The
Connection Machine
System

CM-5 I/O System Programming Guide

Preliminary Documentation for Version 7.2 Beta 1

December 21, 1992

Thinking Machines Corporation
Cambridge, Massachusetts

First beta printing, December 21, 1992

****

PRELIMINARY DOCUMENTATION

The information in this document is subject to change without notice and should not be construed as a commitment by Thinking Machines Corporation. Thinking Machines assumes no liability for errors in this document.

This document does not describe any product that is currently available from Thinking Machines Corporation, and Thinking Machines does not commit to implement the contents of this document in any product.

****

The pages describing **cmtar** in this manual are derived from *tar: The GNU Tape Archive*, by Jay Fenlason. As such, these pages, and no others, carry the following copyright:

Thinking Machines Corporation
245 First Street
Cambridge, Massachusetts 02142-1264
(617) 234-1000

Contents

About This Manual

Objectives of This Manual

This manual describes the Connection Machine file systems available with CMOST Version 7.2 Beta 1.

Intended Audience

We assume the reader is familiar with basic Connection Machine model CM-5 operation and terminology. We assume the reader is familiar with the UNIX file system, or that documentation about it is available to him or her.

Revision Information

This is a preliminary draft of a new manual. We intend to expand the information herein.

The manual *Connection Machine I/O System Programming, Version 6.1*, may provide helpful information in the interim.

Field Test Support

Field test software users are encouraged to communicate with Thinking Machines Corporation as fully as possible throughout the test period. Please report any errors you may find in this software and suggest ways to improve it.

When reporting an error, please provide as much information as possible to help us identify the problem. A code example that failed to execute, a session transcript, the record of a backtrace, or other such information is extremely helpful in this regard.

If your site has an applications engineer or a local site coordinator, please contact that person directly for field test support. Otherwise, please contact Thinking Machines' home office customer support staff:

Internet	
Electronic Mail:	`customer-support@think.com`
uucp	
Electronic Mail:	`ames!think!customer-support`
U.S. Mail:	Thinking Machines Corporation Customer Support 245 First Street Cambridge, Massachusetts 02142-1264
Telephone:	(617) 234-4000

Chapter 1

Introduction to CM-5 I/O

The Connection Machine I/O system facilitates the storage, retrieval, and interprocess sharing of the massive amounts of data used and generated by the Connection Machine supercomputers. The high speeds at which data transfer typically occurs significantly enhances CM application performance. The Connection Machine I/O system is both flexible and easy to use; it can incorporate a wide variety of I/O devices and access any of these devices through a variety of interfaces.

This volume describes how to use the I/O system of the Connection Machine model CM-5 to achieve typical data transfer objectives. This chapter provides background information about the CM-5 I/O system, including descriptions of:

- The hardware components of the CM-5 I/O system.
- The software that manipulates the CM-5 I/O system and its files.

1.1 I/O Components of the CM-5 System

Figure 1 and Figure 2 show the I/O hardware components of a typical CM-5 system. In addition to the CM-5 itself, the system consists of one or more I/O devices, listed below. As shown in Figure 2, a Connection Machine model CM-2 or CM-200 may also have access to the I/O devices. Your system administrator can tell you which devices are available for general use.

As of CMOST Version 7.2, the CM-5 system supports the following I/O devices:

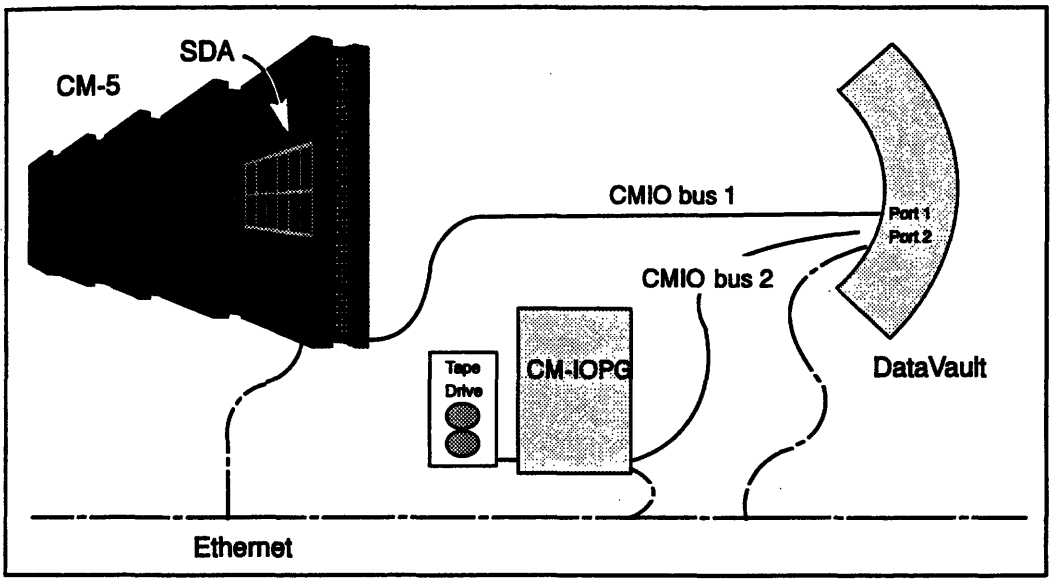

Figure 1. A typical CM I/O system for the CM-5.

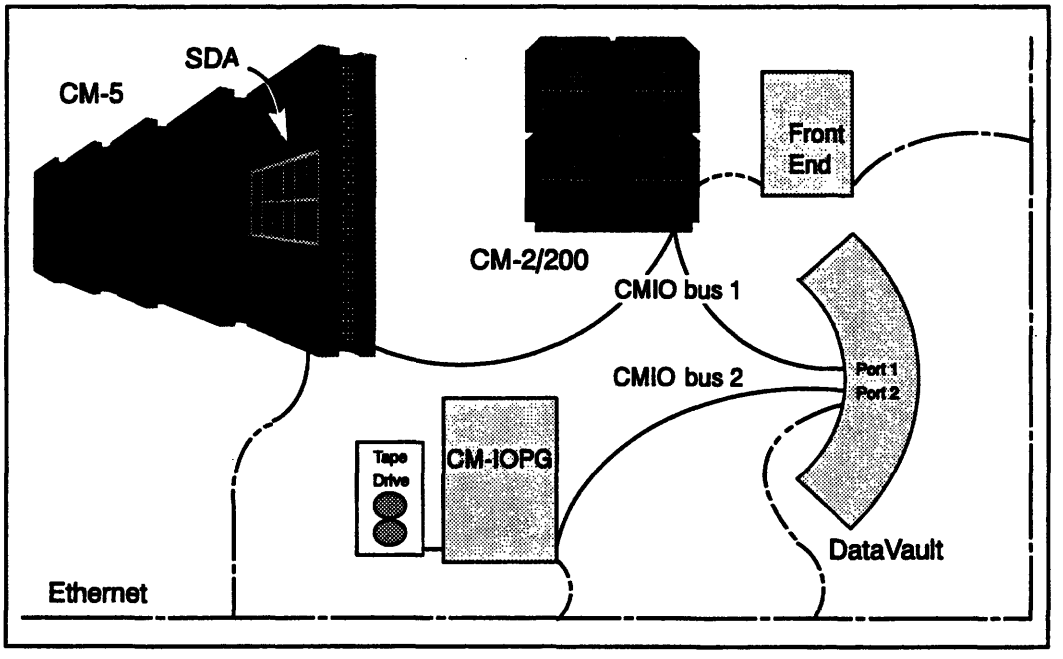

Figure 2. A typical CM I/O system shared by the CM-2/200 and the CM-5.

- Scalable Disk Array (SDA)

  The *Scalable Disk Array* is the CM-5 system's main storage device, typically providing 25-200 Gbytes of disk storage at I/O bandwidths of 33-264 Mbytes/second. The SDA, which resides within the CM-5 cabinets, is a collection of sets of high-speed commodity disks directly connected to the CM-5 networks. The direct connection to the networks enables each set of disks to contribute not only to storage capacity but also to I/O performance; the number of the disk sets in an SDA system can be increased or decreased to achieve an I/O system matched to the performance and capacity needs of CM-5 applications.

  An SDA supports up to two separate UNIX-compatible file systems. Data is striped over all the disks in a file system, transparently to the user. Parity is checked (and single-bit errors are corrected, when necessary) on each I/O operation.

The devices listed below reside on a high-speed multidrop CMIO bus, which has a special interface to the CM-5 networks:

- DataVault

  The *DataVault* is compatible with all the Connection Machine systems: the CM-5, CM-2, and CM-200. It provides up to 60 Gbytes of disk storage. The DataVault is dual-ported to allow high bandwidth data transfer and system flexibility. The *DataVault file server computer*, a microcomputer inside the DataVault, hosts the DataVault's UNIX-like file system.

- VMEIO Host Computer

  The *VMEIO host computer* typically connects VME-based I/O devices, such as video frame grabbers and magnetic tape drives, to the Connection Machine systems. Such devices provide convenient means of backing up and retrieving data. The file system that resides on the VMEIO host computer treats each device as a file, and allows the host computer itself to provide secondary disk storage for files.

- CM-IOPG

  The *CM-IOPG* also supports the VME protocol. Typically, though, the CM-IOPG connects storage devices that have SCSI drives — such as an IBM-3480-compatible tape drive — to the Connection Machine systems. As on the VMEIO host computer, the file system that resides on the CM-IOPG treats each device as a file, and allows the host computer itself to provide secondary disk storage for files.

To communicate with the CM-5 and each other, the I/O peripherals use internal networks or a combination of the networks and external buses. The SDA has a direct interface to the CM-5 processors via the CM-5's Control Network and Data Network. The other peripheral devices — DataVault, VMEIO host computer, and CM-IOPG — use Thinking Machines Corporation's proprietary CMIO bus to convey data among themselves, and to and from the PNs (processing nodes); an Ethernet conveys control and status information.

An Ethernet also can convey data among the DataVault, VMEIO host computer, and CM-IOPG in the absence of an appropriate CMIO bus connection. In Figure 1 and Figure 2, for example, since there is no CMIO bus connection between either CM and the CM-IOPG, CM-2/CM-5 ↔ CM-IOPG data transfer would proceed over the Ethernet. Since the Ethernet is very slow compared to a CMIO bus, if a need were to arise for frequent CM-2/CM-5 ↔ CM-IOPG data transfers in the systems shown in these figures, an additional CMIO bus would be installed between the CM(s) and the CM-IOPG.

1.2 I/O Processes on the CM-5 System

The Connection Machine I/O system is modeled on the typical client-server relationship: a program running on a machine acting as client requests that a machine running a server process either send or receive data. The following CM-5 system components can act as a client:

- CM-5 processing node(s)[1]
- CM-5 control processor memory
- CM-IOPG
- VMEIO host computer

The following Connection Machine system components can act as a server:

- CM-5 control processor memory
- SDA
- DataVault
- CM-IOPG
- VMEIO host computer

1. That is, operating together as one body, or separately; see the *CMMD User's Guide*.

The three kinds of I/O currently supported by the CM-5 system are distinguished by which system component acts as client and which acts as server:

- *Parallel I/O*

  The CM-5 — specifically, one or more processing nodes — is the client, requesting a server to read to it or accept writes from it.

- *Serial I/O*

  A CM-5 partition manager or other control processor, or a serial computer (for example, a CM-IOPG or VMEIO host computer) is the client, requesting a server to read to it or accept writes from it.

- *Tape I/O*

  A CM-5 partition manager or other control processor, or a serial computer is the client, requesting a tape device acting as server to read to it or accept writes from it.

1.3 The Connection Machine File Systems

The CM-5 system can access three file systems, two of which are proprietary to Thinking Machines Corporation. The two proprietary file systems, the *SFS* and *CMFS* file systems, exploit the great speed and massive storage capabilities of the Connection Machine I/O systems. The other supported file system, *UNIX*, further enhances system usability. All three file systems organize files into directories, use pathnames to identify them, and treat all I/O devices as files. Following is a brief description of each file system:

- SFS (Scalable File System) is a UNIX-compatible file system mounted on the SDA's I/O control processor, which manages the SDA's files. The SFS file system is an enhancement of the UNIX file system, with extensions to support parallel I/O and files much larger than most UNIX implementations can accommodate. From a user's perspective, the SFS file system appears like and behaves like a UNIX file system.

  A CM-5 program can access the SFS file system via the CMMD library, the CMFS library and a subset of the CMFS commands, standard UNIX routines and commands, and the CM Fortran Utility Library; see Section 1.4.

- CMFS (CM File System) is a UNIX-like file system that can reside on the CMIO-bus data-storage devices, including the DataVault, CM-IOPG, and VMEIO host computer. Like the SFS file system, the CMFS file system has extensions to support parallel I/O and very large files. A CM-5 program can access the CMFS file system via the CMFS library and CMFS commands, the CMMD library, the CM Fortran Utility Library, and, if the CMFS library is NFS-mounted, standard UNIX routines and commands[2]; see Section 1.4.

  From a user's perspective, although the CMFS file system is similar to the UNIX file system, there are some differences in its appearance and behavior. For example, there are special environment variables that apply only to the CMFS file system. Chapter 2 of this manual is an introduction to the CMFS file system and the CMFS library and commands. In the future, this manual will be expanded to provide more detailed information about using the CMFS library on the CM-5 system. (In the meantime, the manual *Connection Machine I/O System Programming Guide, Version 6.1*, which provides information about using CMFS commands and calls on the CM-2 and CM-200 systems, may be helpful to you.)

- The standard implementation of the UNIX file system can reside on all CM-5 control processors and on all other serial computers in the Connection Machine systems. Note that this manual assumes you are familiar with the UNIX file system, or that documentation that describes it is available to you.

The UNIX file system should hold the executable programs that run on the CM-5, as well as CM-5 system software and (possibly) users' private files, but the SFS and CMFS file systems should store data files only.

From a user's perspective, the CMFS file system is completely separate from the SFS and UNIX file systems; the CMFS file system has a separate directory tree, its own current working directory, and its own environment variables, and only if it is NFS-mounted can the CMFS file system recognize UNIX. The SFS file system, however, shares name space with the UNIX file system on the CP on which it is running; as such, the UNIX environment variables (`PATH`, for example) and UNIX commands (`chdir`, for example), are recognized by the SFS file system.

2. Although NFS enhances usability of the CMFS file system, this release of NFS is preliminary, and UNIX commands and calls may not afford performance as good as that of the other interfaces.

1.3.1 Which File System Do You Want?

In a CM-5 system that has access to more than one of the file systems described in Section 1.3, indicate the file system you want to use by setting the CM-5 environment variable **CMFS_PATHTYPE** in your shell or in your **.cshrc** file:

```
% setenv CMFS_PATHTYPE unix | cmfs | mixed
```

unix Any pathname refers to the local UNIX or UNIX- compatible file system — the SDA, if your system contains one.

cmfs Any pathname refers to a CMFS file system — a DataVault, VMEIO host computer, or CM-IOPG. If your CM-5 system has access to more than one of these devices, set the CMFS environment variables **DVHOSTNAME** and/or **DVWD**. If those environment variables are not set, the default CMFS file system, listed with the kernel, is used.

mixed Any pathname that does not contain a colon (**:**) refers to the SDA. A pathname that does contain a colon refers to a CMFS file system:

- If there is a CMFS-hostname component included before the colon, that device is used.
- If there is no CMFS-hostname component, the device listed with **DVHOSTNAME** and/or **DVWD** is used. If those environment variables are not set, the default CMFS file system, listed with the kernel, is used.

(For file system defaulting behavior when **CMFS_PATHTYPE** is not set, please see the man page **CMFS_PATHTYPE(7)**.)

For example, in the CM-5 system shown in Figure 3, suppose the following:

- You set **CMFS_PATHTYPE** to **mixed**.
- You set **DVHOSTNAME** to **dv1**.
- You set **DVWD** to **big_project**.

If your program calls a routine to open the file **my_data**, the file is opened in your current working directory on the device **sda1**. If your program calls a routine to open the file **:my_data**, the file is opened in **dv1:/big_project**. If

your program calls a routine to open the file `iop1:my_data`, the file is opened in `iop1:/big_project`.

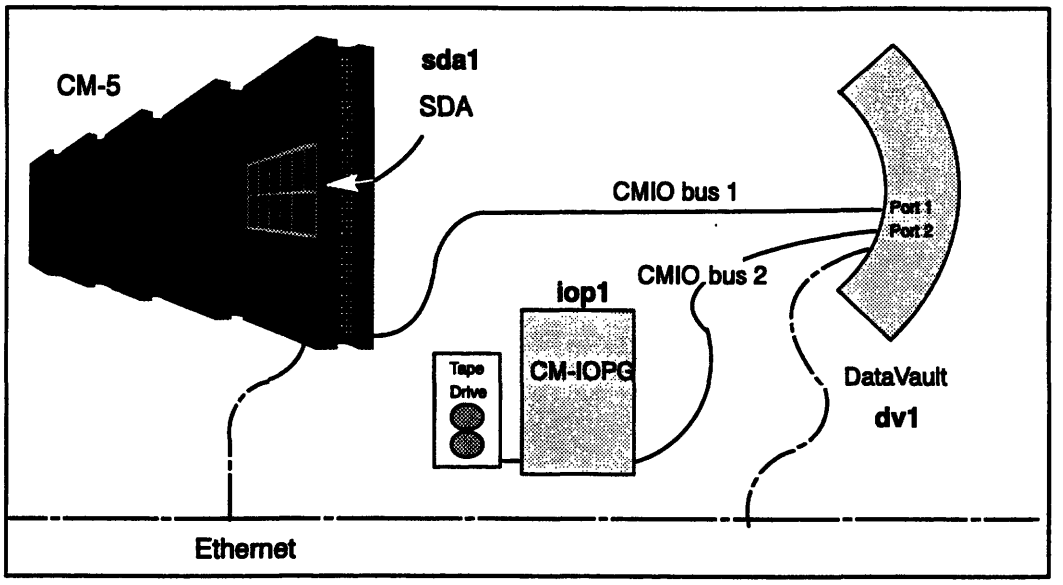

Figure 3. A typical CM I/O system for the CM-5.

1.4 File System Interfaces

The CM programming languages have a variety of interfaces to the Connection Machine file systems; these are described in the following subsections:

- For information about CM-5 I/O using C, see Section 1.5.1.
- For information about CM-5 I/O using C*, see Section 1.5.2.
- For information about CM-5 I/O using Fortran 77, see Section 1.6.1.
- For information about CM-5 I/O using CM Fortran, see Sections 1.6.2 and 1.6.3.

File Padding

Padding — bytes consisting of undefined data — is automatically placed into files to enhance performance. In most cases[3], the padding is handled transparently whenever the file is accessed.

The following padding occurs automatically:

- Parallel write operations to the SFS file system (that is, from the PNs to the SDA) pad the end of the file to the next 16-byte boundary.
- All write operations to the CMFS file system pad the end of the file to the next 512-byte boundary.

1.5 Accessing the File Systems from C-Based Languages

The information in subsections 1.5.1 and 1.5.2 is summarized in Table 1.

1.5.1 Accessing the File Systems from C

To access any Connection Machine file system from a C program running on the CM-5, use either the CMMD library, the CMFS library, or the UNIX routines:

- When performing parallel I/O, the program must use the CMMD library.
- When performing serial I/O, the program can use the CMFS library or the UNIX routines. Note that UNIX routines can access a CMFS file system only if it is NFS-mounted.

3. Certain circumstances may require the user program to handle the padding explicitly: reading a file into an array(s) with characteristics different from the one(s) used to write the file, sharing a file between the CM-5 and CM-2 or CM-200 systems, or reading or writing a file via a language or library different from that used to write or read it.

Table 1. The C and C* interfaces to the Connection Machine file systems.

Programming Language	Mechanism	File System(s)	I/O Process(s)[4]
C	CMMD library	SFS, CMFS, and UNIX	Parallel
C	CMFS library	SFS, CMFS, and UNIX	Serial
C	UNIX routines	SFS, NFS-mounted CMFS, and UNIX	Serial
C*	CMFS library	SFS and CMFS	Parallel and serial
C*	UNIX routines	SFS, NFS-mounted CMFS, and UNIX	Serial

1.5.2 Accessing the File Systems from C*

Reminder

C*'s array dimensions are numbered left-to-right from 0. The leftmost axis — `X1` in array `A(X1, X2, X3)` — varies fastest. This arrangement is called *column-major order* or *axis covariant order.*

CM-5 C* programs can access the SFS, CMFS, and UNIX file systems. C* programs that use the CMFS library must include `<cm/cmfs.h>` and link with `-lcmfs_cs` and `-lcmfs_cm5`.

4. Recall that parallel I/O = file ↔ PNs, and serial I/O = file ↔ serial-computer-memory (which includes file maintenance operations, such as `CMFS_chmod`).

From C*, to access the SDA's SFS file system or a CMFS file system, use either the CMFS library or the UNIX routines:

- When performing serial I/O, the program can use either the CMFS library serial I/O routines or the UNIX I/O routines. (To use UNIX routines to access a CMFS file system, it must be NFS-mounted.)
- When performing parallel I/O, the program must use the CMFS library's synchronous[5] I/O routines (**`CMFS_read_file_always`** and **`CMFS_write_file_always`**). Because the C* versions of these routines are overloaded, they require C*-specific declarations:

```
overload CMFS_read_file_always;
overload CMFS_write_file_always;

int CMFS_read_file_always (int, void:void *, int);
int CMFS_write_file_always (int, void:void *, int);
```

  For complete descriptions of these routines, see their man pages (**`CMFS_read_file_always(3)`** and **`CMFS_write_file_always(3)`**) in Appendix B of this manual.

  Note that these routines are available with C* Version 7.0 *with the Beta 1 patch*, and with C* Version 7.1. For additional information about C* I/O, refer to Thinking Machines Corporation's C* documentation, including release notes.

To access the standard implementation of a UNIX file system via serial I/O only, use the UNIX routines and commands.

1.6 Accessing the File Systems from Fortran-Based Languages

The information in subsections 1.6.1, 1.6.2, and 1.6.3 is summarized in Table 2, Table 3, and Table 4.

---

5. Currently, the CMFS library's buffered I/O and streaming I/O routines are not supported on the CM-5.

1.6.1 Accessing the File Systems from Fortran 77

To access the SFS file system or a CMFS file system from a Fortran 77 program running on the CM-5, use the CMMD library, the CMFS library, or the UNIX routines:

- When performing parallel I/O, the program must use the CMMD library.
- When performing serial I/O, the program must use either the CMFS library's serial routines or the UNIX routines. (To use UNIX routines to access a CMFS file system, it must be NFS-mounted.)

To access a standard implementation of a UNIX file system via serial I/O only, we recommend using UNIX routines and commands

Table 2. The Fortran 77 interfaces to the Connection Machine file systems.

Programming Language	Mechanism	File System(s)	I/O Process(s)[6]
Fortran 77	CMMD library	SFS and CMFS	Parallel
Fortran 77	CMFS library	SFS, CMFS, and UNIX	Serial
Fortran 77	UNIX routines	SFS and NFS-mounted CMFS, and UNIX	Serial

1.6.2 Accessing the File Systems from CM Fortran Version 1.2

Version 1.2 of CM Fortran does not support access to an SFS file system.

To access the CMFS file system from a CM Fortran Version 1.2 program running on the CM-5, use the CM Fortran Utility Library, the CMFS library, or UNIX routines:

- When performing parallel I/O, the program must use the CM Fortran Utility Library.

6. Recall that parallel I/O = file ↔ PNs, and serial I/O = file ↔ serial-computer-memory (which includes file maintenance operations, such as `CMFS_chmod`).

- When performing serial I/O, the program can use the CM Fortran Utility Library, the CMFS library, and, if the file system is NFS-mounted, UNIX routines.

To access a standard implementation of a UNIX file system via serial I/O only, use the CM Fortran Utility Library, the language's I/O statements, the CMFS library, or UNIX routines.

Table 3. CM Fortran (V. 1.2) interfaces to the Connection Machine file systems.

Programming Language	Mechanism	File System(s)	I/O Process(s)[7]
CM Fortran	CMF Utility Library	CMFS	Parallel and serial
CM Fortran	CMF Utility Library	UNIX	Serial
CM Fortran	I/O statements	UNIX	Serial
CM Fortran	CMFS library	CMFS and UNIX	Serial
CM Fortran	UNIX routines	NFS-mounted CMFS and UNIX	Serial

1.6.3 Accessing the File Systems from CM Fortran Version 2.1

Note that as of this writing, CM Fortran 2.1 is in its first Beta release (Beta 0). Release notes issued with subsequent versions of CMF 2.1 may provide information that updates this manual's information.

To access the SFS or CMFS file system from a CM Fortran Version 2.1 program running on the CM-5, use the CM Fortran Utility Library, the CMFS library, or UNIX routines:

- When performing parallel I/O, the program must use either the CM Fortran Utility Library or the CMFS library's synchronous I/O routines, `CMFS_READ_FILE_ALWAYS` and `CMFS_WRITE_FILE_ALWAYS`. Manual pages for these routines are provided in this manual's Appendix B.

7. Recall that parallel I/O = file ↔ PNs, and serial I/O = file ↔ serial-computer-memory (which includes file maintenance operations, such as `CMFS_chmod`).

- When performing serial I/O, the program can use the CM Fortran Utility Library, the CMFS library, and, if the file system is NFS-mounted, UNIX routines.

To access a standard implementation of a UNIX file system via serial I/O only, use the CM Fortran Utility Library, the language's I/O statements, the CMFS library, or UNIX routines.

Table 4. CM Fortran (V. 2.1) interfaces to the Connection Machine file systems.

Programming Language	Mechanism	File System(s)	I/O Process(s)[8]
CM Fortran	CMF Utility Library	SFS and CMFS	Parallel and serial
CM Fortran	CMF Utility Library	UNIX	Serial
CM Fortran	I/O statements	UNIX	Serial
CM Fortran (support projected but not currently available)	*I/O statements*	*SFS and CMFS*	*Parallel and serial*
CM Fortran	CMFS library	SFS and CMFS	Parallel and serial
CM Fortran	CMFS library	UNIX	Serial
CM Fortran	UNIX routines	SFS, NFS-mounted CMFS and UNIX	Serial

8. Recall that parallel I/O = file ↔ PNs, and serial I/O = file ↔ serial-computer-memory (which includes file maintenance operations, such as `CMFS_chmod`).

Chapter 2

Introduction to the CMFS File System and Its Software

The CMFS file system is a UNIX-like file system that can reside on the CMIO-bus data-storage devices, including the DataVault, CM-IOPG, and VMEIO host computer. The CMFS file system software — a set of utilities and a library of routines — operates on not only the CMFS file system, but also on the SFS file system[9]. This chapter provides basic information about using the CMFS file system and its software from a CM-5 system.

2.1 The CMFS File System Environment

The CMFS file system exploits the great speed and massive storage capabilities of the CM-5 I/O system. The CMFS file system is also easy to use, especially if you are familiar with the standard UNIX file system. The two file systems are similar in several ways: they organize files into directories, use pathnames to identify them, and treat I/O devices as files. Unlike a UNIX file system, however, the CMFS file system does not have a single root directory. Each CMIO-bus device can have a CMFS file system directory tree with its own root directory, as Figure 4 illustrates.

As described in Chapter 1, the CM-5 environment variable **`CMFS_PATHTYPE`** defines the heuristic that the CM-5 uses to determine which file system to attempt to access, given a pathname: to use the CMFS file system, set **`CMFS_PATHTYPE`** to **`cmfs`** or **`mixed`**. The specific CMFS file system host and the directory within

9. As of CMOST 7.2 Beta 1, a subset only of the CMFS commands can operate on the SFS file system. In a future release, the full set of CMFS commands will operate on the SFS file system.

it are then defined by either a full pathname or a combination of a relative pathname and environment variables:

- The CMFS file system identifies each data-storage device by its unique hostname — the name of the machine followed by a colon(:). A file's full pathname, therefore, includes a hostname component — for example, **dv1:/project/data**. (Because the hostname component contains a colon character, the name of the file itself cannot contain a colon.) If you specify a full pathname, therefore, the CM-5 system requires no further information in order to attempt to access the file.

- The CMFS environment variables, **DVHOSTNAME** and **DVWD**, define the default CMFS file system host and default working directory, respectively. When you do not give a full pathname, these environment variables are consulted.

