserver (the `fsserver`) has been enhanced to provide improved performance through the use of a serial I/O buffer cache and the asynchronous writing of inode files to the MicroVAX disk. The use of these features is controlled via two new switches to the `fsserver`: `-c` and `-i`. A new manual page (`fsserver.8`) describing these switches is provided on the tape.

   - The `-c` *integer* switch changes the size of the DataVault's buffer cache. Currently, the default buffer cache is 32 buffers (of one block each).

     A buffer cache of optimal size for a particular system can increase the performance of serial CM file system operations. The optimal size of the buffer cache is dependent upon how much physical memory is typically free on the DataVault MicroVAX. If the cache is too big, performance will be degraded by `fsserver` paging. Until that point, the bigger the cache, the better the performance.

     To disable the buffer cache, start the `fsserver` with `-c0`.

   - The `-i 1|0` switch enables and disables asynchronous inode writing. When enabled, this improves the performance of file extension (especially beneficial to serial I/O performance). Specifically, when a file is extended under "asynchronous inode mode," the **fsserver** delays writing the file's inode to disk until one of the following events occurs:

- The file shrinks in size.
- A request is made to close the file, and no other user has it open.
- The file server is idle for approximately 20 seconds.
- The file server is shut down via a SIGTERM signal.

When asynchronous inode writing is enabled, users can disable it on a per-file-descriptor basis by supplying the **CMFS_O_FSYNC** flag when opening the file. (A new **CMFS_open.3** man page that incorporates this new flag is provided on this patch tape.) Note that **CMFS_O_FSYNC** disables asynchronous inode writing for write operations, but not for **ftruncate** operations (for example, **CMFS-[serial]-ftruncate**).

In this patch release, asynchronous inode writing is disabled by default. In future versions of CMOST, however, this feature may be enabled by default.

2. Message passing between the CMFS library and the **fsserver** has been speeded up. This reduces latency of CM file system operations.

3. The **cmdd** command now handles IBM labels. A new version of the **cmdd** man page is on this patch tape and is included in Appendix B of these release notes.

4. The DataVault can now act as an NFS server. This enhances usability of the CMFS file system by allowing UNIX commands and calls to operate on the CMFS file system and its files (via serial I/O only). Since this release of NFS is preliminary, its performance may not be as good as CMFS software performance.

   Three new man pages for the NFS server daemons, **ugidd**, **unfsd/unfsmntd**, and **unfsd_exports**, are on this patch tape and are included in Appendix B of these release notes.

5. A new call for the CM-HIPPI, **CMFS_get_extra_count** (C/Paris only), returns the number of virtual processors that received extra bits the last time the local program read data from a socket into the CM and the **CMFS-parallel--recv-always** or **CMFS-read-file-always** call returned a short bit count. The call's syntax is:

```
int = CMFS_get_extra_count(int sock);
```

A man page for this new call is in the *CM-HIPPI User's Guide for the CM-2*, Version 6.1 Beta 2.

Fixes

1. By default, the commands **cmcp**, **copyfromdv**, and **copytodv** copy not only the data file but also the associated attribute file. In CMOST Version 6.1, however, if the command failed to copy the data file — because of insufficient permission, for example — the command's subsequent attempt to copy the attribute file caused an abort. Now, if the command fails to copy the data file, it does not try to copy the attribute file.

2. Programs that used a VMEIO could become out of sync with the **fsserver** on certain error conditions. This is now fixed.

3. The 6.1 **fsserver** now works with programs linked with the 6.0 CMFS library.

4. Programs that attempt to read a file at EOF no longer occasionally receive the **target busy timeout** error.

5. I/O via the Ethernet to and from files two gigabytes or larger in size now works properly.

6. CM-HIPPI:

    a. Programs that perform direct data transfers between the CM-HIPPI and the DataVault no longer lose the ability to communicate with the DataVault **fsserver** after some error conditions.

    b. It is now possible to fully intermix serial and parallel operations on the CM-HIPPI. (Previously, if all serial operations weren't multiples of 512 bits, subsequent parallel operations would fail.)

7. It is no longer an error to execute **cmchmod**, **cmchgrp**, or **cmchown** on an attribute file that has no associated inode file.

8. A bug in reading directories that caused **cmdu** to loop occasionally has been fixed.

9. The commands **dv_disk_util** and **dv_util_vmeio**, when inadvertently run without root privileges, no longer leave an unusable **raw-disk-special-file** on the DataVault. Also, **dv_disk_util** and **dv_util_**

vmeio have been relinked with the 6.1.1 version of the Paris and CMFS libraries.

10. The **CMFS-rename** routine and the **cmmv** command now verify that the user has write permissions for both the directory in which the file (or directory) resides and the one that the file (or directory) will reside in.

11. The overhead timing test that is part of **dvtest2** now prints more detailed information about CMFS overhead. In addition, **dvtest2**'s usage message is now correct.

12. The **cmtar** command now allows reading or writing more than two gigabytes to a single tape. This functionality is important when using Exabyte drives.

13. Under 6.1 timesharing, if some processes were doing I/O to the DataVault when the timesharing system had to swap jobs out to the DataVault, both the swap and the process I/O could fail. This problem has been corrected.

Known Problems

1. Running **cmfsck** without the **-p** switch, or running **cmfsck** on a dirty file system, changes the last access time of all files.

2. To prevent **fsserver** problems, be sure the permissions on the two DataVault root directories (usually **/dv1** and **/dv2**) are the same.

2.2 Operating System

Enhancements

1. The command that manipulates the CM's access-control lists, **cmacl**, now has a **-c** option. The **-c** option modifies the access-denial lists, which, since they are checked after the access-granted lists, act as a list of exceptions to the access-granted lists.

