CERN Program Library Long Writeups Q123

FATMEN

Distributed

File and Tape Management System

Version 1.88

Application Software Group

Computing and Networks Division

CERN Geneva, Switzerland

```
CERN Program Library Office
CERN-CN Division
CH-1211 Geneva 23
Switzerland
Tel.  +41 22 767 4951
Fax.  +41 22 767 7155
Bitnet:  CERNLIB@CERNVM
DECnet:  VXCERN::CERNLIB (node 22.190)
Internet:  CERNLIB@CERNVM.CERN.CH
```


Contact Person:	Jamie Shiers /CN	(JAMIE@CERNVM.CERN.CH)
Technical Realization:	Michel Goossens /CN	(GOOSSENS@CERNVM.CERN.CH)

Edition – June 1994

Preliminary remarks

This **Complete Reference** of the FATMEN system (for **F**ile **a**nd **T**ape **M**anagement: **E**xperimental **N**eeds), consists of four parts:

1 An **overview** of the system.
2 A **step by step** tutorial introduction.
3 A **user guide**, describing each command in detail.
4 An **installation and management guide**.

The FATMEN system is implemented on various mainframes and personal workstations. In particular versions exist for IBM VM/CMS, IBM MVS, VAX/VMS and various Unix-like platforms, such as APOLLO, HP/UX, Cray Unicos, Ultrix, IBM RS6000, Silicon Graphics and SUN.

Throughout this manual, commands to be **entered** are `underlined`

Acknowledgements

The FATMEN package has undergone continuous evolution since its first introduction in 1989. During this time many people have contributed to the system, through discussions or by providing code and assistance.

The FATMEN system depends upon a number of other packages, such as the Tape Management System and numerous tape staging subsystems. The help of the authors and maintainers of such systems is gratefully acknowledged.

This document has been produced using LaTeX [1] with the `cernman` style option, developed at CERN. A compressed PostScript file `fatmen.ps.gz`, containing a complete printable version of this manual, can be obtained from any CERN machine by anonymous ftp as follows (commands to be typed by the user are underlined)[1]:

```
ftp asis01.cern.ch
Trying 128.141.201.136...
Connected to asis01.cern.ch.
220 asis01 FTP server (Version 6.10 ...) ready.
Name (asis01:username): anonymous
Password: your_mailaddress
230 Guest login ok, access restrictions apply.
ftp> cd cernlib/doc/ps.dir
ftp> binary
ftp> get fatmen.ps.gz    ! one page per physical page
ftp> get fatmen2.ps.gz   ! two pages per physical page
ftp> quit
```

[1] If your site does not carry the gnu `gzip` utility you can get the uncompressed file by dropping the `.gz` suffix from the `get` command, and also skipping the `binary` specification below.

Related Documents

This document can be complemented by the following documents:

- ORACLE User Guide [2]
- KUIP - Kit for a User Interface Package [3]
- ZEBRA - Data Structure Management System [4]
- CSPACK - Client Server package [5]
- HEPDB - Database management package [6]
- The FATMEN Report - DD/89/15 [7]
- TMS - The CERN Tape Management System (to be published)
- The MUSCLE Report - DD/88/1 [8]
- Computing at CERN in the 1990s [9]

Table of Contents

List of Figures

List of Tables

Part I

FATMEN – Overview

Chapter 1: Introduction

When the FATMEN project was launched, the problems of data handling for the LEP experiments were expected to be enormous: each of the four experiments was expected to have accumulated some 7 Terabytes of data by the end of 1991, when the total number of Z^0 events per LEP experiment was to have reached 10 million[8]. Although the majority of this data was to reside on IBM 3480 cartridges, large disk farms were also required to facilitate data analysis. In addition, the physicists involved came from many different institutes, in Europe and elsewhere. Thus, any management tools that were to be developed had to take into account the distributed, and highly heterogeneous, nature of computing in high energy physics (HEP). With this in mind, the FATMEN committee was formed at the beginning of 1989 to propose and develop solutions to these problems. The committee involved members from the LEP groups, plus major fixed target and collider experiments. The recommendations of this committee are summarised in the FATMEN Report, CERN DD/89/15. From a user (physicist) point of view, the most important features of the proposed system were

- It should be possible to access data in a consistent manner, regardless of the medium on which it is stored, its location, host operating system, etc.
- All data should be accessed via a meaningful name.