  Set **DVHOSTNAME** from a C shell as follows:

  ```
  % setenv DVHOSTNAME default_hostname
  ```

  Set **DVHOSTNAME** from a Bourne shell as follows:

  ```
  $ DVHOSTNAME = default_hostname
  $ export DVHOSTNAME
  ```

  To specify your default current working directory in the CMFS file system, set **DVWD** from a C shell:

  ```
  % setenv DVWD new_working_directory
  ```

  or from a Bourne shell:

  ```
  $ DVWD = new_working_directory
  $ export DVWD
  ```

  new_working_directory may or may not include a hostname component. If it does not, *new_working_directory* is relative to the default CMFS host set by **DVHOSTNAME**. For example, if you set the environment variables as follows:

  ```
  % setenv DVHOSTNAME dv1
  % setenv DVWD /project
  ```

  and later refer to the file **data**, the system will interpret the full name of the file as **dv1:/project/data**. If you want instead to use the file **dv1:/tests/data**, either explicitly specify the file's full pathname or reset **DVWD** to **/tests**. If you want to use the file **dv2:/tests/data**,

either explicitly specify the file's full pathname or reset **DVHOSTNAME** to **dv2**.

If you do not set **DVHOSTNAME**, and if **DVWD**'s *new_working_directory* does not include a hostname component, the default CMFS hostname is the one listed with the CM-5 system kernel (see your system administrator). If none is listed with the kernel, the file **/usr/local/etc/dv_hostname** is consulted. If that file is missing, the default CMFS file system is the one on the local host.

For convenience, you can have your **.login** file or your **.cshrc** file set **DVWD** and/or **DVHOSTNAME** (as well as **CMFS_PATHTYPE**) each time you open a shell to work in the CM-5 programming environment.

2.2 CMFS Files

A *CMFS file* is one stored within the CMFS file system, regardless of which computer model generated its data and format:

- A *serial CMFS file* is formatted in the conventional manner, as by serial computers, consisting of only one stream of data. A file created on the CM-5 is serial in format to allow it to be easily shared between Connection Machine models as well as serial computers. A high-performance transposition mechanism automatically "parallelizes" the data when it is read into the CM-5, and "serializes" it when it is written out to the file system.

- *Parallel CMFS files* are indigenous to the Connection Machine model CM-2 and CM-200. Consisting of many streams of data, one stream per CM-2/200 processor, they are specially formatted to take advantage of the machine's massive parallelism. Although files created on the CM-2 and CM-200 are parallel by default, you can choose to write data to them in serial format, or even to bypass parallel formatting altogether by creating the file in serial format; these options are discussed in the *Connection Machine I/O System Programming Guide, Version 6.1*.

It is easy to share files among a CM-2 or CM-200 and a CM-5 or a serial computer by using a CM-2 (and CM-200) transposition routine. Again, see the *Connection Machine I/O System Programming Guide, Version 6.1*.

2.3 The CMFS Software

The CMFS file system software for the CM-5 includes a set of user commands and a library of functions for use by programs written in C, C*, Fortran, or CM Fortran. Generally, you can use CMFS commands and library routines from any client installed with CMFS software, although a few commands and routines require specific hardware.

2.3.1 The CMFS User Commands

The CM-5 CMFS user commands, issued from within a shell, perform typical file/directory manipulation tasks, such as copying, moving, and deleting files, and building, deleting, and listing the contents of a directory. Table 5 lists all the CMFS commands supported on the CM-5. Man pages for the CMFS commands are in Appendix A of this manual.

As the table shows, many of the CMFS commands are simply UNIX commands with a "cm" prefix. In general, a CMFS command performs the same function as its corresponding UNIX command.

The CMFS commands can operate on a file within a CMFS file system or an SFS file system[10], depending on how the `CMFS_PATHTYPE` environment variable is set. For example, in the CM-5 system in shown in Figure 4, suppose the following:

- You set `CMFS_PATHTYPE` to `mixed`.
- You set `DVHOSTNAME` to `dv1`.
- You set `DVWD` to `big_project`.

If you execute `ibmtape` on the file `my_data`, the file acted upon is the one called `my_data` in your current working directory on the device `sda1`. If you execute `ibmtape` on the file `:my_data`, the file acted upon is `dv1:/big_project/my_data`. If you execute `ibmtape` on the file `vme:/my_data`, the file acted upon is `vme:/my_data`.

10. As of CMOST 7.2 Beta 1, a subset only of the CMFS commands can operate on the SFS file system. In a future release, the full set of CMFS commands will operate on the SFS file system.

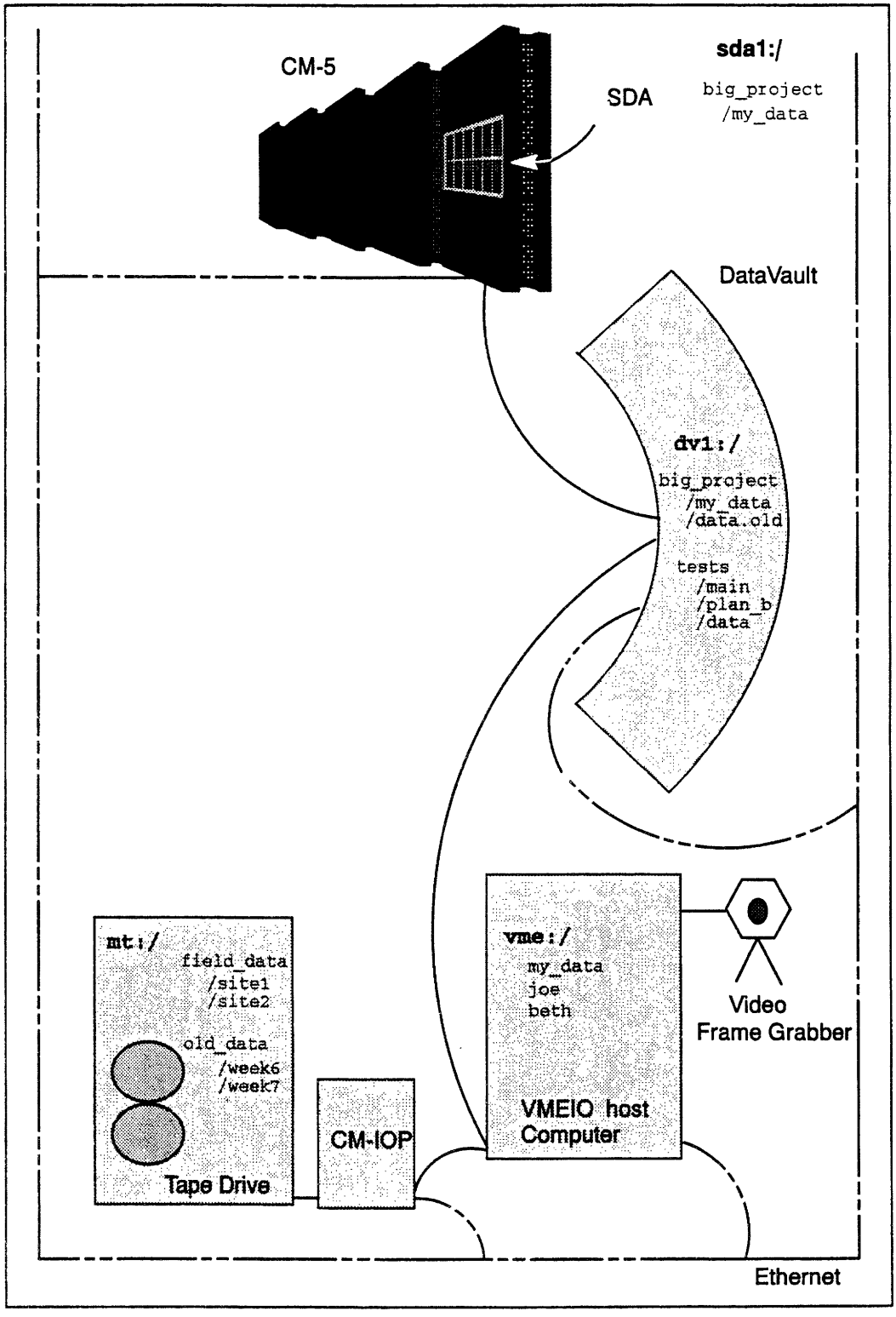

Figure 4. Files within a typical CMFS file system.

Table 5. TheCM-5 CMFS user commands.

Command	Purpose
`cmchgrp`	Changes the group ownership of a CMFS file.
`cmchmod`	Changes the permissions mode of a CMFS file.
`cmchown`	Changes the owner of a CMFS file.
`cmcp`	Copies a file to another location within that CMFS file system or to another CMFS file system.
`cmdd`	Copies raw data on tape into a CMFS file.
`cmdf`	Displays free and used disk space for a CMFS file system.
`cmdu`	Summarizes disk usage for a CMFS file system.
`cmdump`	Archives CMFS files or SFS files.
`cmfind`	Finds CMFS files.
`cmglob`	Prints the CMFS files that match the specified pattern(s), which enables wildcarding.
`cmln`	Makes links to CMFS files or directories.
`cmls`	Lists a CMFS file system directory's contents.
`cmmkdir`	Makes a CMFS file system directory.
`cmmknod`	Makes a CM character-special file.
`cmmv`	Moves (renames) a CMFS file or directory.
`cmptruncate`	Truncates or extends a CMFS file.
`cmrestore`	Extracts files from an archive made by `cmdump`.
`cmrm`	Removes (unlinks) a CMFS file or directory.
`cmrmdir`	Removes (unlinks) an empty CMFS file system directory.
`cmstat`	Prints information about a CMFS file, including its mode, size, and time of last access.
`cmtar`	Creates or extracts a tape archive using either a CMFS file system or an SFS file system.
`cmtruncate`	Truncates or extends a CMFS file.
`copyfromdv`	Copies a CMFS file system file into an SFS file system (or other UNIX file system).
`copytodv`	Copies an SFS file system file (or other UNIX-system file) to a CMFS file system.
`ibmtape`	Import/export IBM data sets to/from a CMFS file system or SFS file system.

2.3.2 The CMFS Library Routines

The CM-5 CMFS library routines, listed in Table 6, enable you to manipulate directories, files, and their data from within a program. Most CMFS library routines have both a C-languages interface and a Fortran-languages interface:

- The C interface is used by programs written in C or C*.
- The Fortran interface is used by programs written in Fortran 90, Fortran 8x, Fortran 77, CM Fortran (CMF).

The CMFS routines can operate on a file within a CMFS file system or an SFS file system or other UNIX file system, depending on how the **CMFS_PATHTYPE** environment variable is set. See the examples in sections 1.3.1 and 2.3.1.

Table 6. The CM-5 CMFS library routines.

CMFS Routine	Purpose
`CMFS_access`	Determines the accessibility of a file.
`CMFS_chdir`	Changes the current working directory.
`CMFS_[f]chmod`	Changes a file's mode.
`CMFS_[f]chown`	Changes a file's owner and group.
`CMFS_close`	Closes a file.
`CMFS_close_all_files`	Closes all files; breaks TCP connections.
`CMFS_closedir`	Closes the directory and releases pointer.
`CMFS_close_files_on_server`	Closes server's files; breaks TCP connection.
`CMFS_creat`	Creates or recreates a file.
`CMFS_errmessage`	Stores the error message associated with the latest CMFS library error in a buffer.
`CMFS_errno`	Stores the last error's error number.
`CMFS_fcntl`	Controls various characteristics of a file.
`CMFS_getwd`	Returns the current CMFS file system's current working directory.
`CMFS_glob`	Prints the files that match the specified pattern(s), which enables wildcarding.
`CMFS_ioctl`	Supports an I/O subsystem device driver.

(continued)

Table 6 (cont'd). The CM-5 CMFS library calls.

CMFS Routines	Purpose
`CMFS_link`	Creates a hard link to a file.
`CMFS_mkdir`	Makes a directory.
`CMFS_mknod`	Creates a CM character-special file.
`CMFS_open`	Opens or creates and opens a file.
`CMFS_opendir`	Opens a directory and returns an identifying pointer.
`CMFS_perror`	Interprets an error condition.
`CMFS_physical_ftruncate`	Truncates or extends an open file.
`CMFS_physical_lseek`	Moves the read/write pointer associated with an open file.
`CMFS_read_file_always`	Performs synchronous parallel I/O (i.e., file to PNs).
`CMFS_readdir`	Returns a pointer to the next directory entry.
`CMFS_rename`	Renames a file.
`CMFS_rmdir`	Removes a directory.
`CMFS_scandir`	Builds array of pointers to a directory's entries.
`CMFS_seekdir`	Sets position of the next directory read operation.
`CMFS_serial_lseek`	Moves the pointer of an open file.
`CMFS_serial_read_file`	Reads into the memory of a client that is a serial computer.
`CMFS_serial_[f]truncate_file`	Truncates or extends an open file.
`CMFS_serial_write_file`	Writes from the memory of a client that is a serial computer.
`CMFS_set_debug_mode`	Enables/disables automatic CMFS debug-message printing.
`CMFS_[f]stat`	Obtains file status information.
`CMFS_statfs`	Obtains file system statistics.
`CMFS_telldir`	Returns a pointer to the current directory location.
`CMFS_unlink`	Removes a file from its directory.
`CMFS_utimes`	Sets files times.
`CMFS_vmeio_allocate`	Allocates DRAM on a VMEIO host computer.
`CMFS_vmeio_free`	Frees a buffer on a VMEIO host computer.
`CMFS_write_file_always`	Performs synchronous parallel I/O (i.e., PNs to file).

Appendixes

Appendix A

CM-5 CMFS Commands

NAME

cmchgrp - Changes the group ownership of a CMFS file.

SYNTAX

cmchgrp [-f] [-R] group filenames

ARGUMENTS

-f	Force. Do not report errors.
-R	Recursive. cmchgrp descends through the directory and any subdirectories, setting the specified group ID as it proceeds.
group	The new group of filenames.
filenames	The file(s) whose group is to be changed.

WHERE EXECUTED

Control processor
VMEIO host
CM-IOP

DESCRIPTION

cmchgrp changes the group ID (GID) of the filenames given as arguments to group. group can be either a decimal GID or a group name found in the GID file /etc/group.

This command uses the /etc/passwd and /etc/group files.

RESTRICTIONS

Only the owner of the filenames, or the superuser, can change the group of filenames.

Only one group number or list of groups is passed to the file server.

Permission checking is enabled only if the appropriate file server is set to check permissions (the default).

The attribute files associated with filenames do not have their protections changed by cmchmod.

SEE ALSO

cmchmod
cmchown
cmls
cmstat

cmchmod (1)

NAME

cmchmod - Changes the permissions mode of a CMFS file.

SYNTAX

cmchmod [-f] [-R] mode filenames

ARGUMENTS

-f	Force. cmchmod will not complain if it fails to change the mode of a file.
-R	Recursively descend through directory arguments, setting the mode for each file.
mode	The new mode of filenames.
filenames	The file(s) whose mode is to be changed.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

cmchmod changes the permissions (mode) of filenames. The mode of filenames is changed according to mode, which can be absolute or symbolic, as follows:

- An absolute mode is an octal number constructed from the OR of the following modes:
  - 0400 Read by owner.
  - 0200 Write by owner.
  - 0100 Execute (search in directory) by owner.
  - 0040 Read by group.
  - 0020 Write by group.
  - 0010 Execute (search) by group.
  - 0004 Read by others.
  - 0002 Write by others.
  - 0001 Execute (search) by others.
- A symbolic mode has the form:

  [who] op permission [op permission] ...

  who is a combination of:

  - u User permissions.
  - g Group permissions.
  - o Others.
  - a All (ugo).

  If who is omitted, the default is a.

  op is one of:

  - \+ To add the permission.
  - \- To remove the permission.
  - = To assign the permission explicitly (all other bits for that category, owner, group, or others, will be reset).

  permission is any combination of:

  - r Read.

cmchmod (1)

w Write.
x Execute.
X Give execute permission if the file s a directory or if there is execute permission for one of the other user classes.

For example, to take away all permissions, specify either -rwx or =.

Multiple symbolic modes, separated by commas, can be given. Operations are performed in the order specified.

This command uses the /etc/passwd and /etc/group files.

RESTRICTIONS

Only the owner of filenames (or the superuser) can change the mode of filenames.

Sticky directory bits are not enforced.

There is no umask handling or umask call.

Only one group numberno list of groupsis passed to the file server.

The attribute files associated with filenames do not have their protections changed by cmchmod.

Permission checking is enabled only if the appropriate file server is set to check permissions (the default).

EXAMPLE

The following code denies write permission to others.

```
% cmchmod o-w filename
```

SEE ALSO

cmchgrp
cmchown
cmls
cmstat

NAME

cmchown - Changes the owner of a CMFS file.

SYNTAX

cmchown [-f] [-R] owner[.group] filenames

ARGUMENTS

-f	Do not report errors.
-R	Recursively descend into directories, setting the ownership of all files in each directory encountered.
owner	The new owner of filenames.
group	The new group of filenames.
filenames	The file(s) whose owner and group are to be changed.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

cmchown changes the owner of filename to owner. The owner can be either a decimal user ID (UID) or a login name found in the password file. An optional group can also be specified. The group can be either a decimal group ID (GID) or a group name found in the GID file.

This command uses the /etc/passwd and /etc/group files.

RESTRICTIONS

Only the superuser can change the owner of filenames.

The attribute files associated with filenames do not have their protections changed by cmchown.

Permission checking is enabled only if the appropriate file server is set to check permissions (the default).

SEE ALSO

cmchgrp
cmchmod
cmls
cmstat

NAME

cmcp - Copies a file within a CMFS file system or from one CMFS file system to another.

SYNTAX

cmcp [-a] [-i] [-r] [-p] file1 file2
cmcp [-a] [-i] [-r] [-p] files directory

ARGUMENTS

-a	Appends the file named by file1 to the file named by file2 rather than overwriting the existing contents. It is an error if the destination file does not already exist.
-i	Interactively prompt for a yes or no response if an existing file will be overwritten. The prompt is the file's name and a question mark. If the response is y, the copy takes place and the existing file is overwritten; any other response prevents the copy.
-p	Preserve the modification times and modes of the file copied.
-r	Recursively copy the contents of directories files, subdirectories, and the subdirectories' contents that are supplied as the first argument, files, to the directory supplied as the second argument. The second argument cannot be a file.
file1	The file to be copied.
file2	The name of the copy.
files	One or more files to be copied.
directory	The directory into which to place the copies of files. The copies in the directory have the same file names as the original files.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

In its first form, the command cmcp copies file1 onto file2. If file2 already exists, the existing contents are overwritten but the mode and owner of file2 are preserved. If file2 does not exist, it is created and is given the same mode and owner as file1. In its second form, the second argument is a directory instead of a file name. One or more files are copied into directory. This command does not copy a file onto itself.

Note that cmcp duplicates a file residing in a CM file system, and places the copy in a CM file system. The command copytodv, however, duplicates a file residing in a local (or networked) UNIX file system, and places the copy in a CM file system. (copyfromdv duplicates a file residing in a CM file system, and places the copy in a local (or networked) UNIX file system.)

If the files systems where file1 and file2 reside are connected by a CMIO bus, the CMIO bus performs the copy; otherwise, the Ethernet performs the copy, albeit slowly.

SEE ALSO

cmmv
copyfromdv
copytodv

NAME

cmdd - Copies an input file to an output file, converting data as specified (supports both SFS and CMFS file systems).

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]

ARGUMENTS

-fromdv Use the input coming from a SFS or CMFS file (depending on the setting of **CMFS_PATHTYPE(7)**). (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 SFS or CMFS file (depending on the setting of **CMFS_PATHTYPE(7)**). (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 SFS or 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 SFS or 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 **block** 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.

WHERE EXECUTED

CP
VMEIO host computer
CM-IOP

DESCRIPTION

NOTE: **cmdd**'s former IBM-dataset-handling capabilities have been moved to a new utility, **ibmtape**, and further enhanced.

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.

RESTRICTIONS

cmdd does not support the use of multiple media, nor does it support files that span more than one tape volume.

EXAMPLES

The following example shows how to read an EBCDIC tape (**/dev/rmt0h**) into the ASCII file **x** in a 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 a CP.)

```
% 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 (dv1) to a tape (**/dev/rmt0h**). 64K bytes are transferred at a time and no conversions are performed. (**cmdd** is executed on a CP.)

```
% 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
```

SEE ALSO

cmcp
copyfromdv
copytodv
dvcp
cmtar
cmdump
cmrestore
CMFS_PATHTYPE(7)
ibmtape

NAME

cmdf - Displays free and used disk space for a CMFS file system.

SYNTAX

cmdf [-i] [files]

ARGUMENTS

-i	Report also the number of used and free inodes.
files	Files whose file systems' free and used disk space is to be reported.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

The command cmdf displays the amount of free and used disk space on CM file systems. The numbers are reported in kilobytes, and do not include disk space taken up by attribute file and directories.

If one or more files are specified, cmdf displays the amount of disk space available on the file system containing files. If no arguments or options are specified, cmdf displays both the used disk space and the free disk space on all of the file systems.

SEE ALSO

cmdu

NAME

cmdu - Summarizes CMFS file system disk usage.

SYNTAX

cmdu [-s] [-a] files

ARGUMENTS

-s	Display the number of kilobytes contained collectively in files.
-a	Display the disk usage for each file in files.
files	The names of the files or directories. If neither -s nor -a is specified, cmdu generates an entry only for each directory, not for files.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

The command cmdu displays the number of kilobytes contained in the specified files and, recursively, in the directories within each specified directory. (Attribute files are included in the number of kilobytes reported for directories.) If the files argument is not supplied, the current directory is used by default.

A file that has more than one link to it is only counted once. Block-special files (for example, raw-disk-special-file) are not counted.

SEE ALSO

cmdf

NAME

cmdump - Archives SFS or CMFS files or directories.

SYNTAX

cmdump [-b number] [-f filename]
[-n] [-s] [-v] [-V vol-name]

ARGUMENTS

-b number Set the archive's blocking factor. The default blocking factor of a Sun-4 is 3200; for a VAX, it is 20.

-f filename Specify the archive file. The default for a Sun-4 is /dev/rstc0. For a VAX, the default is /dev/rmt8. Multiple archives are supported.

-n Disable Emacs-style editing and history. If input is from a tty, Emacs-style and history is enabled by default. If input is not from a tty, editing and history are disabled by default.

-s Disable output pagination. If output is going to a tty, output pagination is enabled (the default). If output is not to a tty, output pagination is disabled by default.

-v Enable verbose mode. By default, verbose mode is disabled.

-V vol-name Specify a volume name for the archive. By default, no volume name is given. -V must be used on cmrestore if the archive was created with a volume name.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

Use cmdump to backup SFS or CMFS files to tape. cmdump provides a robust interface, composed of the commands listed below, that allows users to browse the current file system, mark the files to be backed up, and control some of the arguments passed to the cmtar that cmdump forks. Each file is archived with its relative pathname only; that is, the leading hostname:/ portion of the path is stripped off before archiving.

cmdump supports shell-like globbing (wildcarding), Emacs-style command-line editing and command history (see the GNU Readline Library), and selectable output pagination.

Note that cmdump supports multiple archives: you can specify up to four -f archive arguments. Specifying multiple archives is useful when using multi-volume mode with tape drives that take time to rewind, such as StorageTek. If the cmdump command line is ... -f /dev/rstc0 -f /dev/rstc1, for example, cmdump uses /dev/rstc1 while it is waiting for /dev/rstc0 to rewind, unload, and reload.

Before executing cmdump, set the environment variables CMFS_PATHTYPE, DVHOSTNAME and DVWD for the appropriate file system; although you can later change the current working directory within the file system, you cannot change the current file system itself once you invoke cmdump.

CMDUMP COMMANDS

The commands supported by cmdump are listed below in alphabetical order.

blocking blocking-factor
Set the archive's blocking factor.

cd directory Change the working directory for the ls, find, and mark commands.

display Print a list of the currently marked files.

emacsedit Enables/disables Emacs-style editing and history.

execute Run cmtar with the appropriate arguments on the marked files.

exit, quit Exit cmdump or cmrestore and return to the shell prompt.

find expression
find recursively descends the directory hierarchy from the current directory, seeking files that match a logical expression written using one of the available operators. It is almost identical to cmfind (see its man page in this appendix), with the following exceptions:

Unlike cmfind, this version does not take any pathname arguments.

There are two new operators: -mark and -unmark. These provide a flexible method to (un)mark files based on any of the characteristics supported by cmfind.

Note: do not specify -print and -(un)mark in the same invocation.

help, ? Print a brief synopsis of all available commands.

ls [-aAcdfFgilqrRstu1] files
List the contents of a directory. This command is identical to cmls (see the cmls man page in this appendix), with the exception of the -C (columnar output) flag, which here is the default, and which can be overriden by the -1 (the numeral 1) flag.

mark [-r|-n] files [[-r|-n] files]
Mark files for backup. The -r flag specifies that the filenames following it (until the next -n flag) are directories, and all files in the subtree rooted at those directories are to be marked. The -n flag (the default) specifies that all files following it (until the next -r flag) are not to be marked recursively.

Specifying a directory to be marked non-recursively has no effect. Specifying a regular file to be marked recursively is the same as specifying it non-recursively. In order for a file to be marked, it must reside in either the current directory or one of its sub-directories.

In verbose mode, all files being marked that aren't already marked are listed. See also unmark.

more Toggle output pagination via the UNIX more command. This command has no effect on the output of the forked cmtar.

prompt string
Set the command-line prompt to string.

pwd Print the name of the file system's current working directory.

status Display status information about cmdump, including archive filename, blocking factor, current directory, verbose mode, prompt, Emacs-editing/history mode, and output pagination mode.

unmark [-r|-n] files [[-r|-n] files]
unmark specifies that previously marked files should be unmarked. The -r flag specifies that all files following it (until the next -n flag) are directory names, and all files in the subtree rooted at those directories are to be unmarked. The -n flag (the default) specifies that all files following it (until the next -r flag) are not to be unmarked recursively.

Specifying a directory to be unmarked non-recursively has no effect. Specifying a regular file to be unmarked recursively is the same as specifying it non-recursively. In order for a file to be marked, it must reside in either the current directory or one of its sub-directories.

In verbose mode, all files being unmarked that aren't already unmarked are listed. See

also mark.

verbose Enable/disable verbose mode. In verbose mode, cmdump prints the names of the files as they are marked or unmarked.

volname volume-name
Specify the volume name of the archive.

EXAMPLE

```
% CMFS_PATHTYPE cmfs
% setenv DVHOSTNAME dv1
% setenv DVWD /big_project
% cmdump
Using /dev/rstc0 for archive Current directory is dv1:/big_project
cmdump% ls
june4   june3   june2
june1   may31   may 30
jim1    sean    test
cmdump% mark -r sean -n may30 may31
cmdump% display
dv1:/big_project/

dv1:/big_project/may30
dv1:/big_project/may31
dv1:/big_project/sean/
dv1:/big_project/sean/abc
dv1:/big_project/sean/tree
cmdump% unmark may31
cmdump% display
dv1:/big_project/
dv1:/big_project/may30
dv1:/big_project/sean/abc
dv1:/big_project/sean/tree
cmdump% volname test
cmdump% execute
cmdump% (cmtar -c -M -V test -b 3200 -f /dev/rstc0 -T - -G)
cmtar: Removing datavault spec from names in the archive.
cmtar: Removing leading / from absolute pathnames in the archive.
cmtar: Prepare volume #2 (/dev/rstc0), then hit return
cmdump% exit
```

RESTRICTIONS

As when using cmtar or UNIX tar, you must have read permission on each file and directory that you archive. Archives created by cmdump cannot be restored using UNIX tar.

SEE ALSO

cmrestore
cmtar

cmfind (1)

NAME

cmfind - Finds a CMFS file.

SYNTAX

cmfind pathnames expression

ARGUMENTS

pathnames	The directory hierarchies to be searched.
expression	A boolean expression describing the files that are to be matched by cmfind. The boolean primaries listed below may be used for expression.

DESCRIPTION

The command cmfind recursively descends the directory hierarchy for each pathname in pathnames, seeking files that match the boolean expression. In the boolean expressions, the argument n is used as a decimal integer where +n means more than n, -n means less than n, and n means exactly n.

-atime n	Tests true if the file has been accessed within the last n days.
-exec command	Tests true if the executed command returns a zero value as exit status. The end of the command must be punctuated by an excaped semicolon (;) A command argument '{}' is replaced by the current pathname.
-group gname	Tests true if the file belongs to group gname. gname may be either a group name or a numeric group ID.
-inum n	Tests true if the file has inode number n.
-links n	Tests true if the file has n links.
-mtime n	Tests true if the file has been modified within the last n days.
-name filename	Tests true if filename matches the current file name. Normal C shell argument syntax may be used if escaped (preceded by a character). Watch out for '[', '?' and '*'.
-newer filename	Tests true if the current file has been modified more recently than the argument filename.
-ok command	Executes specified command like -exec, but first writes the generated command on standard output, then reads standard input, and executes the command executed upon response y.
-perm onum	Tests true if the file's permission number exactly matches the octal number onum. For further information, see chmod(1). If onum is prefixed by a minus sign, more flag bits (017777) become significant and the flags are compared: (flags&onum)==onum. For further information, see CMFS-stat.
-print	Always true; the current pathname is printed.
-size n	Tests true if the file is n blocks long (512 bytes per block).
-type c	Tests true if file is c type, where c is one of the following: b block special file c character special file d directory f plain file
-user uname	Tests true if the file belongs to the user uname. uname may be either a login name or numeric user ID.

The boolean primaries above may be combined using the following operators (in order of decreasing precedence):

(...)	A parenthesized group of primaries and operators. Parentheses are special to the shell and must be escaped (preceded by a character).
!primary	The negation of a primary ('!' is the unary not operator).
primary primary	Concatenation of primaries (the and operation is implied by the juxtaposition of two primaries).
primary -o primary	Alternation of primaries (-o is the or operator).

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

EXAMPLE

To remove all files named oldata or *.tmp that have not been accessed for a week:

```
cmfind /  -name oldata -o -name '*.tmp' ) -atime +7 \
    -exec cmrm '{}' ;
```

SEE ALSO

cmglob

NAME

cmglob - Prints the CMFS filenames that match the pattern(s) given by the input argument.

SYNTAX

cmglob filenames

ARGUMENTS

filenames — The name of one or more files for which to find matches. filenames does not have to be the complete name of a file--it can be simply one or more characters.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

cmglob provides a workaround to the inconvenience that arises from the inability of CMFS commands such as cmrm
and cmls to accept a wildcard in an argument. In effect, cmglob provides the CMFS commands with the ability to accept wildcarding.

cmglob takes as an argument one or more filenames, which can contain shell-type wildcard characters (*, ?, []), and it output a list of all CMFS files that are matched by the patterns given. Any wildcard characters used must be escaped either preceded by a backslash character () or enclosed within single or double quotes to prevent the shell from interpreting them. cmglob searches only the current directory; unlike cmfind, it does not search directories recursively.

Used as input to another CMFS command, such as cmrm or cmls, the filenames that make up its output are separated by a space, causing them to be accepted as a list by the oither command.

When cmglob is used on its own, the filenames that it prints are listed one per line, separated by a new-line.

EXAMPLE

```
% cmls -l
total 20640
-rw-rw-rw  1 joe    19824640 Aug 22 17:20 conditional.7385
-rw-rw-rw  1 joe      524288 Aug 22 18:20 reliability.7565
-rw-rw-rw  1 joe      262144 Aug 22 18:29 reliability.7619
-rw-rw-rw  1 root          0 Aug 23 16:54 transpose.14898
-rw-rw-rw  1 root          0 Aug 23 17:49 transpose.15052
-rw-rw-rw  1 root     524288 Aug 23 19:41 transpose.15063
% cmglob \*15\*
transpose.15063
transpose.15052
% echo `cmglob \*15\*`
transpose.15063 transpose.15052
% cmls -l `cmglob \*15\*`
-rw-rw-rw  1 root          0 Aug 23 17:49 transpose.15052
-rw-rw-rw  1 root     524288 Aug 23 19:41 transpose.15063
```

SEE ALSO

cmls
cmrm
cmfind

NAME

cmln - Makes links to CMFS files or directories.

SYNTAX

cmln [-f] file [linkname]
cmln files directory

ARGUMENTS

-f	Suppress all but the usage message.
file	The file or directory to be linked.
linkname	The new name to be associated with file.
files	Files to be linked; the new names are directory/ files.
directory	The new directory name to be associated with files.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

The command cmln assigns an additional name (directory entry), called a link, to a CMFS file or directory. A file, together with its size and all its protection information, may have several links to it.

The command cmln creates hard links. A hard link to a file is indistinguishable from the original directory entry. Any changes to a file are effective independent of the name used to reference the file. Hard links may not span file systems, may not refer to directories, and can be made only to an existing file.

In cmln's first form, file is the name of the file or directory to be linked and linkname is the new name to be associated with file. If linkname is omitted, the last component of the pathname given as file is used. In cmln's second form, links are made in the named dire ctory to all the named files. These links will have the same file names, within directory, as the files to which they are linked.

RESTRICTIONS

In a UNIX file system there are two kinds of links--hard links and symbolic links. The cmln command creates only hard links; the UNIX ln command creates either kind.

SEE ALSO

cmcp
cmmv
cmrm
CMFS-stat

NAME

cmls - Lists the contents of a CMFS file system directory.

SYNTAX

cmls[-acdfgilqrstu1ACFR] files

-a Display all entries including those beginning with a period (.).

-c Sort entries by time of last inode change instead of by name. The inode is changed if the files' size, links, or permissions change.

-d Display names of directories only, not contents. Use -d with -l to display the status of a directory.

-f Display names in the order they exist in the directory. (For more information, see dir(5).) Entries beginning with a period (.) are also listed. This option overrides the -l, -t, -s, and -r options.

-g (Use with -l only). Cause the -l option to also display the assigned group ID . The default is to display the assigned owner ID.

-i Display the i-number for each file in the first column of the report.

-l List the long format, giving the mode, number of links, owner, size in bytes, and time of last modification for each file. See -f. If the file is a special file, the size field reports the major and minor device numbers instead of the file's size.

The mode field consists of 10 characters. The first character indicates the type of entry:

d Directory
b Block-type special file
c Character-type special file
- Plain file

The next nine characters are interpreted as three sets of three characters each. The first set of three characters refers to file-access permissions for the user; the next set, for the group; and the last set, for all others. The permissions are indicated as follows:

r The file is readable
w The file is writable
x The file is executable
- The indicated permission is not granted.

-q Force non-graphic characters in file names to be printed as the question mark character (?). This is the default when output is to a terminal.

-r Sort entries in reverse alphabetic or time order. See -f.

-s Display the size in kilobytes of each file. This is the first item listed in each entry. See -f.

-t Sort by time the files' contents were modified (latest first) instead of by name. See -f.

-u Use the time of last access instead of last modification for sorting (with the -t option) or printing (with the -l option).

-1 Display one entry per linethe default when the output is not to a terminal.

-A Display all entries including those beginning with a period, except for . (current directory) and .. (parent directory).

-C Force multi-column output for pipe or filter. This is the default when the output is to a terminal.

-F Mark directories with trailing slash (/).

-R Recursively list all subdirectories.

files The files or directories to list. If this argument is one or more directories, the contents of the directories are listed.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

cmls is used to list files, and information about them, that are in the CM file system. When the argument is one or more directories, cmls lists the contents of those directories; when the argument is one or more files, cmls repeats each file name and gives any other information you request using the options.

By default, the list is sorted alphabetically. When no argument is given, the current directory is listed. When several arguments are given, files are listed first, followed by directories and the files within each directory.

File names of the form hostname:pathname are valid arguments. hostname specifies a device, other than the default (see DVWD and DVHOST NAME), containing the files you want to list.

This command uses the following files:

/etc/passwd	Used to obtain user IDs for cmls -l
/etc/group	Used to obtain group IDs for cmls -g

RESTRICTIONS

The output device is assumed to be 80 columns wide. Newline and tab are considered printing characters in file names.

This command, like all CMFS commands, does not accept wildcards in its arguments. Using cmglob in conjunction with cmls, however, provides a workaround to this restriction, in effect providing c mls the ability to use wildcarding.

SEE ALSO

UNIX chmod(1)
DVHOSTNAME (env. variable)
cmglob

NAME

cmmkdir - Makes a CMFS file system directory.

SYNTAX

cmmkdir directories

ARGUMENTS

directories	Directories to be created.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

The command cmmkdir creates the specified directories in the CMFS file system. Directories are created with read, write, and execute permissions for user, group, and all others (mode 777). Standard entries (. for the directory itself and .. for its parent) are made automatically.

This command requires write permission in the parent directory.

SEE ALSO

cmrm
cmrmdir

NAME

cmmknod - Makes a CM character-special file.

SYNTAX

cmmknod filename c major-device-number minor-device-number

ARGUMENTS

filename The name of the CM character-special file.

major-device-number
The number of the file's structure entry in the device switch cm_cdevsw (specified by the CM character-special file's inode).

minor-device-number
The CM character-special file's minor device number, as specified by the file's inode.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

The CM file system views I/O devices, such as tape drives, as CM character-special files. To make a special device entry in the CM file system directory tree served by an fsserver process, execute cmmknod as follows:

```
% cmmknod filename c major-device-number minor-device-number
```

where filename is the name by which the CM file system knows your device. Include the hostname: component of the path namethe name of the computer the device is connected to, as given in that computer's /etc/hosts table (for example, vme1:/dev/new-device).

Obtain the major-device-number from the device switch cm_cdevsw: if your device's structure is the nth one in the device switch, major-device-number is n-1 (structures in the switch are numbered from 0).

minor-device-number can specify the device's unit number, drive number, line number, and the like. If more than one minor device number is associated with the device, execute cmmknod as many times as necessary, specifying one minor device number at each execution. (The minor device number is not used by the CMFS library or by fsserver; it is passed to the driver via the dev argument to CMFS-ioctl, and can be used as you see fit.

SEE ALSO

CMFS-fcntl

NAME

cmmv - Moves (renames) a CMFS file system file or directory.

SYNTAX

cmmv [-i] [-f] [-] file1 file2
cmmv [-i] [-f] [-] files directory

ARGUMENTS

-i Interactively prompt for a yes or no response whenever a move results in overwriting an existing file. The prompt is the file's name and a question mark. If the response is y, the move takes place and the existing file is overwritten; any other response prevents the move.

-f Force the move to proceed regardless of any restrictions imposed by the mode or by the -i option.

- Interpret all following arguments as file names, to allow file names starting with a minus sign.

file1 The file or directory to move (rename).

file2 The file or directory name to which to move (rename) file1.

files Files or directories to move into directory.

directory The directory into which to place files.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

The command cmmv moves a CMFS file or directory file1 to file2, in effect changing its name from file1 to file2. In the second form, one or more files (plain files or directories) are moved to directory, retaining their file names. This command refuses to move a file onto itself.

If file2 already exists, it is removed before file1 is moved. If file2 has a mode that forbids writing, cmmv prints the mode and queries the user as to whether to complete the move. If the user responds with y, the move takes place; any other response causes cmmv to exit.

RESTRICTIONS

cmmv may not be used to move a file between CMFS file systems. Use cmcp followed by cmrm to move a file between file systems.

SEE ALSO

cmcp
cmrm
copyfromdv
copytodv

NAME

cmptruncate - Truncates or extends a file.

SYNTAX

cmptruncate *filename number-cmwords number-extra-bytes*

filename The file to be truncated or extended.

number-cmwords

The number of cmwords you want the file *filename* to contain. (Usually, 1 cmword = 512 bytes; verify by executing **cmstat** on the file.)

number-extra-bytes

The number of additional files -- that is, additional to the number of cmwords specified by *number-cmwords* -- you want the truncated file to contain.

WHERE EXECUTED

Control Processor
VMEIO host computer
CM-IOP

DESCRIPTION

cmptruncate truncates or extends a file to any non-negative value, including a value larger than the (2 Gbytes -1) limit imposed by **cmtruncate**. At the conclusion of execution, the total file size (in bytes) = (*number-cmwords* * cmword size in bytes) + *number-extra-bytes*.

SEE ALSO

cmtruncate

cmrestore (1)

NAME

cmrestore - Restores (extracts) files and directories from an archive (supports SFS and CMFS file systems).

SYNTAX

cmrestore [-b number]
[-f filename] [-n] [-s]
[-v] [-V vol-name]

ARGUMENTS

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

Use cmrestore to extract files from the tape archive and place them into the current file system. When executed, cmrestore builds a tree representation of the archive's directory information and provides an interface, composed of the commands listed below, that lets you browse through the tree, mark the files to be extracted, and control some of the arguments passed to the cmtar that cmre store forks. Files are extracted relative to the directory that is current when cmrestore is invoked. cmrestore supports shell-like globbing (wildcarding), Emacs-style command-lin e editing and command history (see the GNU Readline Library), and selectable output pagination.

Note that cmrestore supports multiple archives: you can specify up to four -f archive arguments. Specifying multiple archives is useful when using multi-volume mode with tape drives that take time to rewind, such as StorageTek. If the cmrestor e command line is ... -f /dev/rstc0 -f /dev/rstc1, for example, cmrestore uses /dev/rstc1 while it is waiting for /dev/rstc0 to rewind, unload, and reload.

Before executing cmrestore, set the environment variables DVHOSTNAME, DVWD, and CMFS_PATHTYPE(7) for the appropriate file system; although you can later change the current working directory within the file system, you cannot change the current file system itself once you invoke cmrestore.

cmrestore Commands

The commands supported by cmrestore are listed below in alphabetical order.

blocking blocking-factor
Set the archive's blocking factor.

cd directory
Change the current directory within the archive's tree for the ls and mark commands.

display Print a list of the currently marked files.

Emacsedit
Enables/disable both the Emacs-style command-line editing and history mechanisms.

execute Run cmtar with the appropriate arguments on the marked files.

exit, quit
Exit cmdump or cmrestore and return to the shell prompt.

help, ? Print a brief synopsis of all available commands.

ls List the contents of a directory. ls is a simplistic version of cmls that lists names only, does not provide columnated output, and takes no arguments other than one or more file names.

mark [-r|-n] files [[-r|-n] files]
Flag files for restoration. The -r flag specifies that all files following it (until the next -n flag) are directory names, and all files in the subtree rooted at those directories are to be marked.

The -n flag (the default) specifies that all files following it (until the next -r flag) are not to be marked recursively.

Specifying a directory to be marked non-recursively has no effect. Specifying a regular file to be marked recursively is the same as specifying it non-recursively. In order for a file to be marked, it must reside in either the current directory or one of its sub-directories.

In verbose mode, all files being marked that aren't already marked are listed. See also unmark.

more Toggles output pagination via the UNIX more command. This command has no effect on the output of the forked cmtar.

prompt string
Set cmrestore's command-line prompt to string.

pwd Print the name of the current working directory (a directory within the archive's tree).

status Display status information about cmrestore: archive filename, blocking factor, current directory, verbose mode, prompt, Emacs-editing/history mode, and output pagination mode.

unmark [-r|-n] files [[-r|-n] files]
Specify that previously marked files should be unmarked. The -r flag specifies that all files following it (until the next -n flag) are directory names, and all files in the subtree rooted at those directories are to be unmarked. The -n flag (the default) specifies that all files following it (until the next -r flag) are not to be unmarked recursively.

Specifying a directory to be unmarked non-recursively has no effect. Specifying a regular file to be unmarked recursively is the same as specifying it non-recursively. In order for a file to be marked, it must reside in either the current directory or one of its sub-directories.

In verbose mode, all files being unmarked that aren't already unmarked are listed. See also mark.

verbose Toggle verbose mode. In verbose mode, cmrestore prints the names of the files as they are marked or unmarked.

EXAMPLE

This is a continuation of the example presented in the cmdump man page.

```
% setenv CMFS_PATHTYPE cmfs
% setenv DVHOSTNAME dv1
% setenv DVWD /big_project
% cmrestore
Retrieving directory info from archive(s) /dev/rstc0...done
Current directory is big_project
Files will be restored into CMFS directory dv1:/big_project
cmrestore% ls
big_project/sean/
big_project/may30
cmrestore% cd sean
cmrestore% ls
abc
tree
cmrestore% cd ..
cmrestore% mark -r sean
cmrestore% display
big_project/sean/
big_project/sean/abc
```

cmrestore (1)

```
big_project/sean/tree
cmrestore% execute
(cmtar -x -M -b 3200 -f /dev/rstc0 -T -)
cmrestore% exit
%
```

RESTRICTIONS

As when using cmtar or UNIX tar, you must be owner of the extracted files, and you must have write permission on the directory to which you restore them. Also, if a file with the same name as an extracted file already exists in the directory, you must have write permission on that file, as it will be overwritten.

cmrestore cannot extract files or directories archived by UNIX tar.

SEE ALSO

cmdump
cmtar

NAME

cmrm - Removes (unlinks) a CMFS file or directory.

SYNTAX

cmrm [-f] [-i] [-r] [-] files

ARGUMENTS

-f Force the file to be removed without first requesting confirmation. Only system or usage messages are displayed.

-i Interactively prompt for a yes or no response before removing each file. Do not prompt if combined with the -f option.

-r Recursively delete the contents of the specified directory, its subdirectories, and the directory itself.

- Interpret all following arguments as file names, to allow file names starting with a minus sign.

files The files or directories to be removed.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

The command cmrm removes one or more CMFS files and their directory entries. If a directory entry was the last link to a file, the file's contents are lost. The commands cmrm -r and cmrmdir remove CMFS directories. The cmrmdir command removes a directory only if it is empty; cmrm -r removes a directory if it first removes its files and subdirectories.

To remove a file, you must have write permission in its directory, but you do not need read or write permission on the file itself. If you don't have write permission on the file and you are using cmrm from a terminal, cmrm asks for confirmation before destroying the file. If your response begins with y the file is deleted; otherwise it is not deleted.

If the input to cmrm has been redirected from another command or program, cmrm checks to be sure it is not coming from your terminal. If it is not, cmrm sets the -f option, which overrides the file protection, and removes the files silently regardless of what you have specified in the redirected input. For example, suppose you don't have write permission on the file water. Suppose also the file response contains a response of n. The following deletes the file, even though the redirected input file contains a response of n:

```
cmrm < response water
```

RESTRICTIONS

Wildcards will not work when the files you are referencing are not mounted on the system you are logged in to. Using cmglob in conjunction with cmrm, however, provides a workaround to this restriction, in effect providing cmrm the ability to use wildcarding.

SEE ALSO

cmrmdir
cmln
cmglob

For information about directory entries, see directory operations.

NAME

cmrmdir - Removes (unlinks) an CMFS file system empty directory.

SYNTAX

cmrmdir directories

ARGUMENTS

directories Directories to be removed.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

The command cmrmdir removes one or more directories (and their directory entries) in the CMFS file system. The directories must be empty; if a directory is not empty cmrmdir displays an error message and does not remove it.

SEE ALSO

cmrm

NAME

cmstat - Prints status information about a CMFS file.

SYNTAX

cmstat files

ARGUMENTS

files One or more files about which to print information.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

cmstat prints information about the status of one or more files in the CMFS file system, including its type, mode, owner, size, and time of last access, modification, and change.

The Size field (see the sample output, below) shows the 32-bit byte count returned by CMFS_stat. The True file size field shows the actual file size, calculated using the Size in CM words field, the CMword size (bits) field, and the Free bytes in last CMword field.

The times shown in parentheses in the Access, Modify, and Change fields indicate the amount of time since access, modification, or change, respectively, occurred. The units are (days.hours:minutes:seconds). Changes of owner, group, link count, or mode set the Change field but not the Modify field.

EXAMPLE

A sample output:

```
% cmstat file
File: "file"
 Filetype: Regular File
 Mode: (0666/-rw-rw-rw-) UID: ( 1155/username) Gid: ( 10/ staff)
 Device: 907 Inode: 453 Links: 1
  True file size: 60817408 bytes
   Size: 60817408
   Allocated Blocks: 118784
   Free bytes in last CMword: 0
   Extents on disk: 1
   Size in CM words:  118784
   CMword size (bits): 4096
   Optimal Blocksize: 131072
  Access: Fri May 18 14:53:22 1990(00000.00:04:02)
  Modify: Fri May 18 14:45:34 1990(00000.00:11:50)
  Change: Fri May 18 14:50:38 1990(00000.00:06:46)
```

SEE ALSO

CMFS_stat

NAME

cmtar - Archives a file (tape or other media) (supports SFS and CMFS file systems).

SYNTAX

cmtar command [options] [option argument] [files]

ARGUMENTS

command — One of the following must be specified. For the commands only, the preceding hyphen is optional.

-c — Create a new archive containing files. If files are not specified, no files are archived. If the archive file already exists, it is overwritten; the old contents are lost.

-d — Compare the archive with the files in the file system and report differences in file size, mode, owner, and contents; also report it if a file exists in the archive but not in the file system. If files are specified, they are compared with the tape and they must all exist in the archive. If files are not specified, all the files in the archive are compared.

-r — Add files to the end of the archive. Files must be specified. The archive file must already exist and must be in the proper format (which probably means it was created previously with cmtar). If the archive is not in a format that cmtar understands, the results will be unpredictable.

-t — Display a list of the files in the archive. If files are specified, only those specified that appear in the archive are listed.

-u — Add files to the end of the archive, but only under one of two conditions: if a file is not already in the archive, or if a file is newer than the version in the archive (the time of last modification is compared). Files must be specified. This command can be very slow to execute.

-x — Extract files from the archive. If files are not supplied, all the files in the archives are extracted.

-A — Concatenate several archive files into one big archive file. All files should be archive files; these are appended to the end of the archive file on which cmtar is operating (see the option -f). The original files are not changed. The UNIX cat command cannot be used to concatenate archive files because each archive contains an end-of-archive marker, so material added to the end by cat is ignored. The -A command removes the end-of-archive markers before appending archive files.

-D — Delete files from the archive. This command is extremely slow. Warning: Deleting files from an archive stored on magnetic tape may result in a scrambled archive, because there is no safe way, except for completely rewriting the archive, to delete files from a magnetic tape.

options

options argument

Zero or more of the options below may be specified. Some of the options are meaningful only with certain commands. The options below are grouped according to the commands with which they are used.

Options that do not take arguments may appear either after command (with no hyphen and with no space between command and options), or separately (with the hyphen). The single exception is -version, which must be specified separately to avoid confusion with -v.

Options that take an argument must appear separately (with the hyphen) and must be immediately followed by the argument.

The following options are meaningful with any of the commands:

-b number Specify a blocking factor for the archive; cmtar reads and writes the archive in blocks of number x 512 bytes. The default blocking factor is set when cmtar is compiled, and is usually 3200. A blocking factor of 3200 is fine for the DataVault and the StorageTek tape drive, but it is too high for most standard tape drives. We recommend using a blocking factor of 126 for standard reel tape drives, 1/4" Sun catridge tape drives, and the like. (The maximum size of the blocking factor is limited only by the maximum block size of the device containing the archive, or by the amount of available virtual memory.)

Use the same blocking factor when creating, updating, and extracting an archive.

-f filename Specifies the file name of the archive on which cmtar operates.

Multiple archives are supported: you can specify up to four -f a rchive arguments. Specifying multiple archives is useful when using multi-volume mode with tape drives that take time to rewind, such as StorageTek. If the cmtar command line is ... -f /dev/rstc0 -f /dev/rstc1, for example, cmtar uses /dev/r stc1 while it is waiting for /dev/rstc0 to rewind, unload, and reload.

If -f is not supplied, but the shell environment variable TAPE exists, its value is used; otherwise, a default archive name (picked when cmtar was compiled) is used. The default is normally set to /dev/rstc0, which is the first StorageTek tape drive (if the system has a StorageTek unit).

cmtar supports remote access only to magnetic tape device files, but not to regular files. Therefore, if you are accessing a remote magnetic tape device, filename can be specified as hostnam e:filename (filename must start with /dev/). If hostname is not indicated, the tape drive on the machine you're logged in to is used. If hostname is given, the tape drive on hostname is used. If hostname contains the @ symbol, it is treated as user@hostname: filename. If hostname indicates a remote tape drive, cmtar invokes the command rsh (or remsh) to start up an /etc/rmt on the remote machine. If you give an alternate login name, it will be given to the rsh. Naturally, the remote machine must have a copy of /etc/rmt. The program /etc/r mt is free software from the University of California, and a copy of the source code can be found with the sources for cmtar. This program must be modified to run on non-BSD4.3 UNIX systems.

If filename is -, cmtar reads the archive from standard input (when listing or extracting), or writes it to standard output (when creating). If - is given when updating an archive, cmtar reads the original archive from its standard input and writes the entire new archive to its standard output.

-C dir Change the working directory to dir before continuing. This option is usually interspersed with the files on which cmtar is to operate. It is especially useful when several files from different directories are to be stored in the same directory in the archive. For example, executing the following command places on a tape the files iggy and ziggy from the current directory, followed by the file melvin from the directory baz.

```
% cmtar -c iggy ziggy -C baz melvin
```

The file melvin is recorded in the archive under the precise name melvin, not baz/melvin. Thus, the archive contains three files that all appear to have come from the same directory; if the archive is extracted with cmtar -x, all three files will be created in the current directory. In contrast, the following command records the third file in the archive under the name bar/melvin, so that cmtar -x creates the third file in a subdirectory named bar.