   Using the **-c** option is practical where CM access is available to most users and groups: The access-granted lists can remain empty, indicating that access is unrestricted, while the exceptions are made by executing

   ```
   %  cmacl -c [-u|-g] name [name...]
   ```

where *name* is a user name or a group name, consistent with whether **-u** or **-g** is specified. Note that the primary group ID (set by **login** or **newgrp()**) is the only group ID checked for denial of access due to group membership.

When **cmacl -c** is executed without any *name* arguments, the contents of the access-denial list is printed.

An updated man page for **cmacl** is on this patch tape.

2. The CM device driver now enforces CM connect-time limits, which are set by a new command, **cmcountdown**. Its syntax is:

    ```
    cmcountdown [-u user-name connect-time-limit]
                [-g group-name connect-time-limit]
    ```

    connect-time-limit is the number of seconds the user or group can run jobs on the CM. Note that a group accrues connect time from each member's CM activity; that is, the limit applies to the group as a whole, not to each member.

    A process that runs under timesharing accumulates connect time only when the job is scheduled on the CM, not when it is simply waiting to use the CM. Under exclusive mode, connect time is the same as the difference between detach and attach times.

    When **cmcountdown** is executed without arguments, it prints a list of the current connect-time limits.

    A man page for **cmcountdown** is on this patch tape and is included in Appendix B of these release notes.

3. A new system verifier, **cmverify**, aids in preventive maintenance. A man page for **[cmverify** is on this patch tape and is included in Appendix B of these release notes.

4. As of CMOST 6.1.1, multiprocessor Suns (Sun 600-MP series) are supported as Connection Machine front ends.

Fixes

1. Accounting:

    a. The accounting command **cmsa**, which coalesces and displays CM job accounting records, has been changed:

(1) In CMOST Version 6.1, the value of `%cm.processors` reported by `cmsa` was always the full size of the CM. In Version 6.1.1, `%cm.processors` reports the true value represented in the accounting file.

(2) `cmsa` now tests the contents of the accounting file's data structures for validity, reducing the likelihood of erroneous output.

(3) `cmsa` now prints an additional record on start-up containing the process creation time of the oldest process in the accounting file; the `First` and `Last` records reflect the times of the earliest attach and latest detach, respectively. The `cmsa.awk` script has been modified to understand this new field. Users who have written their own versions of `cmsa.awk` should make similar changes to their scripts.

In addition, accounting summaries generated by the `cmacct` command now reflect the earliest attach time and the last detach time found in the accounting file, so that `cmacct` accounting summaries no longer exaggerate the period of time that they cover. Previously, the summaries stated that they covered a period of time longer than expected because of `cmacct`'s use of the process creation time of the first and last processes in the accounting file rather than the earliest attach time and last detach time.

b. By default, the CM accounting daemon, `cm-acctd`, writes variable-length records to its accounting file, which minimizes wasted disk space but increases the risk of file corruption. A new `cm-acctd` flag, `-F`, allows the system administrator to choose increased log reliability by forcing the accounting daemon to write fixed-length records (of 512 bytes) and perform an `fsync()` operation to flush all records to disk as they arrive.

Regardless of whether the `-F` flag is specified, `cm-acctd` now performs a consistency check on the accounting file on start-up, which eliminates the possibility that a partial record or corrupted datum will cause the `cm-acctd` to crash. Also, `cm-acctd` now logs error conditions to the CM logger facility.

In accordance with the new `-F` switch to `cm-acctd`, the `pracct` command now handles both variable length and fixed length accounting records, and `cm-logger` now understands the subsystem, "CM Accounting Daemon."

c. Another new flag to **cm-acctd**, **-p** *directory*, allows the file containing the process ID of the CM accounting daemon to be placed in a directory other than the default (**/usr/spool/cm**). For example:

```
cm-acctd -p /tmp /tmp/my-accounting-file
```

creates the file **/tmp/***hostname***.cm-acctd.pid** and the file that will contain the accounting data, **/tmp/my-accounting-file**.

d. The script that rotates log files and deletes old ones, **newcmlog**, also now supports the **-p** *directory* option, allowing it to find the accounting daemon's process ID file so that it can signal the daemon that the log files are being rotated. (If **-p** is not specified, **newcmlog** looks for the process ID file in **/usr/spool/cm**.) Specify the **-p** flag to **newcmlog** whenever the **-p** flag is used with **cm-acctd**. For example:

```
newcmlog -a /tmp/my-accounting-file -p /tmp
```

rotates the accounting file **/tmp/my-accounting-file** and signals the accounting daemon to close its open files and start writing to new log files.

2. Timesharing:

a. The **-current** switch has been removed from the **cmts-admin** command, because the functionality provided by this switch is already available by running **cmts-admin** with no arguments. In addition, **cmts-admin** now prints error messages upon encountering error conditions.

b. The **cmts-shutdown** command now prints an error message if for any reason it is unable to shut down timesharing. In addition, **cmts-shutdown** now sends a SIGURG and a SIGINT signal (rather than a SIGKILL and a SIGTERM, as previously documented) so that processes that had been running under timesharing no longer must be killed by hand. A new man page for **cmts-shutdown** is on this patch tape.

c. A bug that caused processes to fail to get a process slot from the **ts-daemon** has been fixed.

d. Timesharing would occasionally terminate an application and log a message on the system console referring to **cxsw error 1**. This resulted from a race condition within the CM device driver and is now corrected.