Thus, a physicist on an Apollo in the control room of the OPAL experiment at CERN and his colleague logged on to CERNVM would use the same command to access a dataset stored on cartridge in the central tape robot.

1.1 Advantages of using the FATMEN system

Some of the main advantages of using the FATMEN system are listed below.

- Access to data via a meaningful name.
- Consistent access, regardless of host operating system, medium on which data are stored or location.
- Support for multiple copies of data within the network.
- Possibility of adding comments and other user information on datasets.
- No need to know underlying commands, such as STAGE, SETUP etc. with their system dependent syntax.

1.2 The components of the FATMEN system

The FATMEN system uses several different tools and packages. A short description of each of them follows.

ZEBRA - The data structure management system

The data structure management package ZEBRA was developed at CERN in order to overcome the lack of dynamic data structure facilities in FORTRAN, the favourite computer language in high energy physics. It implements the **dynamic creation and modification** of data structures at execution time and their transport to and from external media on the same or different computers, memory to memory, to disk or over the network, at an **insignificant cost** in terms of execution-time overheads.

ZEBRA manages any type of structure, but specifically supports linear structures (lists) and trees. ZEBRA input/output is either by a sequential or direct access method. Two data representations, **native** (no data conversion when transferred to/from the external medium) and **exchange** (a conversion to/from an interchange format is made), allow data to be transported between computers of the same and of different architectures. The direct access package **RZ** can be used to manage hierarchical data bases. In FATMEN this facility is exploited to store the catalogue information in a hierarchical direct access directory structure.

KUIP - The user interface package

The purpose of KUIP (**K**it for a **U**ser **I**nterface **P**ackage) is to handle the dialogue between the user and the application program (FATMEN in our case). It parses the commands input into the system, verifies them for correctness and then hands over control to the relevant action routines.

The syntax for the commands accepted by KUIP is specified using a **Command Definition File** (CDF) and the information provided is stored in a ZEBRA data structure, which is accessed not only during the parsing stage of the command but also when the user invokes the **online help** command. Commands are grouped in a tree structure and they can be **abbreviated** to their shortest unambiguous form. If an ambiguous command is typed, then KUIP responds by showing all the possibilities. **Aliases** allow the user to abbreviate part or the whole of commonly used command and parameters. A sequence of FATMEN commands can be stored in a text file and, combined with flow control statements, form a powerful **macro** facility. With the help of **parameters**, whose values can be passed to the macros, general and adaptable task solving procedures can be developed.

Different **styles of dialogue** (command and various menu modes) are available and the user can switch between them at any time. In order to save typing, **default values**, providing reasonable settings, can be used for most parameters of a command. A **history file**, containing the n most recently entered commands, is automatically kept by KUIP and can be inspected, copied or re-entered at any time. The history file of the last FATMEN session is also kept on disk.

ORACLE - The relational database system

ORACLE is the relational database system chosen by CERN. Versions exist for a wide variety of systems, including VM/CMS, VAX/VMS, Apollo, Sun and MacIntoshes. No knowledge of ORACLE is required to use the FATMEN system. ORACLE is not required on remote systems, although it, or SQL/DS may be used, if desired.

TMS - The CERN Tape Management System

The CERN Tape Management System was originally developed at the Rutherford Appleton Laboratories (RAL) in the UK. It is based upon a relational database system, currently IBM's SQL/DS. The TMS code runs in a dedicated service machine. All access to the TMS is via the SYSREQ communications package. Normally, this interface is hidden behind panels or the FATMEN system.