```
% cmtar -c iggy ziggy bar/melvin
```

-M Write a multi-volume archive: one that may not fit on the medium used to store it. Instead of aborting when it cannot read or write any more data, cmtar asks for a new volume. For example, if the archive is on a magnetic tape, the tape must be changed. Each volume of a multi-volume archive is an independent cmta r archive. Any volume can be listed or extracted alone, as long as -M is not specified. However, if a file in the archive is split across volumes, the only way to extract it successfully is with a multi-volume extract command (-xM) starting on or before the volume where the file begins.

-N date Only operate on files whose modification or inode-changed times are newer than date. When creating an archive, only new files are written; when extracting an archive, only newer files are extracted. D ate must be quoted if it contains any spaces. It is parsed using getd ate.

-R Print, along with every message normally produced, the record number within the archive where the message occurred. This option is especially useful when reading damaged archives because it helps pinpoint the damaged sections. -R can also be useful when making a log of a file system backup tape. Its results allow you to locate the file you want to retrieve on several backup tapes, and choose the tape where the file appears closest to the front of the tape.

-T filename Instead of taking the list of files on which to operate from the command line, read it from filename. If filename is -, the list is read from standard input. Specifying both -T - and -f - at once can only be done with the -c command.

-V name Write out a volume header at the beginning of the archive. If -M is specified, each volume of the archive is given a volume header of name Volume n" where n is 1 for the first volume, 2 for the next, etc.

-v Be verbose about the actions taken. Normally, the command to list an archive (-t) prints just the file names and the other commands are silent. The command -tv prints a full line of information about each file, like the output of cmls -l. The -v option with any other command prints the name of each file operated on. The output from -v appears on the standard output except when creating or updating an archive to the standard output, in which case the output from -v is sent to the standard error.

-version Print the version number of cmtar to the standard error. To avoid confusion with the -v option, -version must be given as a separate option preceded by a hyphen.

-w Print a message for each action cmtar intends to take, requesting confirmation. If the response begins with y the action is performed; otherwise it is not. Actions requiring confirmation include adding a file to the archive, extracting a file from the archive, deleting a file from the archive, and deleting a file from disk. If cmtar
is reading the archive from the standard input, it opens the file /dev/tty to ask for confirmation.

-X file Read a list of file names (actually regular expressions) from file and ignore files with those names. For example, cmtar -c -X foo . prevents files in the current directory whose names end in .o from being added to the archive. Multiple -X options may be specified.

-Z, -z Compress the archive as it is written, or decompress it as it is read, using the compress program. This option works on physical devices (tape drives, etc.) and remote files as well as on normal files; data to or from such devices or remote files is reblocked by another copy of the cmtar program to enforce the specified (or default) block size. The default compression parameters are used; to override them run compress explicitly instead of using this option. This option cannot be used with the option -M, nor with the commands -u, -r, -A, and -D.

The following options are meaningful with the commands for creating or updating an archive (-c, -r, -u, -A, and -D). These options control which files are placed in an archive and the format in which the archive is written.

-G At the beginning of the archive, write an entry for each of the directories on which cmtar will operate. -G is only used when creating an incremental backup of a file system. The entry for each directory includes a list of all the files in the directory at the time of the backup, and a flag for each file indicating whether the file is going to be put in the archive. This information is used when doing a complete incremental restore. Note: this option causes cmtar to create a non-standard archive that may not be readable by non-GNU versions of the UNIX tar program.

-h If a symbolic link is encountered, archive the file to which it is linked instead of simply recording the presence of a symbolic link. If the file is archived again, an entire second copy of it is archived instead of a link.

-l Do not cross file system boundaries when archiving parts of a directory tree. This option does not affect files on the command line; these are archived even if they reside in various file systems. Rather, this option only affects files that are being archived because they reside in a directory that is being archived. Any files not archived because of -l are printed on the standard error. This option is useful for making full or incremental archival backups of a file system, as with the UNIX dump command.

-o Write an old format archive, which does not include information about directories, pipes, FIFOs, contiguous files, or device files, and which specifies file ownership by numeric user and group IDs rather than by user and group names. In most cases, a new format archive can be read by an old UNIX tar program without serious trouble, so this option should seldom be needed. When updating an archive, do not use -o unless the archive was created with -o.

-W Verify the archive after writing it. Each volume is checked after it is written and any discrepancies are recorded on the standard error. Verification requires the archive to be on a medium capable of back-spacing. This means pipes, some cartridge tape drives, and some other devices cannot be verified.

The following option is meaningful with the commands for updating an archive (-r, -u, -A, and -D), listing files (-t), and extracting files (-x).

-B If an attempt to read a block from the archive does not return a full block, keep reading until a full block is obtained. This behavior is the default when cmtar is reading an archive from standard input or from a remote machine. This is because on certain U NIX systems a read of a pipe the amount in the pipe even if it is less than the amount cmtar requested. Without this option, c mtar would fail as soon as it read an incomplete block from the pipe.

The following options are meaningful with the commands for listing files in an archive (-t) and extracting files from an archive (-x).

-G The archive is an incremental backup. The behavior of -G depends on which command it modifies.

With -t: for each directory in the archive, list the files in that directory at the time the archive was created. The format of this list, while not easy for users to read, is unambiguous for a program: each file name is preceded by Y if the file is present in the archive, by N if the file is a directory, or by nothing if it is not in the archive. Each file name is terminated by a null character. The last file is followed by an additional null character and a newline to indicate the end of the data.

With -x: when the entry for a directory is found, delete all files in that directory that are

not listed in the archive. This behavior is convenient when restoring a damaged file system from a succession of incremental backups: it restores the entire state of the file system to that which obtained when the backup was made. With out -G, the file system will probably fill up with files that shouldn't exist any more.

-i Ignore blocks of zeros in the archive. Normally a block of zeros indicates the end of the archive. This option allows cmtar to read archives containing blocks of zeros, such as damaged archives, or archives created by concatenating several archives together with cat. This option is not on by default because many versions of tar write garbage after blocks of zeros. Note that this option causes cmtar to read to the end of the archive file, which may sometimes avoid problems when multiple files are stored on a single physical tape.

-K filename Begin extracting or listing the archive with the file filename; consider only the files starting at that point in the archive. This is useful if a previous attempt to extract files failed when it reached filename due to lack of free space.

-s Indicates that the list of file names to be listed or extracted is sorted in the same order as the files in the archive. This option allows a large list of names to be used, even on a small machine that would not otherwise be able to hold all the names in memory at the same time. Such a sorted list can be created by executing c mtar -t on the archive and editing its output. This option is unlikely to be needed on modern computer systems.

The following options are meaningful with the command for extracting files from an archive (-x).

-k Do not overwrite existing files with files of the same name from the archive.

-m Leave the modification times of the files cmtar extracts as the time when the files were extracted, instead of setting it to the time recorded in the archive.

-O Instead of creating the files specified, write the contents of the files extracted to the standard output. This may be useful when extracting the files in order to send them through a pipeline.

-p Set the modes (access permissions) of extracted files exactly as recorded in the archive. If this option is not used, the current umask setting limits the permissions on extracted files.

files One or more SFS or CMFS files to be stored in an archive file. An archive file describes the names and contents of constituent files, providing a way to transport a group of files from one system to another, and to store several files on one tape while retaining their names.

When extracting or listing files, the files are treated as regular expressions, using almost the same syntax as the shell. The shell matches each substring between backslashes separately, while cmtar matches the entire string at once, so some anomalies occur: for example, * or ? can match a /. To specify a regular expression as an argument to cmtar, quote it so the shell will not expand it.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

cmtar provides for storing many SFS or CMFS files into a single archive, which is a file that describes the names and contents of its constituent files. An archive file is the only way to store several files on one tape and retain their names. (A magnetic tape can store several files in sequence, but it stores no names for themjust relative position on the tape). cmtar creates new archives, adds files to an existing archive, lists the files stored in an archive, and extracts files from an archive.

An archive can be stored in a file on the control processor or on an I/O device such as a magnetic tape, floppy, cartridge, or disk; sent over a network; or piped to another program. Piping one cmtar command to another is an easy way to copy a directory's contents from one disk to another while preserving the

dates, modes, owners, and link-structure of all the files therein.

Creating an archive is useful for packaging a set of files to move them to another CM system, as well as for making backup copies of a file system. cmtar has special features for making incremental and full dumps of all the files in a file system. (If a backup file was saved from a point-to-point DataVault and is to be restored onto a multidrop DataVault, use cmtar to write the file to the multidrop DataVault and then convert it using cmdd.)

The first argument to cmtar must include exactly one command, and it may also include zero or more options. The command specifies whether to create a new archive (-c), compare an archive with files in the file system (-d), add files to an archive (-r, -u), list files in an archive (-t), extract files from an archive (-x), concatenate several archives together into one larger archive (-A), or delete specified files from an archive (-D).

The next arguments to cmtar may be one or more options. Since the options control the behavior of the commands, some of the options only make sense with certain commands. The options can be specified in either of two ways. The first method is as described above under Syntax": options that don't take an argument may be specified either with the command or individually preceded by a hyphen, while options that do take an argument are specified individually, preceded by a hyphen and immediately followed by the argument. The second method is that the option letters are specified with the command with no preceding hyphen, and the options' arguments, if any, follow in the same order as the options. The following two commands, which both accomplish the same thing, illustrate both methods:

```
cmtar -c -v -b 20 -f /dev/rmt0

cmtar cvbf 20 /dev/rmt0
```

The last argument to cmtar is files: a list of the files and directories on which cmtar is to operate. For example, they are the files cmtar is to place in a new archive or extract from an existing archive. If a directory is named, cmtar recursively operates on the directory, its files, and its subdirectories. If files is not supplied, the default behavior depends on which command was used. Some commands have no default and report an error if files are not specified. The files on which cmtar is to operate can also be read from a file; see the -T option.

If a full pathname is specified when creating an archive, it is written to the archive without the hostname and initial backslash (/) so the files can later be read into a place other than the original location, and a warning is printed. If files are extracted from an archive that contains full pathnames, they are extracted relative to the current directory and a warning message is printed.

When reading an archive, cmtar continues after finding an error.

The command cmtar is derived from the GNU tar program, which was written by John Gilmore and modified by many people, and whose GNU enhancements were written by Jay Fenlason.

RESTRICTIONS

As in tar, a bug in the Bourne shell usually causes an extra newline to be written to the standard error when remote archives are used.

As in tar, a bug in dd prevents turning off the x+y records in/out messages on the standard error when dd is used to reblock or transport an archive.

SEE ALSO

cmdd
cmfind
cmdump

cmtar (1)

cmrestore
ar(1)
tar(5)
rsh(1)
compress(1)

NAME

cmtruncate - Truncates a CMFS file.

SYNTAX

cmtruncate *path length*

ARGUMENTS

path The path of the file to truncate.

length The length in bytes to which to truncate the file; if *length* is greater than the file's current length, the file is extended to *length* bytes.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

The command **cmtruncate** changes the size of the file named by path to *length* bytes. If the file previously was larger than *length* bytes, the extra data is lost. If the file was previously smaller than *length* bytes, the file is extended to *length* bytes. This command allows you to extend files by pre-allocating contiguous physical disk blocks for efficient writing.

SEE ALSO

CMFS_serial_[f]truncate_file(3)
cmptruncate

copyfromdv (1)

NAME

copyfromdv - Copies a CMFS file to an SFS file system or other UNIX file system.

SYNTAX

copyfromdv [-a] [-i] [-p] [-r] sourcepath destpath

ARGUMENTS

-a	Appends the file named by sourcepath to the file named by destpath rather than overwriting the existing contents. It is an error if the destination file does not already exist.
-i	Interactively prompt for a yes or no response if an existing file will be overwritten. The prompt is the file's name and a question mark. If the response is y, the copy takes place and the existing file is overwritten; any other response prevents the copy.
-p	Preserve the modification times and modes of the file copied.
-r	Recursively copy the contents of directories, files, subdirectories, and the subdirectories' contents that are supplied as the first argument, files, to the directory supplied as the second argument. The second argument cannot be a file.
sourcepath	The file, in a CMFS file system, to be copied. Multiple files may be specified.
destpath	The name of the copy, which is a file in an SFS or other UNIX file system. If sourcepath specifies multiple files, destpath is a directory in which to place the copies of the files.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

The command copyfromdv copies a file residing in the CMFS file system into an SFS or other UNIX file system.

sourcepath names the CMFS file. destpath names the copy, which is a file in an SFS or other UNIX file system.

SEE ALSO

cmcp
copytodv

NAME

copytodv - Copies an SFS file or other UNIX-system file to a CMFS file system.

SYNTAX

copytodv [-a] [-i] [-p] [-r] sourcepath destpath

ARGUMENTS

-a	Appends the file named by sourcepath to the file named by destpath rather than overwriting the existing contents. It is an error if the destination file does not already exist.
-i	Interactively prompt for a yes or no response if an existing file will be overwritten. The prompt is the file's name and a question mark. If the response is y, the copy takes place and the existing file is overwritten; any other response prevents the copy.
-p	Preserve the modification times and modes of the file copied.
-r	Recursively copy the contents of directoriesfiles, subdirectories, and the subdirectories' contentsthat are supplied as the first argument, files, to the directory supplied as the second argument. The second argument cannot be a file.
sourcepath	The file, in a UNIX file system, that is to be copied. Multiple files may be specified.
destpath	The name of the copy, which is a file in the CM file system. If sourcepath specifies multiple files, destpath is a directory in which to place the copies of the files.

WHERE EXECUTED

Control processor
VMEIO host computer
CM-IOP

DESCRIPTION

The command copytodv copies a file residing in an SFS file system or other UNIX file system to a file residing in a CMFS file system.

sourcepath names the file in the SFS or UNIX file system. destpath names the copy, which is a file in a CMFS file system.

The -a append option is used to join several files in a SFS or UNIX file system into a single file in the CMFS file system. For example, the SFS or UNIX file may span several tapes.

The command copytodv extends the destination file to the final size before writing it. If copytodv -a is interrupted while copying, the recorded length for the file is the original length plus the source file's length. cmtruncate can be used to truncate the file to its former length.

SEE ALSO

copytodv
cmcp
cmtruncate

NAME

ibmtape - Imports/exports IBM datasets and handles label reading, writing, and verification.

SYNTAX

ibmtape {**-fromdv** | **-todv**} **if=***inputfile* **of=***outputfile* [**-vmeio**]
label={ **isl** | **ibl** | **inl** | **osl** | **onl** | **obl** | **osl,wsl** | **osl, wnl** }
vsn=*list-of-vol-serial-nos* **dsn=***list-of-dataset-names* [**bs=***buffersize*] **rformat=***fmt*:*blen*:*rlen*
[**stackers=***n*] [**-leds**] [**-compress**] [**-a**]

ARGUMENTS

{**-fromdv** | **-todv**}

These switches indicate the direction of motion of the data. If **-todv** is specified, it is assumed that the input file is a StorageTek tape unit and the output file resides on the DataVault or SDA (depending on the setting of **CMFS_PATHTYPE**); if **-fromdv** is specified, the reverse is true. One and only one of these switches must be specified.

if=*inputfile* Specifies the name of the input file. If a dataset is being written, this specifies the name of the file that is being read; if a dataset is being read from tape, this specifies the name of the device to read it from (usually **/dev/rstc0** through **/dev/rstc3**). Whether the file is being moved to or from a CM file system must be specified by the **-todv** or **fromdv** switch. When *inputfile* is a CMFS file, it can be specified using a full pathname (i.e., *hostname*:*filename*) or, if **DVWD** and/or **DVHOSTNAME** are set appropriately, a relative pathname.

of=*outputfile* Specifies the name of the output file. See **if=**.

-vmeio When executed on a CM-IOP, this switch causes **ibmtape** to use the VMEIO hardware in the CM-IOP to transfer the data to or from the CMFS file system, instead of sending it over the Ethernet. Note that when **ibmtape** is executed on an ITS system, this switch has no effect (the ITS automatically uses the fastest possible data transfer route).

label={ **isl** | **ibl** | **inl** | **osl** | **onl** | **obl** | **osl,wsl** | **osl, wnl** }

Specifies the type of label processing to process against the given input or output dataset. This option applies to the tape being read or written on the StorageTek tape unit.

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.
osl, wsl Replace the existing standard label with the given standard label, rather than using it simply for verification.
osl, wnl Replace the existing standard label with the given non-labelled data, rather than using it simply for verification.

vsn=*list of volume serial numbers*

Specifies the six-digit 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, and extra VSNs are ignored.

dsn=*list of dataset names*

Specifies the name(s) of the current dataset(s), separated by commas. These are matched against the label of each input or output tape, in turn. Concatenating multiple datasets is supported only on input tapes.

bs=*buffersize*

Specifies the size of the buffer to allocate in VMEIO host computer or CP main memory.

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

Specifies the format of the data on the tape. If this option is omitted for input standard label

tapes, the formatting information is extracted from the label and used. **format** must be provided for output data. *fmt* specifies the format of the records on the tape, and must be one of **f, fb, v, vb, vbs,** or **u**. *blen* specifies the length of each physical tape block. *rlen* specifies the length of each logical record on the tape (**0** for variable-format tape). The record length must divide evenly into the block length, and the block length must divide evenly into the buffer size (that is, *bs* >= *blen* >= *rlen*, and *bs*/*blen* = 0, and *blen*/*rlen* = 0). The *fmt* fields are:

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

stackers=*n*

Causes the use of up to four input stackers. This allows the operator to overlap the time needed to rewind and unload one tape that has finished processing with the loading and reading or writing of a subsequent tape in the dataset. It also allows more than 10 tapes to be loaded at one time (each stacker on a StorageTek typically holds 10 tapes).

-leds Displays the volume serial number of the tape currently being processed on the status display of the StorageTek drive.

-compress

Enables use of the ICRC (Improved Cartridge Recording Capability) hardware on the StorageTek drives. This switch is only needed for output tapes; compressed tapes are uncompressed automatically on input and require no special processing. If compressed tapes are loaded on a drive that does not have the compression hardware, the operator is alerted and the tapes are rejected. New compressed tapes are labelled as such and cannot be read by tape units that do not have the ICRC hardware.

-a Specifies that the dataset is to be appended to the destination file.

WHERE EXECUTED

CP
VMEIO host computer
CM-IOP

DESCRIPTION

ibmtape allows the import and export of IBM datasets on IBM Standard Label 3480-type tapes via a StorageTek 4980 tape drive connected to a CM-IOP system or an ITS system. The CM-IOP acts as a gateway between the SCSI interconnect used by the StorageTek drives and the CMIO bus that connects the CM-IOP to a device that hosts a CMFS file system. The ITS acts as a gateway between the SCSI interconnect used by the StorageTek drives and the CM-5 networks that connects the ITS to a Scaleable Disk Array. **ibmtape** allows the operator to optimize the throughput between the StorageTek and the file system through buffering capabilities of either the VMEIO hardware in the CM-IOP or the ITS hardware. **ibmtape** also performs the standard tasks of label reading, writing, and verification.

OVERVIEW: FILE FORMATS

Under UNIX, a file has no format -- it is a raw sequence of bytes existing on a disk or tape, or other media. An application is responsible for intepreting a file in a way appropriate for it. For instance, a text editor interprets a file as sequences of bytes terminated by newlines (carriage returns), and a database program might interpret a file as a series of fixed-length records. On an IBM system, however, files are treated differently. The IBM operating system (typically MVS) handles most of the user's file processing needs. It allows the user to select from a variety of common file formats and allows a user's application to read and write these files without requiring a lot of file-format-specific code.

Definition: a dataset name (dsn) consists of a sequence of characters interspersed with the period character (.), roughly equivalent to a UNIX filename.

An example of a dataset name is "PAYROLL.JULY.SOFTWARE". Dataset names have a maximum length of 44 characters. The names imply a hierarchy. For instance, the above dataset is really a subset of a larger dataset called "PAYROLL.JULY" which is, in turn, a subset of a dataset called "PAYROLL". Datasets consist of records.

Definition: a record is a sequence of bytes from a dataset. It is typically the smallest unit of addressable data in a dataset.

Records come in two types in the IBM world: fixed and variable. Fixed records are all of the same length. These are the most common type and are used widely in databases. Under UNIX, a particular record in a file like this would be extracted by multiplying the record length by the number of the desired record and doing a seek operation. On an IBM, however, the user merely issues a read call with the particular record number.

Variable length records are more complicated. In datasets of this type, the records may be of arbitrary length. Under UNIX, a user would have to scan an entire file to locate a particular record. On an IBM, pointers to specific records are kept along with the file.

OVERVIEW: TAPE FORMATS

As with any form of physical media, tapes impose their own structure upon the data that is written on them. To a UNIX user, tapes are visible simply as files in the filesystem that, when read, present a sequence of bytes followed by an "end of file" condition. To an IBM system user, a tape is an integral extension of the file system. Datasets that have moved to tape have not actually left the file system, *per se*. Rather, they have been entered into a catalog and are archived. The operating system maintains this catalog and is capable of retrieving the dataset when needed (usually by asking a human to put the tape in the drive).

Interestingly enough, when handling tapes more of the structure of the process is visible to the IBM user than to the UNIX user. Here are a few terms used by both the IBM and the UNIX community:

Definition: a tape file is a sequence of records (IBM) or bytes (UNIX) on a tape, terminated by a file mark or tape mark. A tape mark is simply an indicator written on the media by the drive (and unreadable by the user) that indicates an "end of file" condition. There are two other special marks on the tape: the BOT marker and the EOT marker. These are physical indicators on the tape designating the beginning of the tape and the end of the tape.

A tape file should not be confused with an actual data file. Under UNIX, a whole directory structure can be archived into one tape file using the **tar** program, and individual UNIX files can be extracted later. On an IBM, one dataset can span multiple tapes (and hence multiple tape files), or it may occupy only one tape file.

The tape mark is "invisible" to the applications programmer. It is merely an indication to the user that there may be no more data beyond this point. Multiple tape files may be storedon tape by separating them by tapemarks; two tape marks are typically written to indicate the end of the last file on the tape. Tape drives may also position the tape at the granularity of a file. Special hardware exists in the drive to "fast-forward" a tape past a tape mark, leaving the tape ready to read the next file on the tape.

The only indicators on a tape that must be obeyed are the BOT and EOT markers. The BOT marker informs a tape drive that it has completely rewound the tape. The EOT marker lets the applications programmer know that the physical end of tape has been reached. An application is notified of an EOT condition on reading or writing. More data may then be written, but the consequences could be dire --

on oder round tapes, the tape would frequently fall off the reel and require the operator to reload the tape.

The file is not the smallest granularity with which data may be written to a tape, however. Tape drives partition data on the tape into blocks or records.

> *Definition: A tape block is the smallest unit of data that may be addressed on a tape. Tape blocks are unfortunately sometimes referred to as tape records, in which case the records of the dataset on the tape are called logical records and the tape blocks themselves are referred to as physical records.*

Like a tape file, there is not a one-to-one mapping between records in a dataset and blocks on a tape (although there could be). Tape blocks are usually much larger, to improve performance. If they are larger than a record in a dataset, the records in the dataset are packed into the tape block in some logical fashion. A drive may position itself on a specific block on the tape by performing FSR (forward space record) or BSR (backward space record) operations. In addition, mot tape drive units support a high-speed access method called fixed block mode, which allows collections of similar-sized records to be extracted at once. The normal mode of operation is variable block mode.

All of the preceding terms describe attributes that are common to all tapes, whether or not they originate from a UNIX system or an IBM system. They are also common to all tape styles, such as 9-track ("reel-to-reel"), QIC (cartridge), 3480 (IBM "square" tapes), or Exabyte (8mm video). On IBM systems, however, there are a group of conventions for how the user's data is laid out on the tape, known as *tape formats*, and a group of conventions for placing descriptive data on each tape, known as *label formats*.

> *Definition: a tape label is an electronic version of the label found on the outside of the tape, except that it is written on the first few recors of the tape itself. The label usually contains data descriptive of the contents of the tape, such as the dataset name, the volume number, the owner of the tape, the number of the records, etc. There are several formats for labels, the most common of which is the IBM standard label. There is also a label format known as an ANSI label. IBM also supports a tape format called a non-labelled tape, which is not to be confused with a unlabelled tape (a tape that has no label).*

Tape labels serve several different purposes. First, they support the IBM concept of extending the file system out to a collection of tapes, and allow the computer to verify the contents of a particular set of tapes. Second, they provide an element of protection. Jobs that run on an IBM must specify the dataset name and volume serial numbers of the tapesthey want to work with. If the wrong volume is inadvertently mounted on the drive, or the volumes of the dataset are mounted out of order, the operator will be notified and requested to mount the correct volume. Third, they describe the contents of the tape and givethe information necessary to reconstruct the dataset on disk as it has been written to tape (potentially across many separate volumes).

A label on a tape is essentially a small tape file that appears before any tape files containing data. This file typically contains anywhere from one to three 80-character records. If a program such as **dd**(1) is used to read the tape, this file can be extracted and examined by the user. However, the information contained in the label is meant to be read by a program and not a human. Most tape label formats also include trailer labels, which aresimilar to the labels found on the front of the tape, except that they contain information such as checksums, block counts, and anindication as to whether or not there are more volumes to be read in a paticular dataset.

Many of the options to **ibmtape** are used to perform the verification of the tape labels on the various volumes of the datasets. This includes the **label, dsn, vsn,** and rformat options. The most important of these options is the **label** option. This man page describes only processing tapes with standard labels, as

they are by far the most common type.

label=isl This format indicates that each volume of the input dataset is preceded by an IBM standard label. The data read fromthis label is compared against the values given in the **dsn** and **vsn** options. If any of these fields do not match, the operator is alerted and given the option of cancelling processing of the mismatched tape or allowing processing to continue. If no **rformat** option is specified, the data format, block size, and record size in the label are used during processing. When the end of volume on each tape is reached, the number of blocks read from the tape is compared with the block count given in the label. If these counts do not match, the operator is alerted and given the option of aborting or continuing processing. If the tape label indicates that the tape was written in compressed format, and the hardware the tape is loaded on is not capable of reading compressed media, the tape is rejected.

label=osl This format is used when producing output tapes that already have a valid label written on them. Many data centers initialize the label on new tapes with the volume number the tape has been assigned in the center's library, and the name of the dataset that will reside on it. When a volume is processed, the dataset name and volume numberfrom the label are compared against the dataset name and volume number from the command line. If these do not match, the operator is alerted and given the option of aborting or proceeding. If the operator chooses to proceed, the tape will be written, but the label will not be altered.

label=osl,wsl This form of the label option is used when producing output tapes and initializing, rewriting, or overriding the label data on a tape. When a volume is processed, it is first checked forthe presence or absence of a label. If no label is found, one is written using the information provided on the command line. If a label is found, the operator is alerted that allowing the job to continue will overwrite the label on the tape. If the operator chooses to proceed, the old label data is replaced with the information given on the command line, and the dataset is written to the tape.

IBM-FORMAT TAPE-LABEL PROCESSING

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, **ibmtape** 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, **ibmtape** checks for the existence of a corresponding control file. If the file exists, **ibmtape** 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, **ibmtape** fills in the placeholders with the correct descriptor words extracted from the control file.

If no control file is found, **ibmtape** 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, **ibmtape** 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, **ibmtape** 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 **ibmtape**. 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, **ibmtape** automatically sets fixed-block mode to the size specified in the block length field specified in the **HDR2** label.

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

To prevent **ibmtape** 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).

PERFORMANCE CONSIDERATIONS

At full speed, the StorageTek unit is able to read data from tape and provide it to the SCSI host at 2MB/sec. Data from the tape is read into the buffer memory of the VMEIO board in the CM-IOP system. This buffer memory is available to the Ciprico SCSI board via direct DMA across the Sun backplane in the CM-IOP system -- thus, no intervention by the operating system is necessary to move the data. Oncethe data has been deposited in the buffer memory of the VMEIO, it can be written to a CMFS file system via the CMIO bus at 15-20 MB/sec.

The principal reason why performance tuning may be necessary to get close to 2MB/sec isthat the StorageTek is a streaming tape unit. This means that the tape system is capable of reading and writing data as fast as the tape passes beneath the tape heads. However, if the system runs out of data to write, it will not stop motion immediately, and it will be forced to rewind the tape past the point that it stopped I/O and bring the tape up to speed again before resuming data transfer. If the StorageTek system is "starved" for data when writing a tape, or if the user's peripheral is not able to receive data as fast as the StorageTek provides it, more and more time will be spent repositioning the tape.

To solve this problem, the **ibmtape** command employs a system known as double buffering. When the user specifies a buffer size with the **bs** option, two of these buffers are actually allocated in the VMEIO buffer memory. When reading from tape, data is read into one of the two buffers. When that buffer has been filled, the tape unit starts depositing data into the second buffer, and the first buffer is used to write to the CMFS file system. As the StorageTek completes transfers, these buffers are repeatedly switched, ensuring that the tape is kept more or less continually in motion. A similar process is employed for writing, with the CMFS file system filling one buffer with data while theStorageTek is removing data from the other.

Each VMEIO has 32 MB of on-board buffer memory available to the user. A small amount of this memory i reserved for the system (approximately 48K) leaving in the range of 31 MB of memory available for the application or for use by **ibmtape**. In order to achieve maximum performance fromt he CM-IOP system, it is usually wise to setthe **bs** option as large as possible. The **bs** value must be a constant multiple of the block length from the tape (as specified in the tape label or on the command line via the **rformat** argument). Keeping in mind that two buffers must be allocated for the double buffering algorithm, the **bs** option can be no larger than about 15 MB. Maximum performance is usually reached with buffer sizes in the range of 5-10 MB.

Performance of the StorageTek system may also be improved by carefully selecting the data format to

be used on the tape. Not all data formats are equal. In general, the various fixed-record formats (**f, fb**) can be processed much faster than the variable formats (**v, vb, vbs,** and **u**). Due to the fact that the UNIX file system does not support structured files, additional processing is performed on variable-format files as they are read from the tape system. Variable-format fles have information contained in the data stream that indicates the start and end of records. Most application programs will want to remove or skip this data when processing the file, so the **ibmtape** command creates an index into the dataset as it is read, called a control file. This contorl file is created with the smae name as the dataset with an extension of **.ctl**. The process of creating this file can be quite time-conuming since it requires scanning the entire input buffer before it is written to disk. Double buffering provides extra time for **ibmtape** to scan the buffer, but extremely short variable-length records can still cause a considerable performance degradation. If a variable-length data format is being used to store primarily fixed-length data, it would be advantageous to switch to a fixed or unblocked format if at all possible.

Another factor that can degrade performance when reading from the StorageTek system is the presence of short blocks in the data stream on the tape. **ibmtape** attempts to use fixed block mode when reading data fromthe tape, assuming that this will achieve the maximum transfer rate possible. If, during the read option, the StorageTek encounters a block on the tape that is shorter than the given block size, it will be unable to continue the read, and returns a count of the number of blocks actually read to **ibmtape. ibmtape** must then take the driv eout of fixed block mode, back the tape up one record, reread the short record, put the drive back into fixed block mode, and continue with the read. This can cause a considerable performance penalty if it must be done frequently while reading a particular volume. Since the block size for fixed block mode is specified by the block size in the **rformat** option (or in the tape label), it is important that this value accurately represent the blocking of the data on the tape, and not be arbitrary or inaccurate.

It is important to note that even fixed format tapes are not exempt from having short blocks written on them. The term "fixed" refers to the length of the user records in the data, not to the length of the blocks on the tape. So, it is possible to have a tape with an **rformat** of **fb:8000:80** record length of 80 bytes, block size of 8000 bytes, 100 records per tape block), where every other tape block has 99 records on it. Such a data layout can considerably degrade performance. Unfortunately, considerations like this are largely governed byt he system that produced the tapes being read.

EXAMPLES

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
cmrestore
CMFS_PATHTYPE(7)

Appendix B

CM-5 CMFS Calls

NAME

CMFS-access - - Determines the accessibility of a file.

C SYNTAX