3. The following bugs regarding attaching are now fixed:

    a. If an application began execution on the CM via the auto attach mechanism, the CM safety mode would not be set in accordance with the value of the environment variable **CM_DEFAULT_SAFETY**. The value of this environment variable is now used to set the current CM safety mode, regardless of the method used to attach to the CM. (**CM_DEFAULT_SAFETY** is also now applied properly to jobs that run under NQS.) See also Section 2.2.3, number 5.

    b. A **cmattach** bug caused both User A and User B to be detached under the following scenario:

        User A: **cmattach -iX**

        User A: **cmattach -iY** (from within the **cmattach** subshell)

        User B: **cmattach -iX**

        User A: **exit** (exiting the **cmattach** subshell)

        This problem, which affected sites with multiple interfaces on a single front end, is now fixed.

    c. If a user was attached to, for example, 8K processors on a timeshared sequencer, and then issued the command, **cmattach -p16k**, the command would appear to succeed when in fact the user would only be attached to the original 8K processors.

    d. The command, **cmdetach** *user*, when *user* was attached to a timeshared interface, would reply: **Are you sure?** *user/root* **appears to be running timesharing on that interface.**

        Unstated in this query is the fact that the **ts-daemon** and all timeshared users would be detached from the interface, along with *user*. Consequently, **cmdetach** has been changed so that it refuses to detach a timeshared interface if the request is made via **cmdetach** *user*. It will detach a timeshared interface if the request is made via **cmdetach** *interface-number*.

4. If **cmrenice** was invoked with no arguments, it would abort with a segmentation fault. **cmrenice** now prints an appropriate usage message and exits cleanly under these conditions.

5. **coldboot-paris** and **cmlist** have been compiled with the new Paris library.

6. There was a bug in the **cmman** command that caused it to improperly determine the location of the installed CM man pages on a VAX. This bug is now fixed.

7. On CM-200 systems only, the **cmpowerup** command and the (**cm:powerup**) function in Lisp would set the CM clock speed incorrectly. **cmpowerup** and (**cm:powerup**) now set the CM clock speed according to the **:clock-speed** parameter in the CM configuration file, **configuration.lisp**. The binary configuration file, **/usr/spool/cm/configuration.bin**, must be rebuilt before this fix will take effect. To do this, delete the file **/usr/spool/cm/configuration.bin** and run the **cmfinger** command as root.

8. CM timers running under timesharing no longer produce wrong timings. Previously, the performance monitor register was reloaded incorrectly when a process was being switched in. If only one process was doing timings under timesharing, the timer could be fairly accurate, but extremely wrong timings (including negative values) could result if more than one currently running process was using the timers.

Documentation Errors and Clarification

1. On page 13 of the Version 6.1 *CM System Administrator's Guide*, the setting of the **:clock-speed** parameter for a CM-2 is listed incorrectly as **4- Mhz**. The correct setting of this parameter is **7-Mhz**.

   The table below clarifies the appropriate settings of the **:clock-speed** and **:bus-speed-code** parameters of the CM configuration file:

CM	:clock-speed	:bus-speed-code
CM-2	7-Mhz	**0xba**
CM-200	8-Mhz	**0xaa**
CM-200	10-Mhz	**0xdd**

   Running a CM with these parameters set at values different from those listed in this table is not supported and will result in unpredictable behavior.

2. Page 74 of the Version 6.1 *CM System Administrator's Guide* reports incorrectly that the **auto-attach** feature is disabled by default. In fact, the **auto-attach** feature is enabled by default. To disable it, add the following entry to **/etc/cm-base-system-config** (capital letters necessary):

```
ALLOW-AUTO-ATTACH = NO
```

3. Page 85 of the Version 6.1 *CM User's Guide* incorrectly lists the name of a **coldboot** function as **CM_coldboot**. The correct name is **CM_cold_boot** (note the second underscore).

4. Section 5.2 of the Version 6.1 *CM User's Guide* discusses the **CM_attach** routines (**CM_attach**, **CM_attach_to**, etc.). The following caveat should be appended to that discussion:

    The **CM_attach** routines must be called *before* any parallel variables are defined. This means that a program that uses **GLOBAL** parallel variables (C*) or CM arrays in **COMMON** (CM Fortran) cannot use the **CM_attach** routines, since **GLOBAL** parallel variables and CM arrays in **COMMON** are defined by the run-time system before the user's main routine is invoked.

5. Section 6.1 of the *CM User's Guide*, which discusses run-time safety checking, omits an explanation of the interaction between the **CM_ DEFAULT_SAFETY** environment variable and the **cmsetsafety** command. When **CM_DEFAULT_SAFETY** is on, it overrides the **cmsetsafety** command. That is, in order for the **cmsafety** command to turn safety checking on and off for a program, **CM_DEFAULT_SAFETY** must be set to off (it is off by default). Also note that **cmsetsafety** has no effect in an auto-attached program.

Known Problems

1. **vbutil**'s Test 19 (the **VMEFEBI** IllegalOps test) is disabled on Sun 4/600 systems, as it generates VMEbus timeout conditions that cannot be safely handled on these systems.

2.3 NQS

This patch release of NQS does not require an NQS database rebuild.

Fixes

1. The following bugs relating to enforce mode are fixed:

   a. If a sequencer was in use from one front end, and another front end tried to attach to that sequencer on behalf of an enforce-mode NQS batch queue, the enforce-mode attach would fail when it should have succeeded. When the enforce-mode attach failed in this manner on a VAX computer, NQS enforce mode left the interface attached even though the sequencer was not attached.

   b. Any jobs running on the CM resource belonging to a NQS enforce-mode queue would be forcibly detached when the enforce-mode queue started up. This bug is fixed so that only those jobs submitted from a queue with lower priority than the enforce-mode queue are detached when the enforce-mode queue starts up.

2. If a system administrator had not built the NQS database before trying to run **`nmapmgr`**, **`nampmgr`** would report a fatal error and exit. The error message now tells the administrator what to do to fix the problem.