CSPACK - The Client Server package (CERN Program Library Q124)

CSPACK is a collection of programs and FORTRAN callable routines developed for client-server applications. It includes the SYSREQ package, used to access the Tape Management System and the XZ package, used for remote file access and transfer.

SYSREQ is a facility developed at RAL for generalised inter-system communications. It allows commands to be sent to, and replies received from, services running in dedicated service machines under the VM/CMS. All communication with the TMS is via SYSREQ, although this is transparent to the user. At CERN, a facility has been developed to permit remote users use the facilities of SYSREQ, by forwarding the messages and replies over TCP/IP. This system, known as TCPREQ, was developed by F. Hemmer.

VAXTAP - VAX Tape Utilities (CERN Program Library Z312)

The VAXTAP set of utilities was originally developed for the CERN VAX cluster VXCRNA. The utilities are available both with and without an interface to the CERN Tape Management System and may be used with and without FATMEN. FATMEN uses the VAXTAP tape staging system.

Chapter 2: The FATMEN model

The FATMEN system is based upon a layer model, comprising the following components.

- User Programs
- Data structure management packages
- Event directories
- Production database
- File catalogue
- Network file system
- Media management system
- Stage/Setup
- Mount
- Operator or Robot

The file catalogue is provided by the FATMEN package. Layers above the file catalogue are typically experiment specific, or at least vary from experiment to experiment. Thus, the CERN collaboration ALEPH and the DESY group H1 both use BOS as a data structure management package, whereas many other collaborations use ZEBRA. Similarly, the L3 collaboration has developed the DBL3 package as a production database, whereas OPAL uses a system named OPCAL.

The layers below the FATMEN file catalogue are typically system, or at least site dependant. For example, CERN uses different staging systems on the central IBM and Siemens systems, the central VAX cluster and the Cray. In addition, staging software also exists on the L3 Apollos and the SHIFT facility. All of these systems are interfaced to FATMEN in a manner that is completely transparent to the user.

The Media Management System, which at CERN is the Tape Management System (TMS) developed at the Rutherford Appleton Laboratories in the UK, may again be substituted by another package, or be missing altogether. User exits are provided for the case when FATMEN is installed at a site without a media management system, and an interface already exists to the Systems Center VMTAPE product.

Chapter 3: The FATMEN File Catalogue

The main point of entry into the FATMEN system is the FATMEN File Catalogue. This looks to the user much like a distributed Unix file system, where files are referenced by a so-called **generic-name**. Using this file catalogue, the user can issue normal file system commands, described later. In addition, file names may be added to the file catalogue and the data referenced by these names subsequently transparently accessed through the find command.

3.1 The Generic Name

The generic name is of the form

```
//catalogue/experiment/dir1/dir2/.../dirn/filename
```

where the slash character (/) is a directory delimiter, as for Unix file names, **catalogue** indicates in which catalogue the file resides **experiment** is the name of the experiment to which the file belongs The rest of the file name is free format, although its total length may not exceed 255 bytes, and each component may not exceed 20 characters. Typically, experiments will have conventions for file names. DELPHI, for example use the format (prefixed by `//CERN/DELPHI`)

```
 Origin/Stage/Inst/Select/Energy/Sample/FileseqVersion
```

e.g. for simulated raw data of $q\bar{q}$ events:

```
//CERN/DELPHI/SIMD/RAWD/CERN/QQ/E093.25/P01R000001/F0001V01
```

The **generic-name** differs from a conventional file name in that

1 A single generic name can point to a single file, a subset of a file or multiple files.

2 Many generic names may point to the same (set of) files.

3 The generic name is operating system independant.

The components of the generic name

The generic name is made up of several components, namely the *catalogue name*, the *experiment* name, the *path* name and the *file* name.