```
#include <
n = CMFS_access(path, mode)
int n, mode;
char *path;
INTEGER n, mode STRING path
```

ARGUMENTS

path	A pointer to the pathname of the file whose permissions to check.
mode	The mode to verify.

RETURN VALUE

0	Indicates successful completion.
-1	Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_access checks the file named by path for accessibility according to mode, which is an inclusive OR of the following bits:

CMFS_R_OK	test for read permission
CMFS_W_OK	test for write permission
CMFS_X_OK	test for execute or search permission
CMFS_F_OK	test whether the directories leading to the file can be searched and the file exists.

Only access bits are checked. A directory may be indicated as writable by CMFS_access, but an attempt to open it for writing will fail (although files may be created there).

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_unlink fails if any of the following are true:

CMFS_EACCES
: Search permission is denied for any component of the path name.

CMFS_ENOTDIR
: A component of the path prefix is not a directory.

CMFS_EINVAL
: path contains a character with a high-order bit set.

CMFS_ENAMETOOLONG
: path exceeded 255 characters in length.

CMFS_ENOENT
: The file referred to by path does not exist.

CMFS_EFAULT
: path points to an invalid address.

CMFS_EIO An I/O error occurred while reading from or writing to the file system.

CMFS_EROFS
: The named file resides on a read-only file system and write access was requested.

CMFS_access(3)

SEE ALSO

CMFS_chdir
CMFS_[f]chmod
CMFS_[f]stat
CMFS_[f]chown

NAME

CMFS_chdir - Changes the current working directory.

C SYNTAX

```
n = CMFS_chdir(path)
char *path;
int n;
```

FORTRAN SYNTAX

```
n = CMFS_CHDIR(path)
INTEGER n
STRING path
```

ARGUMENTS

path	The directory to become the new current working directory.

RETURN VALUE

0	Indicates successful completion.
-1	Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_chdir causes the directory named by path to become the current working directory, the starting point for path names not beginning with the slash (/).

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_chdir fails--the current working directory remains unchanged--if any of the following are true:

CMFS_ENOTDIR
: A component of the pathname is not a directory.

CMFS_EINVAL
: The pathname contains a character with the high-order bit set.

CMFS_ENAMETOOLONG
: Path exceeds 255 characters in length.

CMFS_ENOENT
: The named directory does not exist.

CMFS_EACCES
: Search permission is denied for any component of the path name.

CMFS_EIO An I/O error occurred while reading from or writing to the file system.

CMFS_ESTALE
: The file handle given in the argument was invalid. The file referred to by that file handle no longer exists or has been revoked.

CMFS_ETIMEDOUT
: A connect" request or remote file operation failed because the connected party did not properly respond after a period of time that is dependent on the communications protocol.

SEE ALSO

DVWD (environment variable)

NAME

CMFS_chmod - Changes the mode of a file.

C SYNTAX

```
#include <sys/types.h>
#include <cm/cm_stat.h>

n = CMFS_chmod (path, mode)
char *path;
int mode, n;

n = CMFS_fchmod (fd, mode)
int n, fd, mode;
```

FORTRAN SYNTAX

```
n = CMFS_CHMOD (path, mode)
INTEGER n, mode
STRING path

n = CMFS_FCHMOD (fd, mode)
INTEGER n, fd, mode
```

ARGUMENTS

path	Character string. The pathname of the file whose mode is to be changed.
mode	Integer. The mode to assign to the file named by path or referenced by fd.
fd	Integer. File descriptor of the file whose mode is to be changed.

RETURN VALUE

0	Indicates successful completion.
-1	Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_[f]chmod changes the mode of the file whose name is given by path or referenced by the file descriptor fd to mode. Modes are constructed by ORing together some combination of the following:

```
CMFS_S_ISUID   04000   Set user ID on execution.
CMFS_S_ISGID   02000   Set group ID on execution.
CMFS_S_IREAD   00400   Read by owner.
CMFS_S_IWRITE  00200   Write by owner.
CMFS_S_IEXEC   00100   Execute (search on directory) by owner.
               00070   Read, write, execute (search) by group.
               00007   Read, write, execute (search) by others.
```

These bit patterns are defined in /usr/include/cm/cm_stat.h.

In order to change the mode of a file, the effective user ID of the process must either match the owner of the file or be superuser.

RESTRICTIONS

Permission checking is enabled only if the appropriate file server is set to check permissions (the default).

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_chmod fails--the file mode remains unchanged--if any of the following are true:

CMFS_ENOTDIR
: A component of the path prefix of path is not a directory.

CMFS_ENAMETOOLONG
: The length of a component of path exceeds 255 characters in length.

CMFS_ENOENT
: The file referred to by path does not exist.

CMFS_EACCES
: Search permission is denied for a component of the path prefix of path.

CMFS_EPERM
: The effective user ID does not match the owner of the file and the effective user ID is not the superuser.

CMFS_fchmod fails--the file mode remains unchanged--if any of the following are true:

CMFS_EBADFD
: The file descriptor is not valid.

CMFS_EPERM
: The effective user ID does not match the owner of the file and the effective user ID is not the superuser.

CMFS_EIO An I/O error occurred while reading from or writing to the file system.

SEE ALSO

CMFS_chown
CMFS_stat
cmchgrp
cmchown
cmchmod

NAME

CMFS_chown - Changes the owner and group of a file.

C SYNTAX

```
n = CMFS_chown (path, owner, group)
char *path;
int n, owner, group;

n = CMFS_fchown (fd, owner, group)
int n, fd, owner, group;
```

FORTRAN SYNTAX

```
n = CMFS_CHOWN (path, owner, group)
INTEGER n, owner, group
STRING path

n = CMFS_FCHOWN (fd, owner, group)
INTEGER n, fd, owner, group
```

ARGUMENTS

path	Character string. The pathname of the file whose owner and group is to be changed.
owner	Integer. The owner to assign to the file named by path or referenced by fd.
group	Integer. The group to assign to the file named by path or referenced by fd.
fd	Integer. File descriptor of the file whose owner and group is to be changed.

RETURN VALUE

0	Indicates successful completion.
-1	Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_[f]chown changes the owner and group of the file named by path or referenced by fd to owner and group, respectively. Only the superuser can change the owner of the file. Both the owner of the file and the superuser can change the group of the file.

If owner or group is specified as -1, the corresponding ID of the file is not changed.

RESTRICTIONS

Permission checking is enabled only if the appropriate file server is set to check permissions (the default).

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_chown fails--the file will be unchanged--if any of the following are true:

CMFS_ENOTDIR
: A component of the path prefix of path is not a directory.

CMFS_ENAMETOOLONG
: The length of a component of path exceeds 255 characters in length.

CMFS_ENOENT
: The file referred to by path does not exist.

CMFS_EACCES

Search permission is denied for a component of the path prefix of path.

CMFS_EPERM
The user ID specified by owner is not the current owner ID of the file, or the group ID specified by group is not the current group ID of the file and the effective user ID is not the superuser.

CMFS_EINVAL
Sockets are not supported.

CMFS_fchown fails--the file mode remains unchanged--if any of the following are true:

CMFS_EBADFD
The file descriptor is not valid.

CMFS_EPERM
The user ID specified by owner is not the current owner ID of the file, or the group ID specified by group is not the current group ID of the file, and the effective user ID is not the superuser.

CMFS_EIO An I/O error occurred while reading from or writing to the file system.

SEE ALSO

CMFS_chmod
CMFS_stat
cmchmod
cmchgrp
cmchown

NAME

CMFS_close - Closes a file or socket.

C SYNTAX

```
n = CMFS_close(fd)
int n, fd;
```

FORTRAN SYNTAX

```
n = CMFS_CLOSE (fd)
INTEGER n, fd
```

LISP SYNTAX

```
(CMFS:close fd)
```

ARGUMENTS

fd	File descriptor returned by a previous call to CMFS_creat or CMFS_open; or socket descriptor returned by a previous call to CMFS_socket or CMFS_accept.

RETURN VALUE

0	Indicates successful completion.
-1	Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_close closes the object represented by fd, deleting the descriptor from the per-process object reference table. Using fd in further I/O calls is no longer valid. If the close operation is the last reference to the underlying object, the object is deactivated. On the last close of a file, the current seek pointer associated with the file is lost; when a socket is closed, associated naming information and queued data are discarded.

A close of all of a process's descriptors is automatic on exit, but since there is a limit (29) on the number of active file descriptors per process, CMFS_close is necessary for programs using many file descriptors. Forking is not supported; if either of two processes closes an object, the other process can no longer use its file descriptor.

When doing buffered I/O, CMFS_close flushes the buffer, ensuring that all buffered writes are actually written to disk.

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_close fails if:

CMFS_EBADF	fd is not an active descriptor.

SEE ALSO

CMFS_accept
CMFS_close_all_files
CMFS_connect
CMFS_open
CMFS_shutdown
CMFS_socket
CMFS_write_file

NAME

CMFS_close_all_files - Close all files and break the TCP connections.

C SYNTAX

```
void
CMFS_close_all_files

n = CMFS_close_files_on_server(hostname)
int n;
char *hostname;
```

FORTRAN SYNTAX

```
SUBROUTINE CMFS_CLOSE_ALL_FILES

n = CMFS_CLOSE_FILES_ON_SERVER (hostname)
INTEGER n
STRING hostname
```

ARGUMENTS

hostname Character string.

RETURN VALUE

CMFS_close_all_files

None.

CMFS_close_files_on_server

n The number of files closed.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_close_all_files closes all open files and breaks their TCP connections.
CMFS_close_files_on_server closes all files residing on the device having the specified hostname and breaks the TCP connection to that device. If hostname is NULL, the default hostname is used.

ERRORS

For CMFS_close_files_on_server, if an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_close_all_files always succeeds.

SEE ALSO

CMFS_close
CMFS_open

NAME

Directory Operations - Open, read, seek, tell the location in, and close directories.

C SYNTAX

```
#include <cm/cm_dir.h>

dirp = *CMFS_opendir(pathname)
char *pathname;
CMDIR *dirp;

entp = CMFS_readdir(dirp)
CMDIR *dirp;
struct cm_direct *entp;

n = CMFS_telldir(dirp)
long n;
CMDIR *dirp;

void CMFS_seekdir(dirp, loc)
CMDIR *dirp;
long loc;

n = CMFS_closedir(dirp)
CMDIR *dirp;
int n;
```

FORTRAN SYNTAX

Not supported.

ARGUMENTS

pathname	Character string. The pathname of the directory to be opened.
dirp	A pointer to identify the directory stream. dirp is returned by CMFS_opendir and used in the other directory operations.
loc	Integer. Returned by an earlier call to CMFS_telldir, this argument to CMFS_seekdir sets the position of the next CMFS_readdir. A loc of 0 sets the position to the beginning of the directory stream.
entp	A pointer to the next directory entry. entp is returned by CMFS_readdir

RETURN VALUES

CMFS_opendir

dirp	Pointer to identify the directory stream.
NULL	Indicates the directory cannot be accessed or insufficient memory is available to open it.

CMFS_readdir

entp	Pointer to the next directory entry.
NULL	Indicates the end of the directory has been reached or an invalid seekdir operation has been detected.

CMFS_telldir

n	Current location associated with the directory stream.

CMFS_seekdir

CMFS_closedir

0	Indicates successful close
-1	Indicates an error; CMFS_errno is set.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_opendir opens the directory named by pathname and associates a directory stream with it. CMFS_opendir returns a pointer to identify the directory stream in subsequent operations. The pointer NULL is returned if the specified filename can not be accessed, or if insufficient memory is available to open the directory file.

CMFS_readdir returns a pointer to the next directory entry. It returns NULL or nil upon reaching the end of the directory or upon detecting an invalid seek operation. CMFS_readdir uses the UNIX get-directories system call to read directories.

CMFS_telldir returns the current location associated with the named directory stream. Values returned by CMFS_telldir are good only for the lifetime of the DIR pointer from which they are derived. If the directory is closed and then reopened, the CMFS_telldir value may be invalidated due to undetected directory compaction.

CMFS_seekdir sets the position of the next CMFS_readdir operation on the directory stream. Only values returned by CMFS_telldir should be used with CMFS_seekdir.

CMFS_closedir closes the named directory stream. If the close is successful, 0 is returned; if the close fails, -1 is returned and CMFS_errno is set. All resources associated with this directory stream are released.

The directory stream that CMFS_opendir associates with pathname contains an entry for each directory residing in pathname. Each directory entry has associated with it four pieces of information. This information is stored in a directory structure having four elements. A pointer to this structure is returned by CMFS_readdir. The four elements are:

cmd_ino	Inode number
cmd_reclen	Directory record length
cmd_namlen	Directory name length
cmd_name	Directory name entry

The content of an element is referenced by pointer -> element-name. For example, if dir_ent_ptr is the pointer returned by CMFS_readdir, the directory name entry is referenced by dir_ent_ptr -> d_name.

EXAMPLE

The following sample code searches a directory for the entry name.

```
len = strlen(name);
dirp = CMFS_opendir(.);
for (dp = CMFS_readdir(dirp); dp != NULL;
    dp = CMFS_readdir(dirp))
if (dp->cmd_namlen == len && !strcmp(dp->cmd_name, name)) {
  CMFS_closedir(dirp)
  return FOUND;
}
CMFS_closedir(dirp);
return NOT_FOUND;
```

NAME

CMFS_creat - Creates or rewrite a file.

C SYNTAX

```
fd = CMFS_creat(path, mode)
char *path;
int fd, mode;
```

FORTRAN SYNTAX

```
fd = CMFS_CREAT(path, mode)
STRING path
INTEGER mode
```

ARGUMENTS

path	Character string. The file to be created, or an existing file to be rewritten.
mode	Integer. The mode to assign to the file. See CMFS_chmod.

RETURN VALUE

fd	A file descriptor, to be used in other operations on the file.
-1	Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_creat either creates a new file or prepares to rewrite an existing file. It returns a non-negative integer called a file descriptor, to be used in other operations on that file. CMFS_creat leaves the file open for writing, even if the mode does not permit writing, and sets the file pointer to the beginning of the file.

The path argument points to the pathname of a file. New files are created with mode mode, as described in CMFS_chmod.

An existing file is truncated to zero length. The mode and owner of the file remain unchanged.

File protection is supported. When a file is created using CMFS_open or CMFS_creat, the owner and group of the file are set to the default values, as described in the UNIX reference manual pages for open(2) and creat(2); the file's mode is explicitly set (see CMFS_chmod). The utility cmls, as well as the library calls CMFS_[f]stat, shows the correct mode, owner, and group associated with a file.

RESTRICTIONS

The name of a CMFS file can be no more than 255 characters.

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_creat fails if any of the following are true:

CMFS_EINVAL
: The argument .contains a character with the high-order bit set.

CMFS_ENOTDIR
: A component of the path prefix is not a directory.

CMFS_EISDIR
: The file is a directory.

CMFS_EMFILE
: There are too many files open.

CMFS_EROFS

The named file resides on a read-only file system.

CMFS_ENXIO
The file is a character-special or block-special file, and the associated device does not exist.

CMFS_EOPNOTSUPP
The file is a socket, which is not implemented.

CMFS_ENAMETOOLONG
Path exceeded 255 characters in length.

CMFS_ENOENT
The named file does not exist.

CMFS_ENFILE
The system file table is full.

CMFS_ENOSPC
The directory in which the entry for the new file is being placed cannot be extended because there is no space left on the file system containing the directory.

CMFS_ENOSPC
There are no free inodes on the file system on which the file is being created.

CMFS_EDQUOT
Quotas are not supported.

CMFS_EIO An I/O error occurred while making the directory entry or allocating the inode.

CMFS_ESTALE
The file handle given in the argument is invalid. The file referred to by that file handle no longer exists or has been revoked.

CMFS_ETIMEDOUT
A connect" request or remote file operation failed because the connected party did not properly respond after a period of time that is dependent on the communications protocol.

SEE ALSO

cmls
CMFS_open
CMFS_chmod
CMFS_chown

NAME

Directory Operations - Open, read, seek, tell the location in, and close directories.

C SYNTAX

```
#include <cm/cm_dir.h>

dirp = *CMFS_opendir(pathname)
char *pathname;
CMDIR *dirp;

entp = CMFS_readdir(dirp)
CMDIR *dirp;
struct cm_direct *entp;

n = CMFS_telldir(dirp)
long n;
CMDIR *dirp;

void CMFS_seekdir(dirp, loc)
CMDIR *dirp;
long loc;

n = CMFS_closedir(dirp)
CMDIR *dirp;
int n;
```

FORTRAN SYNTAX

Not supported.

ARGUMENTS

pathname	Character string. The pathname of the directory to be opened.
dirp	A pointer to identify the directory stream. dirp is returned by CMFS_opendir and used in the other directory operations.
loc	Integer. Returned by an earlier call to CMFS_telldir, this argument to CMFS_seekdir sets the position of the next CMFS_readdir. A loc of 0 sets the position to the beginning of the directory stream.
entp	A pointer to the next directory entry. entp is returned by CMFS_readdir

RETURN VALUES

CMFS_opendir

dirp	Pointer to identify the directory stream.
NULL	Indicates the directory cannot be accessed or insufficient memory is available to open it.

CMFS_readdir

entp	Pointer to the next directory entry.
NULL	Indicates the end of the directory has been reached or an invalid seekdir operation has been detected.

CMFS_telldir

n	Current location associated with the directory stream.

CMFS_seekdir

None.

CMFS_closdir

0	Indicates successful close
-1	Indicates an error; CMFS_errno is set.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_opendir opens the directory named by pathname and associates a directory stream with it. CMFS_opendir returns a pointer to identify the directory stream in subsequent operations. The pointer NULL is returned if the specified filename can not be accessed, or if insufficient memory is available to open the directory file.

CMFS_readdir returns a pointer to the next directory entry. It returns NULL or nil upon reaching the end of the directory or upon detecting an invalid seek operation. CMFS_readdir uses the UNIX get-directories system call to read directories.

CMFS_telldir returns the current location associated with the named directory stream. Values returned by CMFS_telldir are good only for the lifetime of the DIR pointer from which they are derived. If the directory is closed and then reopened, the CMFS_telldir value may be invalidated due to undetected directory compaction.

CMFS_seekdir sets the position of the next CMFS_readdir operation on the directory stream. Only values returned by CMFS_telldir should be used with CMFS_seekdir.

CMFS_closedir closes the named directory stream. If the close is successful, 0 is returned; if the close fails, -1 is returned and CMFS_errno is set. All resources associated with this directory stream are released.

The directory stream that CMFS_opendir associates with pathname contains an entry for each directory residing in pathname. Each directory entry has associated with it four pieces of information. This information is stored in a directory structure having four elements. A pointer to this structure is returned by CMFS_readdir. The four elements are:

cmd_ino	Inode number
cmd_reclen	Directory record length
cmd_namlen	Directory name length
cmd_name	Directory name entry

The content of an element is referenced by pointer -> element-name. For example, if dir_ent_ptr is the pointer returned by CMFS_readdir, the directory name entry is referenced by dir_ent_ptr -> d_name.

EXAMPLE

The following sample code searches a directory for the entry name.

```
len = strlen(name);
dirp = CMFS_opendir(.);
for (dp = CMFS_readdir(dirp); dp != NULL;
    dp = CMFS_readdir(dirp))
if (dp->cmd_namlen == len && !strcmp(dp->cmd_name, name)) {
  CMFS_closedir(dirp)
  return FOUND;
}
CMFS_closedir(dirp);
return NOT_FOUND;
```

NAME

CMFS_errmessage - Stores the error message (associated with the latest CMFS error) in a buffer.

C SYNTAX

```
void
CMFS_errmessage(buffer)
char *buffer;
```

FORTRAN SYNTAX

Not supported.

ARGUMENTS

buffer A pointer to the buffer that will store the error message string.

RETURN VALUE

None

CM STATE CHANGE

None.

DESCRIPTION

CMFS_errmessage places the string corresponding to the number taken from the external variable **CMFS_errno** into a buffer (rather than writing it to standard error, as does **CMFS_perror**).

SEE ALSO

CMFS_perror
CMFS_errno

NAME

CMFS_errno - Stores the error number of the last error that occurred (external variable).

C SYNTAX

```
n = CMFS_errno
extern int n;
```

FORTRAN SYNTAX

Not supported.

```
N = CMFS_ERRNO
INTEGER N
```

DESCRIPTION

When a CMFS call is not successfully completed, the CMFS external variable CMFS_errno and the UNIX external variable errno() are set to a number indicating the cause of the error. If the cause of the error is not unique to a CM system, errno() and CMFS_errno() are set to the same value; if the cause of the error is unique to a CM system, CMFS_errno is set to a value representing the exact error, but errno() is set to a value representing a general I/O error.

If CMFS_errno is set to 128 or less, the error description associated with it matches the UNIX error description given for it in the UNIX reference page intro(2). If CMFS_errno is set to 129 or greater, the error is specific to the CM system and is one of those listed below. (A few of the UNIX error numbers correspond to different error descriptions in different implementations of a UNIX system. CM I/O error numbers and descriptions, for values of 128 or less, match those used in the ULTRIX system, which is the implementation of UNIX running on the DataVault file server computer.)

The system call CMFS_perror is used to print the error message corresponding to the value of CMFS_errno. (Although the UNIX call perror() also prints the correct error message for errors not unique to a CM system, its use is not recommended.) CMFS_perror writes an error message on the standard output file describing the last error encountered, by writing both a user-supplied string and a system-supplied message. See the list below for error numbers and their descriptions. These errors are defined in C in the file <cm/cm_errno.h>.

When calls are successfully completed, CMFS_errno is not cleared so it always contains the value of the last error that occurred.

n =

129 DIAGFL Diagnostic failure.

130 BPHASE Bad phase from file server.

131 SWCHEK Software consistency check.

132 HDWFAL Hardware failure.

133 MEDIAF Disk media failure.

134 MLTECC Multiple-bit ECC error detected.

135 CONFUS DataVault software got confused but is now resetting.

136 CBSPRO CMIO bus protocol error.

137 ILPARM Illegal/invalid parameter in command.

138 PSTEOM Request goes past physical end of media.

139 GARBLD Unknown command blockcommand garbled.

140 RSVDNZ Reserved field(s) not set to zero.

141 BPCKSM Bad PROM checksum.

142 BRCKSM Bad RAM checksum.

143 FNSREV Function not supported in this revision.

144 BADPAR Bad parity on data.

145 DRVFMT Inconsistent drive format.

146 DRVERR Drive error.

147 NODRVA No drives active (cannot initialize drive-related tables).

148 NOTLOD RAM not loaded with viable image.

149 DINAVL Diagnostic not available in current configuration.

150 PIPSFT Data pipeline corrected single bit error(s).

151 BDSPAR Configuration error: incorrect/illegal sparing selection.

152 SCTIME SCSI-related timeout.

153 DATIME Data transfer timeout.

154 COTIME Control transfer timeout.

155 DRVSFT Disk drive recovered from soft error.

156 DRVHRD Disk drive could not recover from error.

157 NAKRNL Command not available from kernel.

158 LOGFUL Log almost full.

159 BUSBSY Bus busy timeout.

160 NAKRCV Target busy timeout.

161 NOARBT No arbiter on bus.

162 TSELTO No response from target select.

163 XCPRCV Bus exception received.

164 NOTSEL No target select received.

165 PCERR DataVault pipe counter mismatch.

179 CMFS_EINVALID_RETURN_COUNT
File server return count was an invalid value.

181 CMFS_ESTRIPE_NOTAVAIL
One or more of the file servers serving the striped CM file system is not running.

182 CMFS_ECMIOBUS_UNREACH
The requested CMIO bus is unreachable from the file server.

183 CMFS_EPROTOCOL_MISMATCH
There is no library support for the feature (for example, disk striping) that the file server is attempting to use, or the file server doesn't support the feature that the application program is trying to use.

184 CMFS_ESYNC_LOST
The synchronization has been lost on the network connection between the CMFS library and the CMFS file server.

185 CMFS_ENO_FILE_SERVER
The file server to which the control processor is trying to connect is not running.

186 CMFS_ERAW_ULTRA
Raw HIPPI attempted on Ultranet.

187 CMFS_EHIPPI_SRC_PARITY
CMHIPPI source board detected a parity error.

188 CMFS_EHIPPI_DST_PARITY
CMHIPPI destination board detected a parity error.

189 CMFS_EHIPPI_IOP_PARITY
CMHIPPI IOP board detected a parity error.

SEE ALSO

CMFS_perror
errno(2) (on Vaxen)
intro(2) (on Suns)

NAME

CMFS_chmod - Changes the mode of a CMFS file.

C SYNTAX

```
#include <sys/types.h>
#include <cm/cm_stat.h>

n = CMFS_chmod (path, mode)
char *path;
int mode, n;

n = CMFS_fchmod (fd, mode)
int n, fd, mode;
```

FORTRAN SYNTAX

```
n = CMFS_CHMOD (path, mode)
INTEGER n, mode
STRING path

n = CMFS_FCHMOD (fd, mode)
INTEGER n, fd, mode
```

ARGUMENTS

path	Character string. The pathname of the file whose mode is to be changed.
mode	Integer. The mode to assign to the file named by path or referenced by fd.
fd	Integer. File descriptor of the file whose mode is to be changed.

RETURN VALUE

0	Indicates successful completion.
-1	Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_[f]chmod changes the mode of the file whose name is given by path or referenced by the file descriptor fd to mode. Modes are constructed by ORing together some combination of the following:

```
CMFS_S_ISUID   04000   Set user ID on execution.
CMFS_S_ISGID   02000   Set group ID on execution.
CMFS_S_IREAD   00400   Read by owner.
CMFS_S_IWRITE  00200   Write by owner.
CMFS_S_IEXEC   00100   Execute (search on directory) by owner.
               00070   Read, write, execute (search) by group.
               00007   Read, write, execute (search) by others.
```

These bit patterns are defined in /usr/include/cm/cm_stat.h.

In order to change the mode of a file, the effective user ID of the process must either match the owner of the file or be superuser.

RESTRICTIONS

Permission checking is enabled only if the appropriate file server is set to check permissions (the default).

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_chmod fails--the file mode remains unchanged--if any of the following are true:

CMFS_ENOTDIR
: A component of the path prefix of path is not a directory.

CMFS_ENAMETOOLONG
: The length of a component of path exceeds 255 characters in length.

CMFS_ENOENT
: The file referred to by path does not exist.

CMFS_EACCES
: Search permission is denied for a component of the path prefix of path.

CMFS_EPERM
: The effective user ID does not match the owner of the file and the effective user ID is not the superuser.

CMFS_fchmod fails--the file mode remains unchanged--if any of the following are true:

CMFS_EBADFD
: The file descriptor is not valid.

CMFS_EPERM
: The effective user ID does not match the owner of the file and the effective user ID is not the superuser.

CMFS_EIO An I/O error occurred while reading from or writing to the file system.

SEE ALSO

CMFS_chown
CMFS_stat
cmchgrp
cmchown
cmchmod

NAME

CMFS_chown - Changes the owner and group of a CMFS file.

C SYNTAX