3. Batch queues are now processed in priority order when a restriction window opens.

4. Pipe queues generally did not work as released in CMOST Version 6.1 because of a configuration problem. As of this release, however, NQS can run jobs submitted to a pipe queue. Following are some pipe-queue-related enhancements:

   a. NQS no longer mistakenly applies a wall-clock limit to a pipe queue.

   b. Jobs remain in the pipe queue until a destination queue that can run the job right away becomes available. (The previous behavior allowed a job to be submitted to a destination queue that was stopped or that didn't have a CM resource available.)

   c. Inter-machine permissions: NQS now understands **`NIS`** (**`YP`**) syntax in **`/etc/hosts.equiv`**. NQS's parsing of **`.rhosts`** has been enhanced:

(1) Trailing whitespace is now allowed after the hostname and username in the `.rhosts` file.

(2) If a user is required to have a `.rhosts` file for cross-machine authentication but does not, the user now gets a **`no access authorization`** error message. (The previous behavior allowed the request to keep trying to route, up to the pipe queue retry limit, and on exit provided no helpful interpretation of the problem.)

d. The order in which a pipe queue attempts to submit jobs to its destination queues has been made more useful: The destination queues are first sorted alphabetically by name, and then by machine ID (MID). When a pipe queue receives a request, it attempts to submit it to each queue in the sorted list, in turn, until one accepts the job.

For example, given the batch queues **`q15M@cmfe3`**, **`q1h@cmfe1`**, **`q2h@cmfe2`**, and **`q6h@cmfe3`**, the following pipe queue destination lists would work appropriately:

Pipe Queue	Desired Destination Queue Order
`cm15m`	`q15m@cmfe3, q1h@cmfe1, q2h@cmfe2, q6h@cmfe3`
`cm1h`	`q1h@cmfe1, q2h@cmfe2, q6h@cmfe3`
`cm2h`	`q2h@cmfe2, q6h@cmfe3`
`cm6h`	`q6h@cmfe3`

For simplicity, we recommend choosing unique names for all pipe queues so that sorting by MIDs is not necessary.

Note that the description of destination-queue sorting given in Section 3.13.2 of the *CM System Administrator's Guide* is incorrect.

Known Problems

1. If a pipe queue on a CM-2 feeds jobs to a CM-5, or vice versa, an incompatibility between CM-2 NQS (software versions 6.1.1 and earlier) and CM-5 NQS (software versions 7.1.3 and later) will cause some error codes generated on one machine to be misinterpreted on the other.

3 Run-Time System

3.1 Enhancements

1. The file `cmrt.h` is on this patch tape.

2. The performance of `CMRT_intern_geometry` and `CMRT_intern_detailed_geometry` has been improved:

   a. Programs that attached via the `cmattach` command no longer generate unnecessary calls to an internal routine, `CM_attached`, used by the auto-attach mechanism.

   b. The geometry creation routine has been speeded up via a geometry caching system.

   These improvements improve the performance of programs that create large numbers of geometries.

3.2 Fixes

1. CM Fortran programs would occasionally fail under timesharing with a message about `out of scratch memory`, when there should have been plenty of free memory. This problem has been fixed.

4 Paris

4.1 Enhancements

1. A dynamically linked, shared version of the Paris library is now available. Using this library greatly reduces application size and link time.

4.2 Fixes

1. CM Fortran/Paris array section transfers would fail because of a bug in **cross-vp-move**. This has been fixed.

2. Calling **CM_get_1L** or **CM_send_1L** with overlapping arguments, although technically illegal, would work but produce a message (only under timesharing) saying, **Compress heap called from within-vp-fields, not compressing**. These functions now work without producing the message.

3. The CMOST Version 6.1.1 of **CM_rank_1L** returns its performance level to that of Version 6.0. In addition, **CM_rank_1L** has been further speeded up for certain VP ratios.

4. **CM:send-to-shared-queue32-2L** on certain hardware configurations would occasionally generate false message-parity errors. This is now fixed.

5. New microcode corrects CM-200-specific potential timing problems.

5 Fortran

5.1 Enhancements

1. A checkpointing subroutine call, **ckpt_periodic**, is now available from Fortran. This subroutine is documented in the Version 6.1 *CM User's Guide*, but was not actually present in CMOST 6.1.

2. The F77 version of **attach-fort.h**, a header file that defines Fortran interfaces for **CM_attach**, is included on this patch tape.

3. The header file **/usr/include/cm/ckpt-fort.h**, which allows checkpointing to be done from Fortran, is included on this patch tape.

6 C*

6.1 Enhancements

1. For various reasons, C* users are not able to access **CM_error_stream**, a **paris.h** variable that redirects some CM functions' output from **stderr** or **stdout** to another file. To perform this task for C* users, there are two new accessor routines:

```
#include <stdio.h>
#include <cm/paris.h>

main()
{
    FILE    *foobar, baz;

    foobar = fopen("myfile","w")
    CM_set_error_stream(foobar);

    baz = CM_get_error_stream();
    [...etc]
}
```

   C/Paris programmers can also use **CM_set_error_stream** and **CM_get_error_stream**.

7 *Lisp

7.1 Bug Fixes

1. The *Lisp example files mentioned in the *Lisp documentation were not present in 6.1. The examples are included in this patch tape and will be in *directory*/**cm-optional/starlisp/interpreter/ f6101**, where *directory* is the directory into which this patch tape is **tar**'ed.