The catalogue name

The catalogue name is that component of the generic name which follows the initial double slash, up to the next slash character, e.g. **CERN** in //CERN/NA31/886/MIN8/KS01. This does **not** indicate where the data resides, nor the site at which the FATMEN software is running. The same catalogue name is used on all systems where FATMEN is used by the experiment NA31, in this example. Thus the *catalogue name* is the logical file system. Experiments based at DESY would use generic names starting //DESY, those based at SLAC would use //SLAC, on all systems where access to the catalogue or data was required.

The experiment name

The experiment name follows immediately after the catalogue name. There are no particular restrictions to this name. However, the catalogues for each experiment are kept in separate files.

The path name

The path name is the complete generic name up to the last slash, e.g. //CERN/NA31/886/MIN8 in the example above.

The file name

The file name is that part of the generic name which follows the last slash, e.g. KS01 in the example above.

3.2 The dataset name, or fileid

The dataset name is the file name as used by the host operating system. This is normally never specified except when entries are added to the catalogue.

3.3 The relationship between the generic name and datasets

As stated above, there can be multiple entries in the FATMEN catalogue for a given generic name. These entries correspond to different copies of a given dataset. In a typical case, the master copy may be on a cartridge at CERN, another copy at RAL, with a disk copy on the workstation cluster at the pit.

For a given generic name, only one entry can exist with the same host name and dataset name, if the dataset resides on disk or with the same volume serial number (VSN), visual identifier (VID) and file sequence number (FSEQ) for tape resident datasets.

3.4 The Command Line interface

The most simple interface to the FATMEN system is the command line interface, or shell. After activating this shell, the user can enter Unix-like commands, such as **ls, cd, pwd, mv, rm, cp** etc. In addition, there are commands to add new file names to the catalogue **(add/disk and add/tape)** and to access existing data **(find)** or create new data **(make)**. Using the **ls** command, users can display information about their files, such as date and time catalogued, owner, file format and user comment string.

3.5 The FIND and MAKE commands

The FIND and MAKE commands provide users with access to their data in a device, medium and location independent manner. The user simply specifies a generic name and logical unit on which they wish to read or write. The FATMEN system will then associate this logical name with the data, either directly, if the data are on disk, or via the tape staging system, if it is a tape dataset. In a future version, the data may reside on a remote disk or tape and a server process will provide record level access with on-the-fly data conversion.

3.6 The FORTRAN callable interface

The FORTRAN callable interface provides all of the functionality of the command line interface and more. As the user has direct access to all of the information in the file catalogue, subject to normal security restrictions, he is able to perform much more sophisticated operations on the various fields of the file catalogue. Basically, the command line interface provides access to information which is essential to safe retrieval of the data, whereas the FORTRAN callable interface allows the user to manipulate the data according to user-defined fields, to override values in the database and so on.

3.7 The Tape Management System

The FATMEN file catalogue retains minimal information about tapes upon which data reside. It assumes that full details are kept in a Tape Management System (TMS), with which the file catalogue is fully integrated. When a user wishes to access a dataset on a given tape or cartridge, the FATMEN system will inquire of the TMS whether the user has the required access permission, whether the media is in fact available, its location etc. Using the information returned, it will then build a valid tape STAGE request, specifying the device type required and other details such as label type (ASCII, EBCDIC, Unlabelled). Thus, users need not know that their data resides on a tape which is in an automatic loader, or indeed if it is on tape at all!

When a new dataset is created, the TMS may also be used to allocate a new tape or cartridge within a certain storage group as specified by the user. The TMS provides a great deal of flexibility in this area, thus an experimental group may divide up its tape allocation into separate pools for raw data, Monte Carlo data, DSTs and other data. The user may then request a tape from the appropriate storage pool, or provide a user routine to perform the allocation.

The CERN TMS resides on CERNVM. All access to the TMS is via the SYSREQ communications package. Remote access is provided via a TCP/IP server into CERNVM, called TCPREQ. The user may issue TMS commands without passing through the FATMEN system: these will typically be performed by the tape librarian or group tape managers. More details of the TMS are given in a separate document: The CERN Tape Management System (to be published).