```
n = CMFS_chown (path, owner, group)
char *path;
int n, owner, group;

n = CMFS_fchown (fd, owner, group)
int n, fd, owner, group;
```

FORTRAN SYNTAX

```
n = CMFS_CHOWN (path, owner, group)
INTEGER n, owner, group
STRING path

n = CMFS_FCHOWN (fd, owner, group)
INTEGER n, fd, owner, group
```

ARGUMENTS

path	Character string. The pathname of the file whose owner and group is to be changed.
owner	Integer. The owner to assign to the file named by path or referenced by fd.
group	Integer. The group to assign to the file named by path or referenced by fd.
fd	Integer. File descriptor of the file whose owner and group is to be changed.

RETURN VALUE

0	Indicates successful completion.
-1	Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_[f]chown changes the owner and group of the file named by path or referenced by fd to owner and group, respectively. Only the superuser can change the owner of the file. Both the owner of the file and the superuser can change the group of the file.

If owner or group is specified as -1, the corresponding ID of the file is not changed.

RESTRICTIONS

Permission checking is enabled only if the appropriate file server is set to check permissions (the default).

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_chown fails--the file will be unchanged--if any of the following are true:

CMFS_ENOTDIR
: A component of the path prefix of path is not a directory.

CMFS_ENAMETOOLONG
: The length of a component of path exceeds 255 characters in length.

CMFS_ENOENT
: The file referred to by path does not exist.

CMFS_EACCES

Search permission is denied for a component of the path prefix of path.

CMFS_EPERM
The user ID specified by owner is not the current owner ID of the file, or the group ID specified by group is not the current group ID of the file and the effective user ID is not the superuser.

CMFS_EINVAL
Sockets are not supported.

CMFS_fchown fails--the file mode remains unchanged--if any of the following are true:

CMFS_EBADFD
The file descriptor is not valid.

CMFS_EPERM
The user ID specified by owner is not the current owner ID of the file, or the group ID specified by group is not the current group ID of the file, and the effective user ID is not the superuser.

CMFS_EIO An I/O error occurred while reading from or writing to the file system.

SEE ALSO

CMFS_chmod
CMFS_stat
cmchmod
cmchgrp
cmchown

NAME

CMFS_fcntl - Controls various characteristics of a file.

C SYNTAX

```
#include <cm/cm_file.h>

res = CMFS_fcntl(fd, request, arg)
int res, fd, request, arg;
```

FORTRAN SYNTAX

```
res = CMFS_FCNTL (fd, request, arg)
INTEGER res, fd, request, arg
```

ARGUMENTS

fd	File descriptor, which was returned by a previous call to CMFS_creat or CMFS_open.
request	Integer. Specifies which operations to perform on fd; the values allowed are described below.
arg	Integer. The meaning of arg depends on request and is described below.

RETURN VALUE

0	Indicates successful completion.
-1	Indicates an error occurred; the external variables CMF_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

The CMFS_fcntl system call provides control over the file descriptors of open files. It performs a variety of operations: creating and sizing CM block-special files, and obtaining and setting a file descriptor status flag.

CMFS_fcntl allows you to create CM block-special files. Such files provide direct access to the raw disk, instead of letting the system handle the disk access as with ordinary file operations. The fd argument is the file descriptor. The request argument defines what you want done; its allowed values are described below. The meaning of the arg argument varies according to the request argument, as described below.

CMFS_fcntl allows you to access raw physical disk blocks on the disk. Two calls to CMFS_fcntl are needed. The first one uses CMFS_MAKE_BLOCK_SPECIAL to declare that the file is a block-special file, and the second one uses CMFS_SET_SIZE to specify the number of blocks needed.

Not all the requests allowed for the UNIX fcntl call are supported by CMFS_fcntl. The requests supported follow:

CMFS_MAKE_BLOCK_SPECIAL
: Make this file a block-special file. The argument arg is the device number. The file must be a regular file of zero length.

CMFS_SET_SIZE
: Set the size in disk blocks of the block-special file. The argument arg is the size of the raw device as the number of CM file disk blocks. The file must be a CM block-special file of zero length. The size of a disk block is defined in cm/cm-param.h as the variable BYTESPERBLOCKSTRIPE.

CMFS_F_GETFL, CMFS_F_SETFL
: Get or set file descriptor status flags. Each file descriptor points to an entry in an array of file pointers which, among other things, define the file's current status. The file descriptor

status flags, which in C are defined in cm_file.h, are returned. The following flags are defined:

CMFS_FSTREAMING Activate streaming I/O mode.
CMFS_FAPPEND Set append mode.

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_fcntl fails if the following is true:

CMFS_EBADF
fd is not a valid open file descriptor.

CMFS_EINVAL
CMFS_MAKE_BLOCK_SPECIAL is requested and fd is not a regular file of length zero.

CMFS_EINVAL
CMFS_SET_SIZE is requested and fd is not a block-special file.

SEE ALSO

CMFS_close
CMFS_open

NAME

CMFS_stat - Obtains file status information.

C SYNTAX

```
#include <cm/cm_stat.h>

n = CMFS_stat(path, buffer)
char *path;
struct cm_stat *buffer;
int n;

n = CMFS_fstat(fd, buffer)
int fd, n;
struct cm_stat *buffer;

struct cm_stat
{
 int  cmst_dev;     /* device inode resides on              */
 long cmst_ino;     /* this inode's number                  */
 unsigned cmst_mode;/* protection                           */
 int  cmst_nlink;   /* number of hard links to the file     */
 int  cmst_uid;     /* user-id of owner                     */
 int  cmst_gid;     /* group-id of owner                    */
 int  cmst_rdev;    /* device type, for inode that is device */
 long cmst_size;    /* total size of file in bytes          */
 long cmst_atime;   /* file last access time                */
 int  cmst_spare1;
 long cmst_mtime;   /* file last modify time                */
 int  cmst_spare2;
 long cmst_ctime;   /* file last status change time         */
 int  cmst_spare3;
 long cmst_blksize; /* optimal blocksize for file,          */
                    /* system i/o ops in total bits         */
 long cmst_blocks;  /* actual number of blocks allocated    */
 long cmst_free_bytes;
 long cmst_physmach;
 long cmst_vps;
 long cmst_numextents;
 long cmst_cmwords;       /* size of file in CM words       */
 long cmst_bits_per_cmword; /* unit of CM word              */
 long cmst_spare[10];     /* reserved for future enhancements */
};
```

FORTRAN SYNTAX

```
n = CMFS_STAT (path, buffer)
INTEGER n, buffer(32)
STRING path

n = CMFS_FSTAT (fd, buffer)
INTEGER n, fd, buffer(32)
```

ARGUMENTS

path	Character string. The file about which to return status information.
buffer	A pointer to a cm_stat structure in C; an array of 32 integers in Fortran. The structure that receives the file status information.

fd — File descriptor (which was returned by a previous call to CMFS_creat or CMFS_open) of the file about which to return status information.

RETURN VALUE

0 — Indicates successful completion.

-1 — Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

None.

DESCRIPTION

CMFS_stat obtains information about the file named by path and places it in the status structure indicated by buffer. Read, write, or execute permission of the named file is not required, but all directories listed in the pathname leading to the file must be searchable. CMFS_fstat obtains the same information about an open file referenced by the descriptor fd.

In C, the argument buffer is a pointer to a cm_stat structure; the structure is defined in the file <cm/cm_stat.h>.

The elements in the status structure are described below. In C, the contents of an element are referenced by structure-name.cmst_element-name.

dev — The device on which the inode resides.

ino — This inode's number.

mode — The protection in effect for this file.

nlink — The number of hard links to the file.

uid — The user ID of the file's owner.

gid — The group ID of the file's owner.

rdev — The device type, for the inode that is the device.

size — The total size of the file in bytes. However, if a file's size is equal to or greater than 2 gigabytes, its reported size will be 2 gigabytes - 1.

atime — Time when file data was last read or modified. This is changed by the system calls CMFS_[f]stat, CMFS_serial_readwrite_file, but is not changed when a directory is searched.

mtime — Time when data was last modified. This is not set by changes of owner, group, link count, or mode. It is changed by the system call CMFS_serial_write_file.

ctime — Time when file status was last changed. This is set both by writing and changing the inode, and is changed by the system calls CMFS_link, CMFS_unlink, and CMFS_serial_write_file(_always).

blksize — The optimal block size for CM file system I/O operations, in the total number of bits.

blocks — The actual number of blocks allocated.

free-bytes — The number of free bytes in the last CM word in the file. On a CM-5, this value is based on the bits-per-cmword and size fields.

physmach — The number of physical processors the file spans (physical width). On a CM-5, this field is set to 1.

vps — The number of virtual processors the file spans (virtual width). On a CM-5, this field is set to 1.

numextents — The number of extents the file occupies. On a CM-5, this field is set to 1.

cmwords — The size of file in CM words. On a CM-5, this value is based on the bits-per-cmword and

size fields.

bits-per-cmword
The size in bits of a CM word. On a CM-5, this field is set to 128 (16 bytes).

The status element mode comprises one of the file-type bit patterns OR'ed with one or more permission-type bit patterns:

File Type

CMFS_S_IFMT	0170000	Is a file.
CMFS_S_IFDIR	0040000	Is a directory.
CMFS_S_IFBLK	0060000	Is a block-special file.
CMFS_S_IFCHR	0020000	Is a character-special file.
CMFS_S_IFREG	0100000	Is a regular file.

Permission Type

CMFS_S_IREAD	0000400	Has read permission for the owner.
CMFS_S_IWRITE	0000200	Has write permission for the owner.
CMFS_S_IEXEC	0000100	Has execute/search permission for the owner.

The mode bits 0000070 and 0000007 encode group and others permissions.

For further information, see CMFS_chmod.

RESTRICTIONS

CMFS_[f]stat do not return a file's geometry information. The command cmstat, however, lists geometry information along with other information contained in the file's attribute file.

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_stat fails if any of the following are true:

CMFS_ENOTDIR
A component of the path prefix is not a directory.

CMFS_EINVAL
path contains a character with the high-order bit set.

CMFS_ENAMETOOLONG
path exceeded 255 characters in length.

CMFS_ENOENT
The file referred to by path does not exist.

CMFS_EACCES
Search permission is denied for a component of the path prefix.

CMFS_EFAULT
The arguments buffer or path point to an invalid address.

CMFS_EIO An I/O error occurred while reading from or writing to the file system.

The system call CMFS_fstat fails if one or more of the following are true:

CMFS_EBADF
fd is not a valid open file descriptor.

CMFS_EFAULT
buffer points to an invalid address.

CMFS_EIO An I/O error occurred while reading from or writing to the file system.

SEE ALSO

CMFS_[f]chmod
CMFS_[f]chown
cmstat

NAME

CMFS_getwd - Returns the current CMFS file system's current working directory.

C SYNTAX

```
n = CMFS_getwd(buffer)
char *n, *buffer;
```

FORTRAN SYNTAX

Not supported

ARGUMENTS

buffer	A pointer to a buffer that will be filled in with the CMFS file system's current working directory.

RETURN VALUE

n	A pointer to the CMFS file system's current working directory.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_getwd returns the CMFS file system's current working directory.

SEE ALSO

CMFS_chdir

NAME

CMFS_ioctl - Supports CM character-special device drivers.

C SYNTAX

```
n = CMFS_ioctl(fd, cmd, arg)
int n, fd, cmd;
caddr_t arg;
```

ARGUMENTS

fd	The file descriptor returned by a preceding CMFS_open call.
cmd	A driver-specific code that specifies the device-specific task to perform.
arg	A pointer to a driver-specific buffer, which usually contains information needed by the driver to perform the task specified by cmd.

RETURN VALUE

0	Indicates successful completion.
-1	Indicates an error occurred; the external variable CMFS_errno is set to the CMFS error code returned by the fileserver (and by the CMFS character-special device driver called).

CM STATE CHANGE

None.

DESCRIPTION

CMFS_ioctl performs an operation--for example, resetting a tape drive or obtaining status information--on the device referred to by the file descriptor fd. Note that the character-special file that represents the device must be open.

The set of tasks that an ioctl routine performs is device-specific. For each device, a header file lists the request codes--defines--for specific tasks. Pass the pertinent define to the driver via the CMFS_ioctl cmd argument. If the driver requires additional information to perform the operation, pass it via the arg argument.

A typical CMFS_ioctl call encodes in cmd the general driver task--to obtain status information about the device, for example--and encodes in arg specific information about the device or operation--the device's status register location, for example.

The steps below summarize the composition of a typical CMFS_ioctl call.

1. CMFS_ioctl's first argument, fd, is the device's open file descriptor.
2. CMFS_ioctl's second argument, cmd, is the name of one of the defines in the device's header file.

   The list of valid cmd values is in the device's header file, which is usually in /usr/include/cm. If there are no comments specifying which defines are cmd values, you can identify them by their typical form:

   ```
   # define define_name _IOxx(x, #, data-type)
   ```

3. CMFS_ioctl's third argument, arg, usually contains information needed by the driver in order to perform the task indicated by cmd. The device's header file indicates what kind of information to assign to arg.

   arg is a pointer to a variable of type data-type (from the define's definition, i.e., _IOxx(x, #, data-type): in your program, define a variable of type data-type, and assign it an appropriate value, according to information in the header file. Then, in the CMFS_ioctl call itself, list arg as a pointer to that variable.

EXAMPLE

The following code demonstrates how to make a CMFS_ioctl call to rewind a tape on a magnetic tape device. For reference, the header file cm_mtio.h is listed after the code.

```
#include <sys/types.h>      /* defines type daddr_t,
                            used in <cm/cm_mtio.h>      */

#include <cm/cm/cm_ioctl.h> /* has the macros associated with
                            the defines in <cm/cm_mtio.h> */

#include <cm/cm_mtio.h>     /* lists a define for each task
                            that the ioctl can perform for
                            this device                 */

[...declarations, including the one below...]
struct cm_mtop      cm_rewind;

[...code to open the device, etc...]
/* rewind the tape */
cm_rewind.mt.op = MTREW;
cm_rewind.mt.count = 1;
CMFS_ioctl(fd, CMMTIOCTOP, &cm_rewind);

[...code to write to the device, close the device, etc...]


/* MagTape header file /usr/include/cm/cm_mtio.h          */

/*
 * Structures and definitions for mag tape io control commands
 */

/*
 * structure for CMMTIOCTOP - mag tape op command
 */

struct  cm_mtop {
        long    mt_op;      /* operations defined below */
        daddr_t mt_count;   /* how many of them         */
};

/* operations */
#ifndef MTFSF
#define MTWEOF   0   /* write an end-of-file record        */
#define MTFSF    1   /* forward space file                 */
#define MTBSF    2   /* backward space file                */
#define MTFSR    3   /* forward space record               */
#define MTBSR    4   /* backward space record              */
#define MTREW    5   /* rewind                             */
#define MTOFFL   6   /* rewind and put the drive offline    */
#define MTNOP    7   /* no operation, sets status only     */
#define MTRETEN  8   /* retension the tape                 */
#define MTERASE  9   /* erase the entire tape              */
```

```
#define MTEOM    10  /* position to end of media (SCSI only) */

#endif

 /*
 /* structure for CMMTIOCGET - mag tape get status command
  */

struct  cm_mtget        {
   short   mt_type;    /* type of magtape device       */
                       /* the next two registers
                          are grossly device-dependent */
   short   mt_dsreg;   /* "drive status" register      */
   short   mt_erreg;   /* "error" register             */
   short   mt_resid;   /* residual count               */
};

#define CMMTIOCTOP    _IOW(m, 1, struct cm_mtop)
                      /* Do a tape op.                */

#define CMMTIOCGET    _IOR(m, 2, struct cm_mtget)
                      /* Get status.                  */

#define CMMTIOCSETREC _IOW(m, 1, int)
                      /* Set record size              */
```

SEE ALSO

CMFS_mknod
/dev/hippi
/dev/ufb[_p]

NAME

CMFS_link - Creates a hard link to a file.

C SYNTAX

```
n = CMFS_link(path1, path2)
char *path1, *path2;
int n;
```

FORTRAN SYNTAX

```
n = CMFS_LINK (path1, path2)
INTEGER n
STRING path1, path2
```

ARGUMENTS

path1	Character string. The file to which to create a link.
path2	Character string. The name of the link.

RETURN VALUE

0	Indicates successful completion.
-1	Indicates an error occurred; the external variables CMFS_errno and are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_link creates a hard link to the file named by path1; the link has the name path2. Both path1 and path2 must be in the same file system. The file path1 must exist, and it must not be a directory. Both the old and the new link share equal access and rights to the underlying object.

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_link fails--no link is created--if any the following are true:

CMFS_ENOTDIR
: A component of either path prefix is not a directory.

CMFS_EINVAL
: Either pathname contains a character with the high-order bit set.

CMFS_ENAMETOOLONG
: Path1 or path2 exceeded 255 characters in length.

CMFS_ENOENT
: A component of either path prefix does not exist.

CMFS_ENOENT
: The file named by path1 does not exist.

CMFS_EACCES
: A component of either path prefix denies search permission.

CMFS_EACCES
: The requested link requires writing in a directory with a mode that denies write permission.

CMFS_EEXIST
: The link named by path2 does exist.

CMFS_EPERM
: The file named by path1 is a directory and the effective user ID is not superuser.

CMFS_EXDEV

The link named by path2 and the file named by path1 are on different file systems.

CMFS_EROFS
The requested link requires writing in a directory on a read-only file system.

CMFS_ENOSPC
The directory in which the entry for the new link is being placed cannot be extended because there is no space left on the file system containing the directory.

CMFS_EDQUOT
Quotas are not supported.

CMFS_EIO An I/O error occurred while reading from or writing to the file system to make the directory entry.

CMFS_ESTALE
The file handle given in the argument is invalid. The file referred to by that file handle no longer exists or has been revoked.

CMFS_ETIMEDOUT
A "connect" request or remote file operation failed because the connected party did not properly respond after a period of time that is dependent on the communications protocol.

SEE ALSO

CMFS_unlink

NAME

CMFS_mkdir - Makes a directory.

C SYNTAX

```
#include <sys/types.h>

n = CMFS_mkdir(path, mode)
char *path;
int n, mode;
```

FORTRAN SYNTAX

```
n = CMFS_MKDIR (path, mode)
INTEGER mode
STRING path
```

ARGUMENTS

path	Character string. The directory to be created.
mode	Integer. The mode to assign to the file. See CMFS_chmod.

RETURN VALUE

0	Indicates successful completion.
-1	Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the error.

DESCRIPTION

CMFS_mkdir creates a new directory file with name path. The directory is created with mode mode, as described in CMFS_chmod. The new directory's owner ID effective user is set to the process's effective user ID. The directory's group ID is set to that of the parent directory in which it is created.

ERRORS

CMFS_ENOTDIR
: A component of the path prefix of path is not a directory.

CMFS_ENAMETOOLONG
: The length of path exceeds 255 characters.

CMFS_ENOENT
: A component of the path prefix of path does not exist.

CMFS_EACCES
: Search permission is denied for a component of the path prefix of path.

CMFS_EIO
: An I/O error occurred while reading from or writing to the file system.

CMFS_ENOSPC
: The directory in which the entry for the new file is being placed cannot be extended because there is no space left on the file system containing the directory.

CMFS_ENOSPC
: There are no free inodes on the file system on which the file is being created.

CMFS_EEXIST
: The file referred to by path exists.

SEE ALSO

CMFS_rmdir
CMFS_chmod

NAME

CMFS_mknod - Create a new file that can call a specific device driver.

C SYNTAX

```
#include <sys/types.h>

n = CMFS_mknod(path, mode, device)
char *path;
dev_t device;
int n, mode;
```

FORTRAN SYNTAX

Not supported.

ARGUMENTS

path	Character string. The file to be created.
mode	Integer. The mode to assign to the file. Must be CMFS_S_IFCHR (0020000) for character-special files or CMFS_S_IFBLK (0060000) for block-special files. The owner ID of the file is set to the effective user ID of the process, and the group of the file is set to the effective group ID of the process.
device	A configuration-dependent specification of a character or block I/O device. It contains the device's major and minor device numbers as encoded by the major() and minor() macros defined in <sys/types.h>.

RETURN VALUE

0	Indicates successful completion.
-1	Indicates an error occurred; CMFS_errno and errno are set to indicate the error.

DESCRIPTION

CMFS_mknod creates a new file in the CM filesystem named by the path name pointed to by path. Such a file, called a special file, represents an I/O device. In the CM file system, I/O devices are connected to a VMEIO host computer or to a CM-IOP, depending on whether they are VME-based or have a SCSI interface, respectively.

ERRORS

ENOTDIR	A component of the path prefix of path is not a directory.
ENAMETOOLONG	The length of path exceeds 255 characters.
ENOENT	A component of the path prefix of path does not exist.
EACCES	Search permission is denied for a component of the path prefix of path.
EIO	An I/O error occurred while reading from or writing to the file system.
ENOSPC	The directory in which the entry for the new file is being placed cannot be extended because there is no space left on the file system containing the directory.
ENOSPC	There are no free inodes on the file system on which the file is being created.
EEXIST	The file referred to by path exists.

SEE ALSO

CMFS_ioctl

NAME

CMFS_open - Opens or creates and opens a file for reading or writing.

C SYNTAX

```
#include <cm/cm_file.h>

fd = CMFS_open(path,flags[,mode])
int fd, flags, mode;
char *path;
```

FORTRAN SYNTAX

```
fd = CMFS_OPEN (path, flags, mode)
INTEGER fd, flags, mode
STRING path
```

LISP SYNTAX

(CMFS:open path flags [mode])

ARGUMENTS

path — Character string. The pathname of the file to be opened.

flags — Integer. Certain characteristics of I/O to the file are controlled by ORing flags from the list below and supplying the result as the flags argument. (Note that only one of CMFS_O_RDONLY, CMFS_O_WRONLY, or CMFS_O_RDWR may be used at once.)

CMFS_O_RDONLY
: Open the file for reading only.

CMFS_O_WRONLY
: Open the file for writing only.

CMFS_O_RDWR
: Open the file for reading and writing.

CMFS_O_APPEND
: Prior to each write, set the file pointer to the end of the file.

CMFS_O_CREAT
: If the file does not exist, create it. Otherwise, do nothing.

CMFS_O_TRUNC
: If the file exists, truncate its length to 0.

CMFS_O_WILDCARD
: If the file does not exist (CMFS_O_CREAT is also specified), create a wildcard file. If the file already exists, ignore its associated geometry and open it in "wildcard mode.".

CMFS_O_EXCL
: If CMFS_O_CREAT is also set, return an error message if the file already exists. This can be used to implement a simple exclusive access locking mechanism.

CMFS_O_FSYNC
: Disable asynchronous inode writing for write operations (but not for ftruncate operations). See fsserver.8.

mode — Integer, optional argument. The mode to assign to the newly created file (see CMFS_chmod). Mode is ignored unless the CMFS_O_CREAT flag is set.

RETURN VALUE

fd — File descriptor, to be used in other operations on the open file.

-1 — Indicates an error occurred; the external variables CMFS_errno and errno are set to

indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_open opens the file specified by path for reading or writing, and returns a non-negative integer called a file descriptor. The file descriptor is then used in other operations on that file. CMFS_open also sets the file pointer, which is used to mark the current position within the file, to the beginning of the file.

The flags argument determines whether the file is read or written, and determines additional options. This argument can be used to indicate the file is to be created if it does not already exist. To do this, set the CMFS_O_CREAT flag. The file is created with the mode given by the optional argument mode (in CM Fortran, however, mode is required), as described in CMFS_chmod.

By default, when either CMFS_creat or CMFS_open is used from a front end attached to a CM to create a file, the format of the new file is parallel: the file's VP geometry is set to match the geometry of the CM's current VP set. When either of these calls is used from a machine not attached to a CM to create a file, the format of the new file is serial, and the file is assigned no geometry.

When both flags CMFS_O_CREAT and CMFS_O_WILDCARD are specified, the CM file system creates a wildcard file, which has the generic geometry of virtual width = physical width = 1. When the CMFS_O_WILDCARD flag is specified without the CMFS_O_CREAT flag, the CM file system ignores the geometry of the already-created and opens it in "wildcard mode," under the assumed generic geometry.

The CMFS_O_RDONLY, CMFS_O_WRONLY, and CMFS_O_RDWR flags are for permission checking. In C, all the flags are defined in the header file <cm/cm_file.h>.

A file can be opened more than once, and can therefore have more than one file descriptor associated with it. There is a system-enforced limit on the number of open file descriptors per process. CMFS_close closes a file descriptor.

File protection is supported. When a file is created using CMFS_open or CMFS_creat, the owner and group of the file are set to the default values, as described in the UNIX reference manual pages for open(2) and creat(2); the file's mode is explicitly set (see CMFS_chmod). The utility cmls, as well as the library calls CMFS_[f]stat, shows the correct mode, owner, and group associated with a file.

RESTRICTIONS

Up to 200 CMFS files can be open on any CMFS file system data-storage device (DataVault, front end, VMEIO host computer, CM-HIPPI). A single process can have a maximum of 64 CMFS files open at any given time.

The name of a CMFS file can be no more than 255 characters.

To open a striped file, all file servers in the striped file system must be operational.

All CMFS files must be closed prior to a UNIX execve(2) system call. A file descriptor remains open across an execve(2) system call, but none of the CM file system data structures are set up.

The following flags, available to a UNIX open(2) call, are not implemented for the CM file system:

O_SYNC
O_FSYNC

O_NDELAY
O_BLKINUSE
O_BLKANDSET

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_open fails if any of the following are true:

CMFS_EINVAL
: The pathname contains a character with the high-order bit set.

CMFS_ENOTDIR
: A component of the path prefix is not a directory.

CMFS_ENOENT
: CMFS_O_CREAT is not set and the named file does not exist.

CMFS_ENOENT
: A component of the pathname that must exist does not exist.

CMFS_EACCES
: The required permissions for reading and/or writing are denied for the named flag.

CMFS_EACCES
: Search permission is denied for a component of the path prefix.

CMFS_EACCES
: CMFS_O_CREAT is specified, the file does not exist, and the directory in which it is to be created does not permit writing.

CMFS_EISDIR
: The named file is a directory, and the arguments specify it is to be opened for writing.

CMFS_EROFS
: The named file resides on a read-only file system, and the file is to be modified.

CMFS_EMFILE
: The maximum number of file descriptors are currently open.

CMFS_ENXIO
: The named file is a character-special or block-special file, and the device associated with this special file does not exist.

CMFS_ETXTBSY
: The file is a pure procedure (shared text) file that is being executed and the open call requests write access.

CMFS_EEXIST
: CMFS_O_CREAT and CMFS_O_EXCL were specified and the file exists.

CMFS_ENAMETOOLONG
: Path exceeded 255 characters in length.

CMFS_ENFILE
: The system file table is full.

CMFS_ENOSPC
: CMFS_O_CREAT is specified, the file does not exist, and the directory in which the entry for the new file is being placed cannot be extended because there is no space left on the file system containing the directory.

CMFS_ENOSPC
: CMFS_O_CREAT is specified, the file does not exist, and there are no free inodes on the file system on which the file is being created.

CMFS_EDQUOT
Quotas are not supported.

CMFS_EIO An I/O error occurred while making the directory entry or allocating the inode for CMFS_O_CREAT.

CMFS_ESTALE
The file handle given in the argument is invalid. The file referred to by that file handle no longer exists or has been revoked.

CMFS_ETIMEDOUT
A "connect" request or remote file operation failed because the connected party did not properly respond after a period of time that is dependent on the communications protocol.

CMFS_ESTRIPE_NOTAVAIL
One or more of the file servers serving the striped CM file system is not running.

SEE ALSO

cmls
CMFS_[f]chmod
CMFS_close
CMFS_creat

NAME

Directory Operations - Open, read, seek, tell the location in, and close directories.

C SYNTAX

```
#include <cm/cm_dir.h>

dirp = *CMFS_opendir(pathname)
char *pathname;
CMDIR *dirp;

entp = CMFS_readdir(dirp)
CMDIR *dirp;
struct cm_direct *entp;

n = CMFS_telldir(dirp)
long n;
CMDIR *dirp;

void CMFS_seekdir(dirp, loc)
CMDIR *dirp;
long loc;

n = CMFS_closedir(dirp)
CMDIR *dirp;
int n;
```

FORTRAN SYNTAX

Not supported.

ARGUMENTS

pathname	Character string. The pathname of the directory to be opened.
dirp	A pointer to identify the directory stream. dirp is returned by CMFS_opendir and used in the other directory operations.
loc	Integer. Returned by an earlier call to CMFS_telldir, this argument to CMFS_seekdir sets the position of the next CMFS_readdir. A loc of 0 sets the position to the beginning of the directory stream.
entp	A pointer to the next directory entry. entp is returned by CMFS_readdir.

RETURN VALUES

CMFS_opendir

dirp	Pointer to identify the directory stream.
NULL	Indicates the directory cannot be accessed or insufficient memory is available to open it.

CMFS_readdir

entp	Pointer to the next directory entry.
NULL	Indicates the end of the directory has been reached or an invalid seekdir operation has been detected.

CMFS_telldir

n	Current location associated with the directory stream.

CMFS_seekdir

None.

CMFS_closdir

0	Indicates successful close
-1	Indicates an error; CMFS_errno is set.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_opendir opens the directory named by pathname and associates a directory stream with it. CMFS_opendir returns a pointer to identify the directory stream in subsequent operations. The pointer NULL is returned if the specified filename can not be accessed, or if insufficient memory is available to open the directory file.

CMFS_readdir returns a pointer to the next directory entry. It returns NULL or nil upon reaching the end of the directory or upon detecting an invalid seek operation. CMFS_readdir uses the UNIX get-directories system call to read directories.

CMFS_telldir returns the current location associated with the named directory stream. Values returned by CMFS_telldir are good only for the lifetime of the DIR pointer from which they are derived. If the directory is closed and then reopened, the CMFS_telldir value may be invalidated due to undetected directory compaction.

CMFS_seekdir sets the position of the next CMFS_readdir operation on the directory stream. Only values returned by CMFS_telldir should be used with CMFS_seekdir.

CMFS_closedir closes the named directory stream. If the close is successful, 0 is returned; if the close fails, -1 is returned and CMFS_errno is set. All resources associated with this directory stream are released.

The directory stream that CMFS_opendir associates with pathname contains an entry for each directory residing in pathname. Each directory entry has associated with it four pieces of information. This information is stored in a directory structure having four elements. A pointer to this structure is returned by CMFS_readdir. The four elements are:

cmd_ino	Inode number
cmd_reclen	Directory record length
cmd_namlen	Directory name length
cmd_name	Directory name entry

The content of an element is referenced by pointer -> element-name. For example, if dir_ent_ptr is the pointer returned by CMFS_readdir, the directory name entry is referenced by dir_ent_ptr -> d_name.

EXAMPLE

The following sample code searches a directory for the entry name.