Appendix A

CMOST Version 6.1.1 contains the following:

- New microcode for CM2/CM200
- Paris
  - regular (compiled `-O`), Sun/VAX
  - profile (compiled `-pg`), Sun/VAX
  - Shareable libraries for Sun
- Slicewise RTS
  - All versions, CM2/200, Sun/VAX
- Sys-Commands
  - **`cmsa`**, **`cmsa.awk`**
  - **`cm-acctd`**
  - **`cmacl`**
  - **`pracct`**
  - **`cmrenice`**
  - **`cmverify`**
  - **`cmts-admin`**
  - **`coldboot-paris`**
  - **`cmcountdown`**
  - **`cmts-shutdown`**
- User-Commands
  - **`cmpowerup`**
  - **`cmlist`**
  - **`cmattach`**
  - **`cm-logger`**
  - **`cm-update-config`**

  - **cmman**
- **Ts-daemon**
- NQS
  - All commands and all daemons for the Sun and VAX
- Man pages
  - **cmdd.1**, **CMFS_open.3**, **unfsd_exports.5**, **fsserver.8**, **ugidd.8**, **unfsd/unfsmtd.8**, **cmacl.8**, **cmverify.8**, **cmcount-down.8**
- Header files that have changed since CMOST Version 6.1 final
  - Also **cmrt.h**
- CM device driver *et al.*
- CMFS
  - **libcmfs.a**
  - **fsserver**
  - **dvtest2**
  - **cmdd**
  - **dv_disk_util**
  - **cmtar**
  - **dv_util_vmeio**
  - **cmcp**
  - **copyfromdv**
  - **copytodv**
- Auxiliary libraries:
  - **libckpt**
  - **libcmfe**
  - **libtoolkit**
- DataVault/Microvax
  - **fsserver**
  - **nfsd**
  - **nfs_mountd**
  - **ugidd**

Appendix B

The remainder of this manual contains updated man pages for the following commands and daemons:

- `cmdd`
- `cmcountdown`
- `cmverify`
- `ugidd`
- `unfsd`, `unfsmntd`
- `unfsd_exports`

NAME

cmdd - Copies an input file to an output file, converting data as specified.

SYNTAX

cmdd [**-fromdv**] [**-todv**] [**ifbs**=*n*]
[**obfs**=*n*] [**-a**] [**if**=*name*] [**of**=*name*]
[**obs**=*n*] [**bs**=*n*] [**cbs**=*n*] [**skip**=*n*] [**files**=*n*] [**seek**=*n*] [**count**=*n*]
[**conv**=*value*] [**label**=[**isl** | **inl** | **osl** |
onl | **obl**]] [**vsn**=*List-of-vol-serial-nos*]
[**dsn**=*List-of-dataset-names*] [**rformat**=*fmt*:*blen*:*rlen*]

ARGUMENTS

-fromdv Use the input coming from a CMFS file. (Otherwise, input is assumed to be coming from the computer that executes **cmdd**.) The **if**=*name* option must be used with **-fromdv**.

-todv Send output to a CMFS file. (Otherwise, output is assumed to be going to the computer that executes **cmdd**.) The **of**=*name* option must be used with **-todv**.

ifbs=*n* Use the input coming from a StorageTek tape drive, with a fixed block size of *n* bytes, where 0 < *n* < 64K. (Otherwise, input is assumed to be coming from the computer that executes **cmdd**.) This option is used to put the StorageTek tape drive in fixed-block mode for improved performance.

ofbs=*n* Send output to a StorageTek tape drive, with a fixed block size of *n* bytes, where 0 < *n* < 64K. (Otherwise, output is assumed to be going to the computer that executes **cmdd**.) This option is used to put the tape drive in fixed-block mode for improved performance.

-a Append the input to the output file (rather than rewrite it if it already exists).

if=*name* Input file name. If the input is a CMFS file, this option is required. If this option is not specified, the default is the standard input of the computer that executes **cmdd**.

of=*name* Output file name. If the output is a CMFS file, this option is required. If this option is not specified, the default is the standard output of the computer that executes **cmdd**.

ibs=*n* Input block size in bytes--65,536 bytes by default. Some devices do not support block size greater than 65,535 bytes. See **bs**.

obs=*n* Output block size in bytes; 65,536 bytes by default. Some devices do not support block size greater than 65,535 bytes. See **bs**.

bs=*n* Set both input and output block size to *n* bytes, superseding **ibs** and **obs**. Also, if **bs** is specified, the copy is more efficient since no blocking conversion is necessary.

cbs=*n* Conversion buffer size in bytes. Use this option only if **ascii, unblock, ebcdic, ibm,** or /fBblock/fR conversion is specified. For **ascii** and **unblock,** *n* characters are placed into the buffer, any specified character mapping is done, trailing blanks are trimmed, and a newline is added before sending the line to the output. For **ebcdic, ibm,** or **block,** characters are read into the conversion buffer and blanks are added to make an output record of size *n* bytes.

skip=*n* Skip *n* input records before starting to copy.

files=*n* Copy *n* input files before terminating. This option is useful only when the input is a magnetic tape or similar device.

seek=*n* Seek *n* records from beginning of output file before copying.

count=*n* Copy only *n* input records.

conv=*arg* Perform specified conversion. *arg* is a comma-separated list of any of the following (see the Examples section):

`ascii`
Convert EBCDIC to ASCII.

`ibm` Slightly different map of ASCII to EBCDIC (see Restrictions).

`block`
Convert variable-length records to fixed length.

`unblock`
Convert fixed-length records to variable length.

`lcase`
Map alphabetics to lower case.

`ucase`
Map alphabetics to upper case.

`swap` Swap every pair of bytes.

`noerror`
Do not stop processing on an error.

`sync` Pad every input record to *ibs.*

`tomultidrop`
Convert a file so that it can be used on multidrop DataVault hardware.