Remote sites with their own TMS's are required to provide two FORTRAN interface routines to link their TMS to the FATMEN system. An interface already exists for VMTAPE and is selected with the PATCHY flag

```
+USE,VMTAPE.
```

The following restrictions apply if the FATMEN file catalogue is used without a TMS:

1 Default values are used for media types (e.g. 3480), generic device types (e.g. CT1), label types (e.g. SL) etc. These may be changed at any time by calling the routine FMEDIA.

2 For individual tape volumes, this information can be overridden by providing a user exit FMUTMS. This exit also permits the user to indicate that a volume is inaccessible (unavailable or permission denied).

3 To allocate volumes from named pools at run-time, a user-exit FMUALL must be provided.

3.8 Interacting with the FATMEN catalogue

The logical structure of the FATMEN user/server interaction is shown in figure 3.1 on page 10. The FATMEN catalogue is accessed in read-only mode, using FORTRAN direct-access or C I/O. The catalogue itself may be local or remote - this is completely transparent to the user. All that is required is that a symbol (VAX/VMS systems) or environmental variable (Unix systems) is defined, giving the location of the catalogue. In this way, remote catalogues may be accessed using `NFS` [10], `DFS` [11], in VAXcluster systems, using the `SHIFT` [12] remote file I/O software, or using other remote file access techniques, as they become available.

Updates to the catalogue are made by writing update files in a subdirectory of the directory containing the catalogue itself. These updates are processed by a dedicated server, which applies the updates to the catalogue and sends them to all other servers that are defined. In addition, the updates are saved as journal files in a special directory. The updates may be transmitted using a variety of network protocols. The default is via TCP/IP, but the exact method is chosen using a configuration file.

When building the servers, one may also activate code that causes the updates to be applied both to the normal catalogue which is retained in a ZEBRA RZ file, and also in a relational database, such as ORACLE. This second database is not normally seen by users (unless they explicitly invoke SQL commands) and is maintained principally for backup purposes.

3.9 Remote access to the FATMEN file catalogue

At CERN, a dedicated FATMEN server exists on an IBM RS6000 machine. The catalogues maintained on this system, which are automatically kept in synchronisation with those on the CERNVM system, may be accessed from remote nodes via NFS, or via the facilities provided in CSPACK.

Remote systems may retain local copies or extracts of the central catalogue. These are useful for systems with poor network connections to CERNVM (or the nearest production centre running FATMEN) or for systems which will mainly process 'closed-shop' data. In all cases, updates to the local catalogue are forwarded to CERNVM and all other production centres.

Remote FATMEN catalogues are currently in use on several IBM systems, such as LEPICS, IN2P3, SACLAY and UKACRL. They are also be installed on various VMS and Unix machines (including the Cray).

The FATMEN catalogue servers distribute updates according to entries in configuration files. Updates may be sent over Bitnet, DECnet or TCP/IP connections. More details concerning the distribution of updates are given in the installation section of this manual.

3.10 Remote access to data

Remote access to disk data is provided through standard facilities such as DECnet, NFS or DFS. It is also possible using the CSPACK routines and, in the SHIFT environment, via interfaces to the SHIFT software.

Remote access to tape data is available via the SHIFT software and, on the L3 Apollos, via an interface to the L3 stage command.