```
len = strlen(name);
dirp = CMFS_opendir(.);
for (dp = CMFS_readdir(dirp); dp != NULL;
    dp = CMFS_readdir(dirp))
if (dp->cmd_namlen == len && !strcmp(dp->cmd_name, name)) {
  CMFS_closedir(dirp)
  return FOUND;
}
CMFS_closedir(dirp);
return NOT_FOUND;
```

CMFS_perror (3)

NAME

CMFS_perror - Writes a system error message to the standard error file.

C SYNTAX

```
void
CMFS_perror(s)
char *s;
```

FORTRAN SYNTAX

```
SUBROUTINE CMFS_PERROR(s)
STRING s
```

ARGUMENTS

s Character string. The user-supplied portion of the error message.

RETURN VALUE

None.

CM STATE CHANGE

None.

DESCRIPTION

The CMFS_perror subroutine writes a short error message on the standard error file describing the last error encountered during a system call. The error message consists of the string s, a colon, the message, and a newline.

The error description written is determined by the number taken from the external variable CMFS_errno.

SEE ALSO

CMFS_errno

NAME

CMFS_physical_ftruncate - Truncates or extends an open file.

C SYNTAX

```
n = CMFS_physical_ftruncate(fd, number_cmwords, number_extra_bytes)
long number_cmwords, number_extra_bytes;
int fd, n;
```

FORTRAN SYNTAX

Not supported

LISP SYNTAX

Not supported

ARGUMENTS

fd File descriptor, which was returned by a previous call to **CMFS_creat** or **CMFS_open.**

number_cmwords
The number of cmwords you want the truncated file to contain. (Usually, 1 cmword = 512 bytes; verify by executing **cmstat** or **CMFS_[f]stat** on the file.)

number_extra_bytes
The number of additional bytes -- that is, additional to the number of cmwords specified by *number_cmwords* -- you want the truncated file to contain.

RETURN VALUE

0 Indicates successful completion.

-1 Indicates an error occurred; the external variables **CMFS_errno** and **errno** are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_physical_ftruncate changes the size of the file referenced by *fd* to any non-negative value, including one larger than the (2 Gbytes -1) limit imposed by **CMFS_serial_truncate_file.** The new size of the file in bytes is calculated by:

(*number-cmwords* * cmword size in bytes) + *number-extra-bytes*

ERRORS

If an error occurs, the value -1 is returned and the external variables **CMFS_errno** and **errno** are set to indicate the cause of the error.

CMFS_physical_ftruncate fails if any of the following are true:

CMFS_EROFS
The named file resides on a read-only file system.

CMFS_EIO An I/O error occurred updating the inode.

CMFS_ENOSPC
An error occurred while attempting to allocate disk blocks.

CMFS_EBADF
fd is not a valid descriptor.

CMFS_EBADF
fd is not open for writing.

CMFS_ETIMEDOUT
A "connect" request or remote file operation failed because the connected party did not properly respond after a period of time that is dependent on the communications protocol.

CMFS_EINVAL

number_cmwords or *number_extra_bytes* is negative.

CMFS_EINVAL
number_extra_bytes is greater than or equal to the number of bytes per cmword.

CMFS_EINVAL
fd references a socket, not a file.

SEE ALSO

CMFS_open
CMFS_serial_truncate_file
CMFS_[f]truncate_file

NAME

CMFS_physical_lseek - Moves the read/write pointer associated with an open file.

C SYNTAX

```
n = CMFS_physical_lseek(fd, offset_cmwords, offset_extra_bytes, whence)
long *offset_cmwords, *offset_extra_bytes;
int n, fd, whence;
```

FORTRAN SYNTAX

Not supported

ARGUMENTS

fd File descriptor (which was returned by a previous call to **CMFS_creat** or **CMFS_open**) of a file open for reading or writing.

offset_cmwords
The number of cmwords to offset the file pointer from the position specified by *whence*. Usually, 1 cmword = 512 bytes; verify by executing **cmstat** or **CMFS_[f]stat** on the file.) *offset_cmwords* may be negative.

offset_extra_bytes
The number of additional bytes -- that is, additional to the number of bytes specified by *offset_cmwords* -- to offset the file pointer from the position specified by *whence*. *offset_extra_bytes* may be negative.

whence Integer. The location at which to set the file pointer. *whence* is one of the following:

CMFS_L_SET
Set the pointer to ((*offset_cmwords* * cmword size in bytes) + *offset_extra_bytes*) bytes.

CMFS_L_INCR
Set the pointer to the current position plus ((*offset_cmwords* * cmword size in bytes) + *offset_extra_bytes*) bytes.

CMFS_L_XTND
Set the pointer to the length of the file plus ((*offset_cmwords* * cmword size in bytes) + *offset_extra_bytes*) bytes.

RETURN VALUE

0 Indicates success.

-1 Indicates an error occurred; the external variables **CMFS_errno** and **errno** are set to indicate the cause of the error.

offset_cmwords
When successful, on return this variable's value is the number of cmwords contained from the beginning of the file to the new file pointer position.

offset_extra_bytes
When successful, on return this variable's value is the number of bytes in the space between the last cmword boundary and the new file pointer position. For instance, if the file pointer's position is on a cmword boundary, this variable's value is 0.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_physical_lseek moves the file pointer associated with a file open for reading or writing, so the next data item read or written need not be logically adjacent to the item used in the previous operation. The file-pointer-position value can be greater than (2 Gbytes -1) (the limit imposed by **CMFS_lseek** and **CMFS_serial_lseek**). The file-pointer-position value is specified by the arguments *offset_cmwords*, *offset_extra_bytes*, and *whence*, as indicated in the previous ARGUMENTS section.

The argument *fd* represents the file open for reading or writing.

Note that **CMFS_physical_lseek** does not actually perform a seek on the disks; that is, it only moves the file pointer, not the disk heads. The disks don't seek until a **CMFS_read_file(_always)** or **CMFS_write_file(_always)** is issued.

Seeking far beyond the end of a file, then writing, causes file blocks to be allocated in the gap between the end of the file and the location at which the write proceeds. The contents of these blocks are undefined, containing whatever was left there by previous uses of those blocks.

Upon successful completion, 0 is returned.

RESTRICTIONS

Although the file-pointer-position value can be negative, any attempt to perform I/O while at a negative offset will fail.

Some devices are incapable of seeking. The value of the pointer associated with such a device is undefined.

ERRORS

If an error occurs, the value -1 is returned and the external variables **CMFS_errno** and **errno** are set to indicate the cause of the error. **CMFS_physical_lseek** fails if any of the following are true:

CMFS_EBADF
: *fd* is not an open file descriptor.

CMFS_EINVAL
: *whence* is not a proper value.

CMFS_EISPIPE
: *fd* refers to a socket, not a file.

SEE ALSO

CMFS_flush
CMFS_open
CMFS_serial_lseek
CMFS_lseek

NAME

CMFS_read_file[_always] - Reads from a file or a connected socket into CM-5 processing nodes.

SYNTAX

C SYNTAX

Not supported

C* SYNTAX

n = **CMFS_read_file[_always]**(*fd*, *pvar**, *count*)
int *n*, *fd*, *count*;
void * *pvar*;

FORTRAN SYNTAX

Not supported

CM FORTRAN SYNTAX

N = **CMFS_READ_FILE[_ALWAYS]**(*FD*, *ARRAY*, *COUNT*)
INTEGER *N*, *FD*, *COUNT*
ANY_TYPE ARRAY(:,:,...,:) ! any type, any rank

ARGUMENTS

fd File descriptor (returned by a previous call to **CMFS_creat** or **CMFS_open**) of a file open for reading, or socket descriptor (returned by a previous call to **CMFS_socket** or **CMFS_accept**) of a connected socket.

*pvar** (C*) A pointer to a parallel variable of any type in which to place the data.

ARRAY (CMF)
An array (of any type or rank) in which to place the data.

count Integer. The number of bytes to read into each element of the parallel variable or array. *count* can be any number of bytes, but performance is diminished when *count* is not a multiple of 4 bytes (the CM-5 word size).

RETURN VALUE

n, *N* The total number of bytes read into the parallel variable or array

0 Indicates that end-of-file has been reached, or that the sending program has closed or shut down its socket and all pending data has been read.

-1 Indicates an error occurred; the external variables **CMFS_errno** and **errno** are set to indicate the cause of the error.

CM STATE CHANGE

In the parallel variable *pvar*, or array *ARRAY*, *n* bytes are placed.

CONTEXT

CMFS_read_file: Conditional, occurs in selected PNs.
CMFS_read_file_always: Unconditional, occurs in all PNs.
Currently both **CMFS_read_file** and **CMFS_read_file_always** operate unconditionally. Since a future software release may support conditional reads via **CMFS_read_file**, we recommend that programs use only **CMFS_read_file_always** at this time.

DESCRIPTION

CMFS_read_file and **CMFS_read_file_always** attempt to read *count* bytes of information into each element of the parallel variable pointed to by the *pvar* argument (in C*) or each element of the array *ARRAY* (in CM Fortran) from the object represented by fd; fd is either a file descriptor returned by a previous call to **CMFS_creat** or **CMFS_open**, or a socket descriptor returned by a previous call to **CMFS_socket** or **CMFS_accept**. To read from a socket, you must use **CMFS_read_file_always**, as reading from a socket is unconditional. If no messages are available at the socket, the call waits for a message to arrive. When a file is read, the read starts at a position given by the file pointer associated with *fd* (see **CMFS_serial_lseek** (3)). Upon return from **CMFS_read_file[_always]**, the pointer is incremented by the number of bytes actually read.

Upon successful completion, **CMFS_read_file** and **CMFS_read_file_always** return the number of bytes read. If fewer bytes remain to be read than the number requested, the remaining bytes are read until the end-of-file is encountered or all data has been read from the socket. The routine returns the number of bytes actually read (as opposed to the number requested). If an attempt to read is made after end-of-file has been encountered, zero is returned. If the sending program has closed or shut down its socket, the remaining data is read; if no more data remains to be read, zero is returned.

RESTRICTIONS

Reading a file requires that the read begin on a cmword boundary, where a cmword is usually defined as 4096 bits. You will receive an error if you attempt to read starting at an illegal position.

ERRORS

If an error occurs, the value **-1** is returned and the external variables **CMFS_errno** and **errno** are set to indicate the cause of the error. **CMFS_read_file** and **CMFS_read_file_always** fail if any of the following are true:

CMFS_EBADF
: *fd* is neither a valid file descriptor open for reading, nor a valid socket descriptor.

CMFS_EINVAL
: The current file position is negative.

CMFS_EIO
: An I/O error occurred while reading from the file system. This error could occur if a disk drive failed and a second DataVault disk drive failed before the system administrator corrected the first failure.

CMFS_ESTRIPE_NOTAVAIL
: One or more of the file servers serving the striped CM file system is not running.

CMFS_ECONNABORTED
: The socket connection has been aborted by the underlying TCP protocol.

CMFS_ECONNRESET
: The socket connection has been reset by the underlying TCP protocol.

CMFS_ENOTCONN
: If *fd* is a socket descriptor, the socket is not connected. If *fd* is the descriptor of the CM-HIPPI device, the connection to the remote system has been broken or has not yet been established.

CMFS_ETIMEDOUT
: A connection request or remote file operation failed because the connected party did not properly respond after a period of time that is dependent on the communications protocol. If *fd* is the descriptor of the CM-HIPPI device, the CM-HIPPI timed out waiting for data or for the remote system to issue a connection request.

SEE ALSO

CMFS_serial_lseek
CMFS_socket
CMFS_open
CMFS_write_file
CMFS_errno
intro(2) (on Suns)

NAME

Directory Operations - Open, read, seek, tell the location in, and close directories.

C SYNTAX

```
#include <cm/cm_dir.h>

dirp = *CMFS_opendir(pathname)
char *pathname;
CMDIR *dirp;

entp = CMFS_readdir(dirp)
CMDIR *dirp;
struct cm_direct *entp;

n = CMFS_telldir(dirp)
long n;
CMDIR *dirp;

void CMFS_seekdir(dirp, loc)
CMDIR *dirp;
long loc;

n = CMFS_closedir(dirp)
CMDIR *dirp;
int n;
```

FORTRAN SYNTAX

Not supported.

ARGUMENTS

pathname	Character string. The pathname of the directory to be opened.
dirp	A pointer to identify the directory stream. dirp is returned by CMFS_opendir and used in the other directory operations.
loc	Integer. Returned by an earlier call to CMFS_telldir, this argument to CMFS_seekdir sets the position of the next CMFS_readdir. A loc of 0 sets the position to the beginning of the directory stream.
entp	A pointer to the next directory entry.

RETURN VALUES

CMFS_opendir

dirp	Pointer to identify the directory stream.
NULL	Indicates the directory cannot be accessed or insufficient memory is available to open it.

CMFS_readdir

entp	Pointer to the next directory entry.
NULL	Indicates the end of the directory has been reached or an invalid seekdir operation has been detected.

CMFS_telldir

n	Current location associated with the directory stream.

CMFS_seekdir

None.

CMFS_closdir

0	Indicates successful close
-1	Indicates an error; CMFS_errno is set.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_opendir opens the directory named by pathname and associates a directory stream with it. CMFS_opendir returns a pointer to identify the directory stream in subsequent operations. The pointer NULL is returned if the specified filename can not be accessed, or if insufficient memory is available to open the directory file.

CMFS_readdir returns a pointer to the next directory entry. It returns NULL or nil upon reaching the end of the directory or upon detecting an invalid seek operation. CMFS_readdir uses the UNIX get-directories system call to read directories.

CMFS_telldir returns the current location associated with the named directory stream. Values returned by CMFS_telldir are good only for the lifetime of the DIR pointer from which they are derived. If the directory is closed and then reopened, the CMFS_telldir value may be invalidated due to undetected directory compaction.

CMFS_seekdir sets the position of the next CMFS_readdir operation on the directory stream. Only values returned by CMFS_telldir should be used with CMFS_seekdir.

CMFS_closedir closes the named directory stream. If the close is successful, 0 is returned; if the close fails, -1 is returned and CMFS_errno is set. All resources associated with this directory stream are released.

The directory stream that CMFS_opendir associates with pathname contains an entry for each directory residing in pathname. Each directory entry has associated with it four pieces of information. This information is stored in a directory structure having four elements. A pointer to this structure is returned by CMFS_readdir. The four elements are:

cmd_ino	Inode number
cmd_reclen	Directory record length
cmd_namlen	Directory name length
cmd_name	Directory name entry

In C, the content of an element is referenced by pointer -> element-name. For example, if dir_ent_ptr is the pointer returned by CMFS_readdir, the directory name entry is referenced by dir_ent_ptr -> d_name.

EXAMPLE

The following sample code searches a directory for the entry name.

```
len = strlen(name);
dirp = CMFS_opendir(.);
for (dp = CMFS_readdir(dirp); dp != NULL;
    dp = CMFS_readdir(dirp))
if (dp->cmd_namlen == len && !strcmp(dp->cmd_name, name)) {
  CMFS_closedir(dirp)
  return FOUND;
}
CMFS_closedir(dirp);
return NOT_FOUND;
```

NAME

CMFS_rename - Renames a file.

SYNTAX

C Syntax

```
n = CMFS_rename(oldname, newname)
char *oldname, *newname;
int n;
```

Fortran Syntax

```
n = CMFS_RENAME (oldname, newname)
INTEGER n
STRING oldname, newname
```

ARGUMENTS

oldname	Character string. The name of the file to be renamed.
newname	Character string. The file's new name.

RETURN VALUE

0	Indicates successful completion.
-1	Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_rename renames the file (or directory) named oldname to newname. If newname already exists, then it is first removed. Both oldname and newname must be of the same type (that is, both directories or both non-directories), and must reside on the same file system. If newname is an existing directory, it must be empty.

CMFS_rename guarantees that an instance of newname will always exist, even if the system should crash in the middle of the operation.

The system can deadlock if a loop in the file system graph is present. Suppose an entry in directory a, say a/file1, is a hard link to directory b, and an entry in directory b, say b/file2, is a hard link to directory a. If two separate processes attempt to perform rename a/file1 b/file2 and rename b/file2 a/file1, respectively, the system may deadlock attempting to lock both directories for modification.

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_rename fails if any of the following are true:

CMFS_ENOTDIR	A component of the path prefix of either oldnam e or newname is not a directory.
CMFS_ENAMETOOLONG	Oldname or newname exceeded 255 characters in length.
CMFS_ENOENT	A component of the path prefix of either oldname or newname does not exist.
CMFS_ENOENT	The file named by oldname does not exist.
CMFS_EACCES	A component of the path prefix of either oldname or newname denies search permission.

CMFS_EACCES	The requested rename operation requires writing in a directory with a mode that denies write permission.
CMFS_EXDEV	The link named by newname and the file named by oldname are on different logical devices (file systems).
CMFS_ENOSPC	The directory in which the entry for the new name is being placed cannot be extended because there is no space left on the file system containing the directory.
CMFS_EDQUOT	Quotas are not supported.
CMFS_EIO	An I/O error occurred while reading oldname or writing to the file system.
CMFS_EROFS	The requested rename requires writing in a directory on a read-only file system.
CMFS_EINVAL	oldname is a parent directory of newname, or an attempt is made to rename . (the current directory) or .. (the parent directory).
CMFS_ENOTEMPTY	newname is a directory and is not empty.
CMFS_EBUSY	Mount points are not supported.

SEE ALSO

CMFS_open

NAME

CMFS_rmdir - Removes a directory.

SYNTAX

C Syntax

```
n = CMFS_rmdir(path)
char *path;
int n;
```

Fortran Syntax

```
n = CMFS_RMDIR (path)
INTEGER n
STRING path
```

ARGUMENTS

path Character string. The directory to be removed.

RETURN VALUE

0 Indicates successful completion.

-1 Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_rmdir removes the directory whose name is path. The directory must be empty; it must not have any entries other than . (the current directory) and .. (the parent directory).

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_rmdir fails if any of the following are true:

CMFS_ENOTEMPTY	The named directory contains files other than . (the current directory) and .. (the parent directory) in it.
CMFS_EPERM	The directory containing the directory to be removed is marked sticky, and neither the containing directory nor the directory to be removed are owned by the effective user ID.
CMFS_ENOTDIR	A component of the path is not a directory.
CMFS_ENOENT	The named directory does not exist.
CMFS_EACCES	Search permission is denied for a component of the path prefix.
CMFS_EACCES	Write permission is denied on the directory containing the link to be removed.
CMFS_EBUSY	The directory to be removed is the mount point for a mounted file system.
CMFS_EROFS	The directory entry to be removed resides on a read-only file system.
CMFS_EINVAL	The pathname contains a character with the high-order bit set.
CMFS_ENAMETOOLONG	Path exceeded 255 characters in length.
CMFS_EIO	An I/O error occurred while deleting the directory entry or deallocating the inode.

CMFS_ETIMEDOUT	A "connect" request or remote file operation failed because the connected party did not properly respond after a period of time that is dependent on the communications protocol.

SEE ALSO

CMFS_mkdir
CMFS_unlink

NAME

CMFS_scandir - Scans a directory.

SYNTAX

C Syntax

```
#include <sys/types.h>
#include <cm/cm_dir.h>

n = CMFS_scandir(directory, namelist, select, compare)
char *directory;
struct cm_direct *(*namelist[ ]);
int n, (*select)();
int (*compare)();

alphasort(d1, d2)
struct direct **d1, **d2;
```

ARGUMENTS

directory	Character string. The directory to be scanned.
namelist	A pointer to an array of structure pointers.
select	Integer. A pointer to a user-supplied routine that is called by CMFS_scandir to select directory entries.
compare	Integer. A pointer to a user-supplied routine that is passed to qsort(3) to sort directory entries.

RETURN VALUE

n	The number of entries in the array. Through the argument namelist, a pointer to the array is also returned.
-1	Indicates the directory cannot be opened for reading, or malloc (3) cannot allocate enough memory to hold all the data structures.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_scandir reads the directory directory and builds an array of pointers to directory entries using malloc(3). It returns the number of entries in the array and a pointer to the array through namelist.

Select is a pointer to a user-supplied routine that is called by CMFS_scandir to select the directory entries that are to be included in the array. It is called with a pointer to a directory entry and should return a non-zero value if the directory entry is to be included in the array. If select is NULL all the directory entries are included.

Compare is a pointer to a user-supplied routine that is passed to qsort(3) to sort the completed array. If this pointer is NULL the array is not sorted. The routine alphasort can be used to sort the array alphabetically.

The memory allocated for the array can be deallocated with free (see malloc(3)) by freeing each pointer in the array and the array itself.

ERRORS

A return value of -1 indicates that either the directory cannot be opened for reading, or malloc(3) cannot allocate enough memory to hold all the data structures.

CMFS_scandir (3)

SEE ALSO

directory operations
malloc(3)
qsort(3)
dir(5))

NAME

Directory Operations - Open, read, seek, tell the location in, and close directories.

C SYNTAX

```
#include <cm/cm_dir.h>

dirp = *CMFS_opendir(pathname)
char *pathname;
CMDIR *dirp;

entp = CMFS_readdir(dirp)
CMDIR *dirp;
struct cm_direct *entp;

n = CMFS_telldir(dirp)
long n;
CMDIR *dirp;

void CMFS_seekdir(dirp, loc)
CMDIR *dirp;
long loc;

n = CMFS_closedir(dirp)
CMDIR *dirp;
int n;
```

FORTRAN SYNTAX

Not supported.

ARGUMENTS

pathname	Character string. The pathname of the directory to be opened.
dirp	A pointer to identify the directory stream. dirp is returned by CMFS_opendir and used in the other directory operations.
loc	Integer. Returned by an earlier call to CMFS_telldir, this argument to CMFS_seekdir sets the position of the next CMFS_readdir. A loc of 0 sets the position to the beginning of the directory stream.
entp	A pointer to the next directory entry. entp is returned by CMFS_readdir.

RETURN VALUES

CMFS_opendir

dirp	Pointer to identify the directory stream.
NULL	Indicates the directory cannot be accessed or insufficient memory is available to open it.

CMFS_readdir

entp	Pointer to the next directory entry.
NULL	Indicates the end of the directory has been reached or an invalid seekdir operation has been detected.

CMFS_telldir

n	Current location associated with the directory stream.

CMFS_seekdir

None.

CMFS_closdir

0	Indicates successful close
-1	Indicates an error; CMFS_errno is set.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_opendir opens the directory named by pathname and associates a directory stream with it. CMFS_opendir returns a pointer to identify the directory stream in subsequent operations. The pointer NULL is returned if the specified filename can not be accessed, or if insufficient memory is available to open the directory file.

CMFS_readdir returns a pointer to the next directory entry. It returns NULL or nil upon reaching the end of the directory or upon detecting an invalid seek operation. CMFS_readdir uses the UNIX get-directories system call to read directories.

CMFS_telldir returns the current location associated with the named directory stream. Values returned by CMFS_telldir are good only for the lifetime of the DIR pointer from which they are derived. If the directory is closed and then reopened, the CMFS_telldir value may be invalidated due to undetected directory compaction.

CMFS_seekdir sets the position of the next CMFS_readdir operation on the directory stream. Only values returned by CMFS_telldir should be used with CMFS_seekdir.

CMFS_closedir closes the named directory stream. If the close is successful, 0 is returned; if the close fails, -1 is returned and CMFS_errno is set. All resources associated with this directory stream are released.

The directory stream that CMFS_opendir associates with pathname contains an entry for each directory residing in pathname. Each directory entry has associated with it four pieces of information. This information is stored in a directory structure having four elements. A pointer to this structure is returned by CMFS_readdir. The four elements are:

cmd_ino	Inode number
cmd_reclen	Directory record length
cmd_namlen	Directory name length
cmd_name	Directory name entry

In C, the content of an element is referenced by pointer -> element-name. For example, if dir_ent_ptr is the pointer returned by CMFS_readdir, the directory name entry is referenced by dir_ent_ptr -> d_name.

EXAMPLE

The following sample code searches a directory for the entry name.