`frommultidrop`
Convert a file from one that can be used on multidrop DataVault hardware.

The following options are used in IBM-format tape label processing:

label=[**isl** | **inl** | **ibl** | **osl** | **onl** | **obl**]

isl	Standard label processing on input tape.
inl	Non-labeled processing on input tape.
ibl	Bypass label processing on input tape.
osl	Standard label processing on output tape.
onl	Non-labeled processing on output tape.
obl	Bypass label processing on output tape.

vsn=*List of Volume Serial Numbers*

Specifies the volume serial numbers (VSNs) of the different volumes, separated by commas. On input tapes, only the volumes specified are processed. For output tapes, enough VSNs must be specified to accomodate all the data. Extra VSNs are ignored.

dsn=*List of dataset names*

Specifies the name(s) of the current dataset(s), separated by commas. Concatenating multiple datasets is supported only on input tapes.

rformat=*fmt*:*blen*:*rlen*

Specifies the record format and blocking format of the tape. This option is currently ignored on input standard-label tapes because format information is determined automatically from the labels on input tapes. The fields are:

fmt: (also see the Description section)

f	Fixed-length records
fb	Fixed-length blocked records
v	Variable-length records
vb	Variable-length blocked records
vbs	Variable-length, blocked, spanned records
u	Undefined

blen: (Maximum) block length

rlen: (Maximum) record length

WHERE EXECUTED

UNIX front end
VMEIO host computer
CM-IOP

DESCRIPTION

The command **cmdd** copies a specified input file to a specified output file with any requested conversions. The input and output block size may be specified to take advantage of raw physical I/O. After completion, **cmdd** reports the number of whole and partial input and output blocks.

Where sizes (*n*) are given for an option, the number may end with **k** for kilobytes (1024 bytes), **b** for blocks (512 bytes), or **w** for words (2 bytes). Also, two numbers can be separated by the character **x** to indicate a product.

Following are some of the more common reasons for converting a file:

- To convert unarchived data to a CMFS file. See the Examples section.
- To convert files moved via the Ethernet from a point-to-point DataVault to a multidrop DataVault, or vice versa. See the example in the Example section. (Note that generally it is not necessary to convert files residing on a DataVault at the time the Datavault is upgraded from point-to-point to multidrop. Serial files, however, are special cases; their conversion is explained below.)
- To convert serial files stored on a point-to-point DataVault so they can be used on a multidropped CM system. The conversion is done at the time of the hardware upgrade. To perform this conversion, use **cmdd** with the **conv=frommultidrop** option (yes, this is counterintuitive). This conversion is necessary both for serial files that have been transposed from parallel format and for serial data read directly to a DataVault using CMFS software that supports point-to-point hardware.

IBM-FORMAT TAPE-LABEL PROCESSING

cmdd has the capability to efficiently process IBM-format tapes. It supports Standard Label (SL), Non-Labeled (NL), and Bypass Label (BL) tape-processing modes. This man pages assumes familiarity with IBM tape-label formats and processing modes. For background information on tape labels and label processing, consult *MVS/370 Magnetic Tape Labels and File Structure Administration*, IBM manual #GC26-4064-2, Chapters 1, 2, and 5.

The IBM-format tape-label processing options are **label=**, **vsn=**, **dsn=**, and **rformat=**.

Note the following about variable-record format processing (specifically, **v**, **vb**, **fs**, and **vbs** of the *fmt* field of the **rformat** option):

On input variable-record format tapes, **cmdd** creates a control file (the filename is generated by the output filename and is named *output-filename*.ctl), which contains information about the record and block lengths of the input tape. The block descriptor words (BDW) and record/segment descriptor words (RDW/SDW) are NOT stripped from the data.

On output variable-record format tapes, **cmdd** checks for the existence of a corresponding control file. If the file exists, **cmdd** uses it to determine the size of each tape block to write. The data must either contain the correct BDW and RDW/SDW entries, or contain placeholders (zeros) at the correct locations, as specified by the control file. In the latter case, **cmdd** fills in the placeholders with the correct descriptor words extracted from the control file.

If no control file is found, **cmdd** peruses the data itself to determine the size of each tape block to write. In this case, the data must contain the correct descriptor words.

When processing variable-record output tapes, **cmdd** attempts to use fixed-block mode (see below) when it determines that all the block sizes in its buffer are the same size. In this instance, the buffer size specified on the command line does not need to be a multiple of the fixed block size. This use of fixed-block mode cannot be overridden.

Note the following about fixed-block mode processing (specifically, **f** and **fbs** of the *fmt* field of the **rformat** option):

If the tape drive is determined to be a StorageTek 4980, **cmdd** uses fixed-block mode as described below to greatly improve performance. To take full advantage of this, specify a large buffer size (usually on the order of several megabytes) to **cmdd**. Just as when specifying fixed-block mode explicitly on the command line, the buffer size specified must be a multiple of the value used for fixed-block mode, which is determined as follows:

For **isl** processing, **cmdd** automatically sets fixed-block mode to the size specified in the block length field specified in the **HDR2** label.

For **osl** or **onl** processing, **cmdd** automatically sets fixed-block mode to the block length specified in the **rformat** argument, except for variable-record format tapes (see below).

To prevent **cmdd** from automatically using fixed-block mode as described above, specify a fixed-block mode size of -1 on the command line (using the **ifbs** or **ofbs** arguments).

RESTRICTIONS

cmdd does not support the use of multiple media, nor does it support files that span more than one tape volume, except when handling IBM tapes.
Specifying conversion operations on IBM tapes is not supported.
The full- and partial-block counts reported by **cmdd** are not meaningful for IBM tapes.

EXAMPLE

The following example shows how to read an EBCDIC tape (**/dev/rmt0h**) on the front end into the ASCII file **x** in the CM file system. The tape is blocked in ten 80-byte EBCDIC card images per record. The resulting ASCII file has all lowercase characters. (**cmdd** is executed on the front end.)

```
% cmdd -todv if=/dev/rmt0h of=x ibs=800 cbs=80 conv=ascii,lcase
```

The following example writes a file (**final.data**) on the DataVault to a tape (**/dev/rmt0h**) on the front end. 64K bytes are transferred at a time and no conversions are performed.

(**cmdd** is executed on the front end.)

```
% cmdd -fromdv if=dv1:/final.data of=/dev/rmt0h
```

Note the use of raw magnetic tape. The **cmdd** command is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary record sizes.

The following example shows how to move files from a point-to-point DataVault to a multidrop DataVault. If a backup file (on tape) called **oldfile** was saved from a point-to-point DataVault and is to be restored as **newfile** on a multidrop DataVault, the file must first be written to the multidrop DataVault using the appropriate program (such as **cmtar**). Then issue the following command:

```
% cmdd -fromdv -todv if=oldfile of=/newfile conv=tomultidrop
```

To read a four-volume, IBM standard-labeled dataset with VBS format and maximum blocksize of 32760 bytes:

```
% cmdd if=/dev/rstc0 of=dv1:dataset -todv label=isl  \
dsn=TMC.IBM.DATASET vsn=811302,808911,823984,822922 bs=3276000
```

To write an IBM standard-labeled dataset spanning no more than four volumes, with fixed-size records of 80 bytes and tape blocksize of 8000 bytes:

```
% cmdd if=dv1:data of=/dev/rstc0 -fromdv label=osl \
dsn=TMC.IBM.FIXEDDATA vsn=812391,822661,824907,808961 \
rformat=f:8000:80 bs=800000
```

To read two datasets from IBM non-labeled tapes with 6400-byte blocks, concatenating them:

```
% cmdd if=/dev/rstc0 of=dv1:merged-data -todv label=inl \
vsn=111111,222222 files=2 ifbs=6400 bs=640000
```

To read two datasets from a single IBM standard-labeled tape volume, concatenating them:

```
% cmdd if=/dev/rstc0 of=dv1:merged-data -todv label=isl vsn=865148
dsn=IBM.DATA.1,IBM.DATA.2 bs=3276000
```

SEE ALSO

cmcp
copyfromdv
copytodv
dvcp
cmtar
cmdump

NAME

cmcountdown – Limit the amount of time a user/group may consume on any CM.

SYNOPSIS

cmcountdown [**-u** *user-name connect-time-limit*] [**-g** *group-name connect-time-limit*]

DESCRIPTION

cmcountdown, when run with the **-u** and **-g** switches, sets limits for the amount of connect time that a certain user or group may spend on any Connection Machine. When **cmcountdown** is executed without arguments, it prints a list of the current connect-time limits.

A process that runs under timesharing accumulates connect time only when the job is scheduled on the CM, not also when it is simply waiting to use the CM. Under exclusive mode, connect time is the same as the difference between detach and attach times.

Note that a group accures connect time from each member's CM activity; that is, the limit applies to the group as a whole, not to each member.

Connect time is never automatically "refreshed"; to allot more time to a user or group, **cmcountdown** must be re-executed. When **cmcountdown** is rerun for a user or group that has time remaining, the effect is that the new connect-time-limit replaces the old connect-time-limit, rather than adding the new time allotment to the old time allotment. (To increase the amount of allotted connect-time, **cmcountdown** must be run as Root. When a user (not root) runs **cmcountdown** with the **-u** switch, he or she can reduce the connect-time allotment but not increase it.)

As the system does not remember connect-time allotment and expenditure between system reboots, **cmcountdown** must be rerun at reboot time with the updated account information (extracted from the most recent set of accounting records: currently there are no tools to do so automatically).

ARGUMENTS

-u	Specify *connect-time-limit*, in seconds, for user *user-name*.
-g	Specify *connect-time-limit*, in seconds, for group *group-name*.

SEE ALSO

cmsa

NAME

cmverify – CM system verifier

SYNOPSIS

cmverify [**-iHv**] [**-n** *number*] [**-r** *number*]

ARGUMENTS

-i Ignore errors.

-H Run the check-hot-boards verifier.

-n *number-of-passes*
Run the verifier the specified number of times.

-r *number*
Run *number* router verifiers. *number* must be 0, 1, 2, 3, or 4.

-v Be verbose.

DESCRIPTION

cmverify is a user-level system verifier that provides an additional level of system verification beyond that provided by the CM diagnostics. Although it is useful for detecting hardware problems, **cmverify** generally cannot provide enough information to call out the specific component that is failing. CM diagnostics should always be used to provide detailed fault isolation before any hardware is moved or replaced. **cmverify** can run 7 basic test groups, listed below. By default, **cmverify** runs all of the test groups except **check-hot-boards.**

check-hot-boards
This test prints out information about all boards in the system that are signalling a thermal warning condition. A thermal warning condition is NOT NECESSARILY A PROBLEM: large CM's indicate thermal warnings in interior locations under normal conditions. A change in the thermal warnings exhibited by a given system can be useful in tracking down broken or stuck fans, however. System administrators are encouraged to keep a log of thermal warnings exhibited over time, to allow them to notice changes in the heat characteristics of the system. This test is disabled by default.

test-UC-scratch-ram
This test exercises IMP memory, using the same mechanisms used by PARIS and the Fortran runtime system to download and execute IMPS.

test-timer-trap

This test checks for a sequencer branch condition failure mode, which diagnostics often do not catch.

basic-routing
This test, which is run at various VP ratios and geometries, does some basic send operations with and without combinors. Results are checked and errors are reported if they are not as expected.

indirect-addressing
This test uses the **CM_aset32_shared_2L** instruction to verify correct operation of the indirect addressing hardware.

router-verifiers
Several levels of router verifiers can be run. Level 1 is the quickest, with level 4 taking several hours to complete. By default, the level 2 verifiers are run.

check-cm-memory
After running all tests, **cmverify** scans CM memory and reports any single bit ECC errors found. Double bit ECC errors are fatal to any application, and will be reported by the error system.