```
len = strlen(name);
dirp = CMFS_opendir(.);
for (dp = CMFS_readdir(dirp); dp != NULL;
    dp = CMFS_readdir(dirp))
if (dp->cmd_namlen == len && !strcmp(dp->cmd_name, name)) {
  CMFS_closedir(dirp)
  return FOUND;
}
CMFS_closedir(dirp);
return NOT_FOUND;
```

NAME

CMFS_serial_lseek, CMFS_lseek - Moves the pointer associated with an open file.
Currently **CMFS_serial_lseek** and **CMFS_lseek** are functionally equivalent. Since a future software release may provide parallel seeks via **CMFS_lseek**, we recommend that programs use only **CMFS_serial_lseek** at this time.

C SYNTAX

```
#include <cm/cm_file.h>

n = CMFS_[serial_]lseek(fd, offset, whence)
long n, offset;
int fd, whence;
```

FORTRAN SYNTAX

```
n = CMFS_[SERIAL_]LSEEK (fd, offset, whence)
INTEGER n, fd, offset, whence
```

ARGUMENTS

fd	File descriptor, which was returned by a previous call to CMFS_creat or CMFS_open of a serial file open for reading or writing.
offset	Integer. The number of bytes to offset the file pointer from the position specified by whence.
whence	Integer (values defined in <cm/cm_file.h>): CMFS_L_SET sets the pointer to offset bytes. CMFS_L_INCR sets the pointer to the current position plus offset bytes. CMFS_L_XTND sets the pointer to the length of the file plus offset bytes.

RETURN VALUE

n	The current file pointer value. This value is measured in bytes from the beginning of the file. The first byte is byte 0.
-1	Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_[serial_]lseek moves the file pointer associated with a file open for reading or writing, so the next data item read or written need not be logically adjacent to the item used in the previous operation. The argument fd represents the file open for reading or writing. CMFS_[serial_]lseek sets the file pointer to the position specified by offset and whence.

Seeking far beyond the end of a file, then writing, causes file blocks to be allocated in the gap between the end of the file and the location at which the write proceeds. The contents of these blocks are undefined, containing whatever was left there by previous uses of those blocks.

Upon successful completion, the current file pointer value is returned. This pointer is measured in bytes from the beginning of the file, where the first byte is byte 0.

RESTRICTIONS

Some devices are incapable of seeking. The value of the pointer associated with such a device is undefined.

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_[serial_]lseek fails if any of the following are true:

CMFS_EBADF
: fd is not an open file descriptor.

CMFS_EINVAL
: whence is not a proper value.

SEE ALSO

CMFS_open
CMFS_serial_truncate_file

NAME

CMFS_serial_read_file -

Reads from a file or a connected socket into the memory of a serial computer that supports the CMFS library.

C SYNTAX

```
bytes = CMFS_serial_read_file(fd, buffer, count)
int bytes, fd, count;
char *buffer
```

FORTRAN SYNTAX

```
bytes = CMFS_SERIAL_READ_FILE (fd, buffer, count)
INTEGER bytes, fd, buffer(1), count
```

Note: The actual size of the buffer array is count/4

ARGUMENTS

fd	File descriptor (returned by a previous call to CMFS_creat or CMFS_open) of a file open for reading, or socket descriptor (returned by a previous call to CMFS_socket or CMFS_accept) of a connected socket.
buffer	A pointer to a buffer in C; an integer in Fortran. The location in the memory of the serial computer at which to place the data read from the file or socket.
count	Integer. The number of bytes to read from the file identified by fd.

RETURN VALUE

bytes	The number of bytes read and placed in the buffer of the serial computer.
-1	Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_serial_read_file attempts to read data from a CMFS file or a connected socket into the memory of a serial computer that supports the CMFS library and is connected to the rest of the CM system across Ethernet (for example, a CM-IOP or a VMEIO host computer). These reads completely bypass the CM. If a CMIO bus connects the destination machine's memory to the system where the file resides (or, if a socket is being read, to the CM-HIPPI), the data transfer occurs over the CMIO bus; otherwise, the data transfer occurs over the Ethernet, albeit slowly.

CMFS_serial_read_file attempts to read count bytes of information from the file or connected socket referenced by fd into buffer. In C, buffer is a byte address in the serial computer's memory. The value returned is the number of bytes read. If fewer bytes remain to be read than the number requested, the remaining bytes are read until the end-of-file is encountered, or until all remaining data has been read from the socket. The routine returns the number of bytes actually read (as opposed to the number requested) If an attempt to read is made after end-of-file has been encountered, zero is returned. If the sending program has closed or shut down its socket, the remaining data is read; if no more data remains to be read, zero is returned.

When a file is read, the read starts at a position given by the pointer associated with fd (see CMFS_serial_lseek). Upon return from CMFS_serial_read_file, the pointer is incremented by the number of bytes actually read.

CM-2/200 files read using CMFS_serial_read_file may be in parallel format. If they are, when the file is subsequently used by a serial computer, the data will be in an inappropriate order. To ensure that the

file will be able to be used by a serial computer, before calling CMFS_serial_read_file:

Call CMFS_read_file(_always) to read the file into the CM-2/200.

Call CMFS_transpose_always or CMFS_transpose_record_always on a CM-2/200 to rearrange the data for use by a serial machine.

Call CMFS_write_file(_always) to write the data back to the file (in serial order).

Similarly, when you read data into a serial computer from a CM-2/200 socket, you must ensure that the data is arranged correctly in the serial computer's memory. Call CMFS_transpose(_record)_always on a CM-2/200 to transpose the data to serial format if necessary.

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_serial_read fails if any of the following are true:

CMFS_EINVAL Reading was attempted with a current file position that was negative.

CMFS_EBADF fd is not a valid file descriptor open for reading.

CMFS_EIO An I/O error occurred while reading from the file system.

CMFS_ECONNABORTED
The socket connection has been aborted by the underlying TCP protocol.

CMFS_ECONNRESET The socket connection has been reset by the underlying TCP protocol.

CMFS_ENOTCONN If fd is a socket descriptor, the socket is not connected. If fd is the descriptor of the CM- HIPPI device, the connection to the remote system has been broken or has not yet been established.

CMFS_ETIMEDOUT A connection request or remote file operation failed because the connected party did not properly respond after a period of time that is dependent on the communications protocol. If fd is the descriptor of the CM-HIPPI device, the CM-HIPPI timed out waiting for data or for the remote system to issue a connection request.

CMFS_EHIPPI_SRC_PARITY
A parity error was detected on the CM-HIPPI source board.

CMFS_EHIPPI_DST_PARITY
A parity error was detected on the CM-HIPPI destination board.

CMFS_EHIPPI_DST_PARITY
A parity error was detected on a CM-HIPPI IOP board.

SEE ALSO

CMFS_open CMFS_serial_truncate_file
CMFS_read_file(_always) CMFS_READ_AND_TRANSPOSE_ALWAYS (CM-2/200 only)
CMFS_socket CMFS_twuffle_to_serial_order_always_1l
CMFS_serial_lseek CMFS_twuffle_from_serial_order_always_1L
CMFS_serial_write_file

NAME

CMFS_serial_truncate_file - Truncates or extends a file.

C SYNTAX

```
n = CMFS_serial_truncate_file(path, length)
char *path;
long length;
int n;

n = CMFS_serial_ftruncate_file(fd, length)
int fd;
long length;
int n;
```

FORTRAN SYNTAX

```
n = CMFS_SERIAL_TRUNCATE_FILE (path, length)
INTEGER n, length
STRING path

n = CMFS_SERIAL_FTRUNCATE_FILE (fd, length)
INTEGER n, fd, length
```

ARGUMENTS

path	Character string. The file whose length is to be changed.
length	The new length, in bytes, of the file named by path.
fd	File descriptor, which was returned by a previous call to CMFS_creat or CMFS_open.

RETURN VALUE

0	Indicates successful completion.
-1	Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_serial_truncate_file and CMFS_serial_ftruncate_file change the size of the file named by path, or referenced by fd, respectively, to length bytes. If the file was previously larger than this size, the extra data is lost. If the file was previously smaller than this size, the file is extended to length bytes. (This allows you to pre-allocate physical disk blocks in as contiguous a manner as possible.)
CMFS_serial_ftruncate_file requires the file to be open for writing.

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_serial_truncate_file fails if any of the following are true:

CMFS_ENOTDIR
: A component of the path prefix is not a directory.

CMFS_ENOENT
: The named file does not exist.

CMFS_EACCES
: Search permission is denied for a component of the path prefix.

CMFS_EISDIR
: The named file is a directory.

CMFS_EROFS

The named file resides on a read-only file system.

CMFS_ETXTBSY
The file is a pure procedure (shared text) file that is being executed.

CMFS_EINVAL
The pathname contains a character with the high-order bit set.

CMFS_ENAMETOOLONG
Path exceeded 255 characters in length.

CMFS_EIO An I/O error occurred updating the inode.

CMFS_ENOSPC
An error occurred while attempting to allocate disk blocks.

The library call CMFS_serial_ftruncate_file fails if any of the following are true:

CMFS_EBADF
fd is not a valid descriptor.

CMFS_EINVAL
fd references a socket, not a file.

CMFS_ETIMEDOUT
A "connect" request or remote file operation failed because the connected party did not properly respond after a period of time that is dependent on the communications protocol.

SEE ALSO

CMFS_open
CMFS_serial_lseek
CMFS_serial_read_file
CMFS_serial_write_file

CMFS_serial_write_file (CMFS)

NAME

CMFS_serial_write_file -

Writes to a file or a connected socket from the memory of a serial computer that supports the CMFS library.

C SYNTAX

```
bytes = CMFS_serial_write_file(fd, buffer, count)
int bytes, fd, count;
char *buffer;
```

FORTRAN SYNTAX

```
bytes = CMFS_SERIAL_WRITE_FILE (fd, buffer, count)
INTEGER bytes, fd, buffer(1), count
```

Note: The actual size of the buffer array is count/4

ARGUMENTS

fd	File descriptor (returned by a previous call to CMFS_creat or CMFS_open) of a file open for writing, or socket descriptor (returned by a previous call to CMFS_socket or CMFS_accept.
buffer	A pointer to a buffer in C; an integer in Fortran. The location in the serial computer's memory from which to write data to the file or socket identified by fd.
count	Integer. The number of bytes to write into the file identified by fd.

RETURN VALUE

bytes	The number of bytes written from the buffer on the serial computer to fd.
-1	Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_serial_write_file attempts to write data into a CMFS file or to a connected socket from the memory of a serial computer that supports the CMFS library and that connects to the rest of the CM system across Ethernet (for example, a front-end computer, a CM-IOP, or a VMEIO host computer). These write operations completely bypass the CM. If a socket is not ready for writing, the CMFS_serial_write_file waits for it to become ready.

If a CMIO bus connects the source system's memory to the system where the file resides (or, in the case of a socket, to the CM-HIPPI), the data transfer occurs over the CMIO bus; otherwise, the data transfer occurs over the Ethernet, albeit slowly.

CMFS_serial_write_file attempts to write count bytes of information from the buffer pointed to by buffer to the file or socket referenced by fd. buffer is a byte address in the serial computer's memory. (To obtain decent performance when writing to a DataVault, buffer should contain several megabytes.) The value returned is the number of bytes written.

When a file is written, the write starts at a position given by the file pointer associated with the file descriptor fd. When CMFS_serial_write_file returns, this file pointer is incremented by the number of bytes actually written.

Data written using CMFS_serial_write_file is written to a CMFS file in serial format. Consequently, when the file is subsequently read into the CM-2/200 using CMFS_read_file(_always), the data is placed in the CM-2/200 processors in an order that, although predictable, is not desired. To arrange the order

CMFS_serial_write_file (CMFS)

appropriately, call CMFS_transpose_always or CMFS_transpose_record_always on the CM-2/200 after reading the data into the CM-2/200.

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_serial_write_file fails the file pointer remains unchanged if any of the following are true:

CMFS_EBADF	fd is not a valid descriptor open for writing.
CMFS_ENOSPC	There is no free space remaining on the file system containing the file.
CMFS_EIO	An I/O error occurred while reading from or writing to the file system.
CMFS_EINVAL	count is negative or the current file pointer position is negative.
CMFS_ENOTCONN	If fd is a socket descriptor, the socket is not connected. If fd is the descriptor of the CM-HIPPI device, the connection to the remote system has been broken or has not yet been established.
CMFS_ETIMEDOUT	A "connect" request or remote file operation failed because the connected party did not properly respond after a period of time that is dependent on the communications protocol. If fd is the descriptor of the CM-HIPPI device, the CM-HIPPI timed out waiting for data or for the remote system to respond to a connection request.
CMFS_EHIPPI_SRC_PARITY	A parity error was detected on the CM-HIPPI source board.
CMFS_EHIPPI_DST_PARITY	A parity error was detected on the CM-HIPPI destination board.
CMFS_EHIPPI_DST_PARITY	A parity error was detected on a CM-HIPPI IOP board.

SEE ALSO

CMFS_open
CMFS_serial_lseek
CMFS_serial_read_file
CMFS_serial_truncate_file
CMFS_socket
CMFS_transpose_always (CM-2/200 only)
CMFS_transpose_record_always (CM-2/200 only)
CMFS_write_file
CMFS_TRANSPOSE_AND_WRITE_ALWAYS (CM-2/200 only)
CMFS_twuffle_to_serial_order_always_1l
CMFS_twuffle_from_serial_order_always_1L

NAME

CMFS_set_debug_mode - Activates/deactivates automatic CMFS debugging messages.

C SYNTAX

```
void
CMFS_set_debug_mode (val)
int val;
```

FORTRAN SYNTAX

```
SUBROUTINE CMFS_SET_DEBUG_MODE (val)
INTEGER val
```

ARGUMENTS

val	Integer. If val is non-zero, the automatic printing of CMFS debugging messages is activated. If val is zero, the automatic printing of CMFS debugging messages is deactivated.

RETURN VALUE

None

CM STATE CHANGE

None.

DESCRIPTION

The setting of CMFS_set_debug_mode activates or deactivates the automatic printing of CMFS debugging messages. The CMFS_set_debug_mode routine is especially useful for Symbolics users, since they cannot use the environment variable CMFS_DEBUG.

CMFS_set_debug_mode is also useful to those users who want to explicitly control the portions of their code for which automatic CMFS debugging message printing is enabled: when CMFS_set_debug_mode 1 is called, debugging messages are printed for all code that executes before CMFS_set_debug_mode 0 is called.

SEE ALSO

CMFS_DEBUG (environment variable)

NAME

CMFS_statfs - Obtains file system statistics.

C SYNTAX

```
#include <cm/cm_mount.h>

n = CMFS_statfs (path, buffer)
char *path;
struct cm_statfs *buffer;
int n;

struct cm_statfs {
  long cmf_type;    /* type of info, zero for now            */
  long cmf_bsize;   /* fundamental file system block size    */
  long cmf_blocks;  /* total blocks in file system           */
  long cmf_bfree;   /* free blocks                           */
  long cmf_bavail;  /* free blocks available to non-superuser */
  long cmf_files;   /* total 'inodes' in the file system     */
  long cmf_ffree;   /* free 'inodes' in the file system      */
  long cmf_stripe;  /* striping factor                       */
  long cmf_spare[8]; /* reserved                             */
};
```

FORTRAN SYNTAX

```
n = CMFS_STATFS (path, buffer)
INTEGER n, buffer(16)
STRING path
```

ARGUMENTS

path	Character string. The path name of any file within the mounted file system.
buffer	A pointer to a CMFS_statfs structure in C; an array of 16 integers in Fortran. The structure that receives the status information about the mounted file system.

RETURN VALUE

0	Indicates successful completion.
-1	Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

value

CM STATE CHANGE

None.

DESCRIPTION

CMFS_statfs places up-to-date information about a mounted file system into a file system data structure indicated by buffer. path is the pathname of any file within the mounted file system.

In C, the argument buffer is a pointer to a cm_statfs structure.

The elements in the file system data structure are described below. Fields that are undefined for a file system are set to -1. In C, the contents of an element are referenced by structure-name.cmst_element-name.

type The type of information.
bsize The fundamental file system block size.
blocks The total number of blocks in the file system.
bfree The number of free blocks in the file system.

bavail The number of free blocks available to users other than the superuser.
files The total number of inodes in the file system.
ffree The number of free inodes in the file system.
stripe The number of DataVaults in the striped file system.

EXAMPLE

```
(setq fs-info (cmfs:make-statfs)) ; Generate a statfs struct
(cmfs:statfs "." fs-info)         ; Fill in the structure
(cmfs:statfs-bfree fs-info)       ; Return number of free
                                  ; blocks
```

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_statfs fails if any of the following are true:

CMFS_ENOTDIR
A component of the path prefix of path is not a directory.

CMFS_EINVAL
path contains a character with the high-order bit set.

CMFS_ENAMETOOLONG
path exceeded 255 characters in length.

CMFS_ENOENT
The file referred to by path does not exist.

CMFS_EACCESS
Search permission is denied for a component prefix of path.

CMFS_EFAULT
buffer or path points to an invalid address.

CMFS_EIO An I/O error occurred while reading from, or writing to, the file system.

NAME

Directory Operations - Open, read, seek, tell the location in, and close directories.

C SYNTAX

```
#include <cm/cm_dir.h>

dirp = *CMFS_opendir(pathname)
char *pathname;
CMDIR *dirp;

entp = CMFS_readdir(dirp)
CMDIR *dirp;
struct cm_direct *entp;

n = CMFS_telldir(dirp)
long n;
CMDIR *dirp;

void CMFS_seekdir(dirp, loc)
CMDIR *dirp;
long loc;

n = CMFS_closedir(dirp)
CMDIR *dirp;
int n;
```

FORTRAN SYNTAX

Not supported.

ARGUMENTS

pathname	Character string. The pathname of the directory to be opened.
dirp	A pointer to identify the directory stream. dirp is returned by CMFS_opendir and used in the other directory operations.
loc	Integer. Returned by an earlier call to CMFS_telldir, this argument to CMFS_seekdir sets the position of the next CMFS_readdir. A loc of 0 sets the position to the beginning of the directory stream.
entp	A pointer to the next directory entry. entp is returned by CMFS_readdir.

RETURN VALUES

CMFS_opendir

dirp	Pointer to identify the directory stream.
NULL	Indicates the directory cannot be accessed or insufficient memory is available to open it.

CMFS_readdir

entp	Pointer to the next directory entry.
NULL	Indicates the end of the directory has been reached or an invalid seekdir operation has been detected.

CMFS_telldir

n	Current location associated with the directory stream.

CMFS_seekdir

None. CMFS_closdir

0	Indicates successful close
-1	Indicates an error; CMFS_errno is set.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_opendir opens the directory named by pathname and associates a directory stream with it. CMFS_opendir returns a pointer to identify the directory stream in subsequent operations. The pointer NULL is returned if the specified filename can not be accessed, or if insufficient memory is available to open the directory file.

CMFS_readdir returns a pointer to the next directory entry. It returns NULL or nil upon reaching the end of the directory or upon detecting an invalid seek operation. CMFS_readdir uses the UNIX get-directories system call to read directories.

CMFS_telldir returns the current location associated with the named directory stream. Values returned by CMFS_telldir are good only for the lifetime of the DIR pointer from which they are derived. If the directory is closed and then reopened, the CMFS_telldir value may be invalidated due to undetected directory compaction.

CMFS_seekdir sets the position of the next CMFS_readdir operation on the directory stream. Only values returned by CMFS_telldir should be used with CMFS_seekdir.

CMFS_closedir closes the named directory stream. If the close is successful, 0 is returned; if the close fails, -1 is returned and CMFS_errno is set. All resources associated with this directory stream are released.

The directory stream that CMFS_opendir associates with pathname contains an entry for each directory residing in pathname. Each directory entry has associated with it four pieces of information. This information is stored in a directory structure having four elements. A pointer to this structure is returned by CMFS_readdir. The four elements are:

cmd_ino	Inode number
cmd_reclen	Directory record length
cmd_namlen	Directory name length
cmd_name	Directory name entry

In C, the content of an element is referenced by pointer -> element-name. For example, if dir_ent_ptr is the pointer returned by CMFS_readdir, the directory name entry is referenced by dir_ent_ptr -> d_name.

EXAMPLE

The following sample code searches a directory for the entry name.

```
len = strlen(name);
dirp = CMFS_opendir(.);
for (dp = CMFS_readdir(dirp); dp != NULL;
    dp = CMFS_readdir(dirp))
if (dp->cmd_namlen == len && !strcmp(dp->cmd_name, name)) {
  CMFS_closedir(dirp)
  return FOUND;
}
CMFS_closedir(dirp);
return NOT_FOUND;
```

NAME

CMFS_unlink - - Removes a file from a directory.

C SYNTAX

```
n = CMFS_unlink(path)
int n;
char *path;
```

FORTRAN SYNTAX

```
n = CMFS_UNLINK (path)
INTEGER n
STRING path
```

ARGUMENTS

path	Character string. The directory entry to be unlinked.

RETURN VALUE

0	Indicates successful completion.
-1	Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_unlink removes the entry for the file path from its directory. If this entry was the last link to the file and no process has the file open, then the file is deleted and all resources associated with the file are reclaimed. If, however, the file was open in any process, the actual resource reclamation is delayed until it is closed, even though the directory entry has disappeared.

ERRORS

If an error occurs, the value -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_unlink fails if any of the following are true:

CMFS_ENOTDIR
: A component of the path prefix is not a directory.

CMFS_ENOENT
: The named file does not exist.

CMFS_EACCES
: Search permission is denied for a component of the path prefix.

CMFS_EACCES
: Write permission is denied on the directory containing the link to be removed.

CMFS_EROFS
: The named file resides on a read-only file system.

CMFS_EINVAL
: The pathname contains a character with the high-order bit set.

CMFS_ENAMETOOLONG
: path exceeded 255 characters in length.

CMFS_EIO An I/O error occurred while deleting the directory entry or deallocating the inode.

CMFS_ETIMEDOUT
: A connect" request or remote file operation ailed because the connected party did not properly respond after a period of time that is dependent on the communications protocol.

SEE ALSO

CMFS_link
CMFS_rmdir

NAME

. CMFS_utimes - - Set file times.

C SYNTAX

```
#include <sys/types.h>
n = CMFS_utimes(path, tvp)
char *path;
int n;
struct timeval *tvp;
```

FORTRAN SYNTAX

Not available.

ARGUMENTS

path — A pointer to the pathname of the file whose access and modification times to check.

tvp — An indication of how to set the access and modification times.

access-seconds
: An integer indicating the seconds portion of the access time, the seconds since 1/1/70 00:00:00 GMT.

access-usecs — An integer indicating the microseconds portion of the access time.

modification-time
: An integer indicating the seconds portion of the access time (seconds since 1/1/70 00:00:00 GMT).

modification-usecs
: An integer indicating the microseconds portion of the modification time.

RETURN VALUE

n

0 — Indicates success.

-1 — Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_utimes sets the access and modification times of the file named by path. If tvp is NULL, the access and modification times are set to the current time; a process must be the owner of the file or have write permission for the file to use CMFS_utimes in this manner.

If tvp is not NULL, it is assumed to point to an array of two timeval structures. The access time is set to the value of the first member, and the modification time is set to the value of the second member. Only the owner of the file or the super-user may use CMFS_utimes in this manner. In either case, the inode-changed time of the file is set to the current time.

ERRORS

If an error occurs, -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_utimes fails if any of the following are true:

CMFS_EACCES
: Search permission is denied for any component of the path name.

CMFS_EACCES
: The effective user ID of the process is not super-user and not the owner of the file, write permission is denied for the file, and tvp is NULL.

CMFS_ENOTDIR

A component of the path prefix is not a directory.

CMFS_ENOENT
The file referred to by path does not exist.

CMFS_EFAULT
path or tvp points outside the process's allocated address space.

CMFS_EIO An I/O error occurred while reading from or writing to the file system.

CMFS_EROFS
The file system containing the file is mounted read-only.

CMFS_ELOOP
Too many symbolic links were encountered in translating path.

CMFS_EPERM
The effective user ID of the process is not super-user and not the owner of the file, and tvp is not NULL.

SEE ALSO

CMFS_stat

NAME

CMFS_vmeio_allocate - - Allocates DRAM on a VMEIO host computer.

C SYNTAX

```
n = CMFS_vmeio_allocate(length, unit)
char *n;
int length, unit;
```

FORTRAN SYNTAX

Not available.

ARGUMENTS

length Integer. The number of contiguous bytes of DRAM to allocate.

unit The VMEIO board from which to allocate the DRAM.

RETURN VALUE

n For success, a pointer to the allocated buffer.

NULL pointer
Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_vmeio_allocate attempts to allocate length bytes of contiguous VMEIO DRAM on the VMEIO board specified by unit. unit is a small integer, usually 0 or 1, but possibly also 2 or 3. See your system administrator of your site's Thinking Machines Corporation application engineer for the unit number of the VMEIO board on the same bus as the CMFS device to which you are writing or from which you are reading.

ERRORS

If an error occurs, a NULL pointer is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_vmeio_allocate fails if any of the following are true:

CMFS_ENODEV
The requested VMEIO board was not found.

CMFS_ENOBUFS
The requested buffer space was not available.

SEE ALSO

CMFS_vmeio_free

NAME

CMFS_vmeio_free - Frees a buffer on a VMEIO host computer.

C SYNTAX

```
n = CMFS_vmeio_free(buffer, unit)
int n;
int unit;
char *buffer;
```

FORTRAN SYNTAX

Not available.

LISP SYNTAX

Not available.

ARGUMENTS

buffer	A pointer to the VMEIO memory which to free.
unit	The VMEIO board on which the memory resides.

RETURN VALUE

n	0 indicates success.
-1	Indicates an error occurred; the external variables CMFS_errno and errno are set to indicate the cause of the error.

CM STATE CHANGE

None.

DESCRIPTION

CMFS_vmeio_free frees the previously allocated memory pointed to by buffer on the VMEIO board represented by unit.

ERRORS

If an error occurs, -1 is returned and the external variables CMFS_errno and errno are set to indicate the cause of the error. CMFS_vmeio_free fails if any of the following are true:

CMFS_ENODEV
: The requested VMEIO board was not found.

CMFS_EINVAL
: Invalid buffer argument for unit.

SEE ALSO

CMFS_vmeio_allocate

NAME

CMFS_write_file[_always] - Writes to a file or a connected socket from CM-5 processing nodes.

SYNTAX

C SYNTAX

Not supported

C* SYNTAX

n = **CMFS_write_file[_always]**(*fd*, *pvar**, *count*)
int *n*, *fd*, *count*;
void * *pvar*;

FORTRAN SYNTAX

Not supported

CM FORTRAN SYNTAX

N = **CMFS_WRITE_FILE[_ALWAYS]**(*FD*, *ARRAY*, *COUNT*)
INTEGER *N*, *FD*, *COUNT*
ANY_TYPE ARRAY(:,:,...,:) ! any type, any rank

ARGUMENTS

fd File descriptor (returned by a previous call to **CMFS_creat** or **CMFS_open**) of a file open for writing, or socket descriptor (returned by a previous call to **CMFS_socket** or **CMFS_accept**) of a connected socket.

*pvar** A pointer to a parallel variable of any type from which to write CM-5 data.

ARRAY (CMF)
An array (of any type or rank) from which to write the data.

count Integer. The number of bytes to write from each element of the parallel variable or array. *count* can be any number of bytes, but performance is diminished when *count* is not a multiple of 4 bytes (the CM-5 word size).

RETURN VALUE

n The total number of bytes written from the parallel variable or array. **-1** Indicates an error occurred; the external variables **CMFS_errno** and **errno** are set to indicate the cause of the error.

CM STATE CHANGE

None.

CONTEXT

CMFS_write_file: Conditional, occurs in selected PNs.
CMFS_write_file_always: Unconditional, occurs in all PNs.
Currently both **CMFS_write_file** and **CMFS_write_file_always** operate unconditionally. Since a future software release may support conditional writes via **CMFS_write_file**, we recommend that programs use only **CMFS_write_file_always** at this time.

DESCRIPTION

CMFS_write_file and **CMFS_write_file_always** attempt to write *count* bytes of information from each element of the parallel variable pointed to by the *pvar* argument (for C*) or each element of the array *ARRAY* (in CM Fortran) to the object represented by *fd*; *fd* is either a file descriptor returned by a previous call to **CMFS_creat** or **CMFS_open**, or a socket descriptor returned by a previous call to

CMFS_socket or **CMFS_accept**. To write to a socket, you must use **CMFS_write_file_always**, as writing to a socket is unconditional. If a socket is not ready for writing, **CMFS_write_file_always** waits for it to become ready.

When a file is written, the write starts at a position given by the file pointer associated with *fd* (see **CMFS_serial_lseek** (3)). When the write call returns, the file pointer is incremented by *count* bytes. Upon successful completion, **CMFS_write_file** and **CMFS_write_file_always** return the number of bytes written.

RESTRICTIONS

Writing to a file requires that the write begin on a cmword boundary, where a cmword is usually defined as 4096 bits. You will receive an error if you attempt to write starting at an illegal position.

ERRORS

If an error occurs, the value **-1** is returned and the external variables **CMFS-errno** and **errno** are set to indicate the cause of the error. **CMFS_read_file** and **CMFS_read_file_always** fail if any of the following are true:

CMFS_EBADF
: *fd* is neither a valid file descriptor open for writing, nor a valid socket descriptor.

CMFS_EINVAL
: The current file position is negative.

CMFS_EIO
: An I/O error occurred while writing to the file system. This error could occur if a disk drive failed and a second DataVault disk drive failed before the system administrator corrected the first failure.

CMFS_ESTRIPE_NOTAVAIL
: One or more of the file servers serving the striped CM file system is not running.

CMFS_ENOSPC
: There is no free space remaining on the file system containing the file.

CMFS_EDQUOT
: Quotas are not supported.

CMFS_ECONNABORTED
: The socket connection has been aborted by the underlying TCP protocol.

CMFS_ECONNRESET
: The socket connection has been reset by the underlying TCP protocol.

CMFS_ENOTCONN
: If *fd* is a socket descriptor, the socket is not connected. If *fd* is the descriptor of the CM-HIPPI device, the connection to the remote system has been broken or has not yet been established.

CMFS_ETIMEDOUT
: A connection request or remote file operation failed because the connected party did not properly respond after a period of time that is dependent on the communications protocol. If *fd* is the descriptor of the CM-HIPPI device, the CM-HIPPI timed out waiting for data or for the remote system to issue a connection request.

SEE ALSO

CMFS_creat
CMFS_fcntl
CMFS_serial_lseek
CMFS_socket
CMFS_open
CMFS_read_file

CMFS_errno
errno(2) (on Vaxen)
intro(2) (on Suns)