SAMPLE RUN

Sample run: % `cmverify -H`

```
********************************************************************************
Attached to {CM}* Timesharing on "Rosie" Sequencer 1
********************************************************************************
Running "cmverify" on CM "ROSIE", sequencer(s) 1, 1 pass
Test started Mon Feb  3 23:15:51 1992
2048 physical processors
Eunuch and Boxer chips
CM physical memory limit = 254464
FPU = WTL3164
Calibrating CM timer...Done. CM speed = 7.00 MHz
Pass 1, (0 errors): check-hot-boards ...
Pass 1, (0 errors): Testing UC scratch ram...
Pass 1, (0 errors): test-timer-trap...
Pass 1, (0 errors): basic routing; VP ratio 1, (4, 512)
Pass 1, (0 errors): basic routing; VP ratio 2, (4096, 1)
Pass 1, (0 errors): basic routing; VP ratio 4, (16, 512)
Pass 1, (0 errors): basic routing; VP ratio 8, (4, 4096)
Pass 1, (0 errors): basic routing; VP ratio 16, (512, 64)
Pass 1, (0 errors): basic routing; VP ratio 32, (1, 65536)
Pass 1, (0 errors): basic routing; VP ratio 64, (2, 65536)
Pass 1, (0 errors): basic routing; VP ratio 128, (1, 262144)
Pass 1, (0 errors): basic routing; VP ratio 256, (524288, 1)
Pass 1, (0 errors): indirect-addressing ...
Pass 1, (0 errors): router-verifiers (level 2) ...
Pass 1, (0 errors): check-cm-memory ...
```

```
Test completed, 0 errors
147.9 user 5.5 system 4:00 (63%)
0+5576k (414 max) 85+6io 153+1084pf 0sw
```

SEE ALSO

vbutil
bbutil
new-test-nexus
test-uc
hardware-test-complete
dvtest2 (a comparable I/O-system verifier)

NAME

unfsd, unfsmntd – user-level NFS daemons

SYNOPSIS

/usr/local/etc/unfsd [**-f** *exports-file*] [**-p**] [**-o** *option-string*]

/usr/local/etc/unfsmntd

DESCRIPTION

unfsd starts a daemon that handles client filesystem requests. Unlike **nfsd(8)**, **unfsd** operates as a normal user-level process and can be run on standard 4.3BSD systems.

unfsmntd starts an ancillary user-level mount daemon.

Options: The **-f** option specifies the exports file, listing the clients that this server is prepared to serve and parameters to apply to each such mount (see **unfsd_exports(5)**). By default exports are read from **/usr/local/etc/unfsd_exports** **-p** option puts the server into promiscuous mode where it will serve any host on the network. The **-o** option specifies default mount parameters in the same format as those appearing in the **unfsd_exports** file.

SEE ALSO

unfsd_exports(5)
ugidd(8C)

BUGS

Does not understand netgroups.

NAME

unfsd_exports – NFS file systems being exported by user-level NFS server

SYNOPSIS

/usr/local/etc/unfsd_exports

DESCRIPTION

The file **/usr/local/etc/unfsd_exports** describes the file systems which are being exported to nfs clients. It is processed by the user-level NFS daemon **unfsd**(8C) when the daemon is started.

The file format is similar to that of the SunOS **exports** file. The file is organized by lines. A # introduces a comment to the end of the line; a \e preceding a new line disables the line break, making the entry of long input lines more convenient. Each line consists of a mount point and list of machine names allowed to remote mount the server's file hierarchy at that mount point. A machine name is optionally followed by a list of mount parameters enclosed in parentheses. These are the parameters that are currently recognized.

`secure *`	Reject requests that originate on an internet port ≥ IPPORT_RESERVED.
`insecure`	Accept requests originating on any port.
`root_squash`	Map requests from uid 0 on the client to uid -2 on the server.
`no_root_squash *`	Don't map requests from uid 0.
`ro *`	Mount file hierarchy read-only.
`rw`	Mount file hierarchy read-write.
`link_relative *`	Convert symbolic links starting with a slash into relative links by prepending the necessary number of ../'s to get from the link directory to the file hierarchy root on the server.
`link_absolute`	Leave symbolic links starting with a slash as they are.
`map_identity *`	Assume the client and server share the same uid/gid space.
`map_daemon`	Map local and remote names and numeric ids using a lname/uid map daemon on the client from which the NFS request originated, to map between the client and server uid spaces (see **ugidd**(8)).

(* indicates defaults.)

EXAMPLE

```
/     snail whelk(map_identity)
tusk(root_squash, map_daemon, ro)
/usr  usage(root_squash, map_daemon, ro)
```

FILES

/usr/local/etc/unfsd_exports
Sample **unfsd_exports**:

```
/ fe1(insecure, no_root_squash, map_identity, rw)
/ fe2(insecure, no_root_squash, map_identity, rw)
/ fe3(insecure, no_root_squash, map_identity, rw)
```

SEE ALSO

mountd(8C)
unfsd(8C)
ugidd(8C)

BUGS

The mount point at the start of each line is currently ignored. Authorized clients may mount at any point in the server's hierarchy.