Figure 3.1: The logical structure of the FATMEN user/server interaction

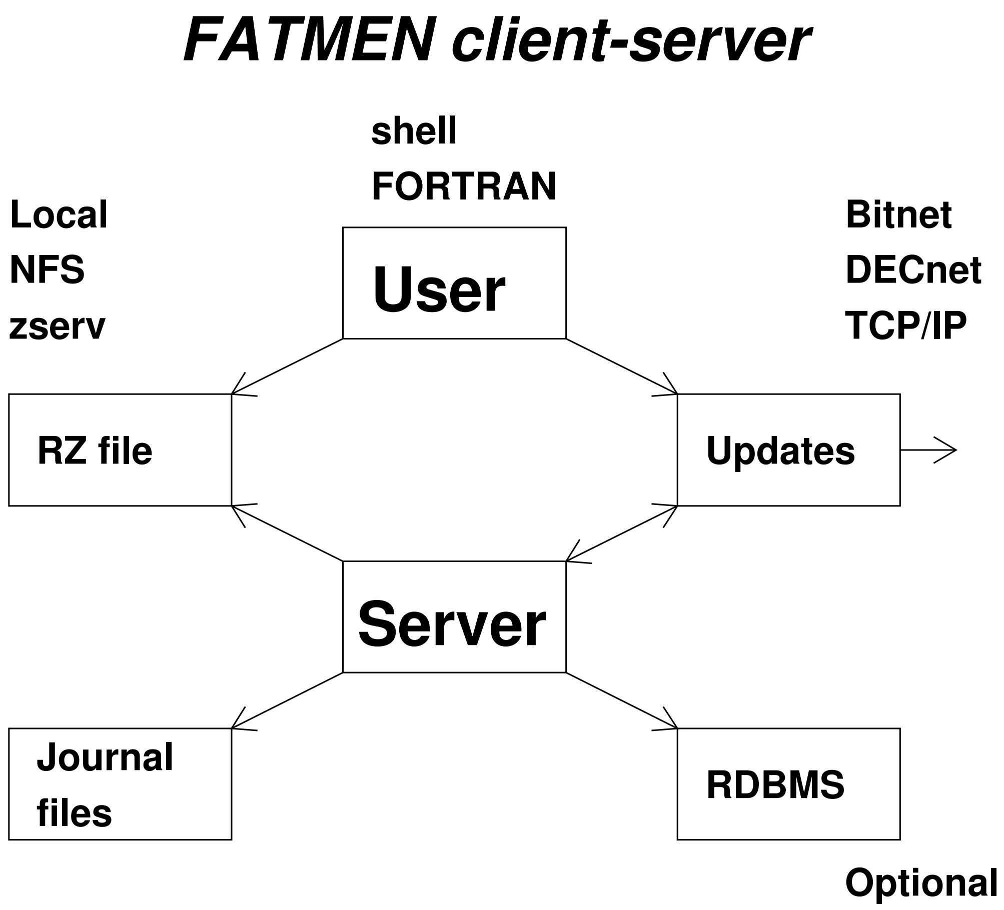


Figure 3.2: The FATMEN user/server interaction on VM/CMS systems

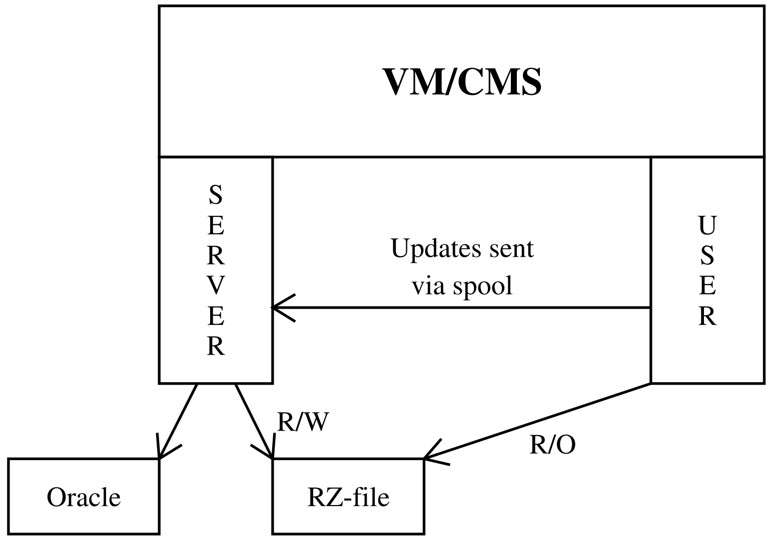
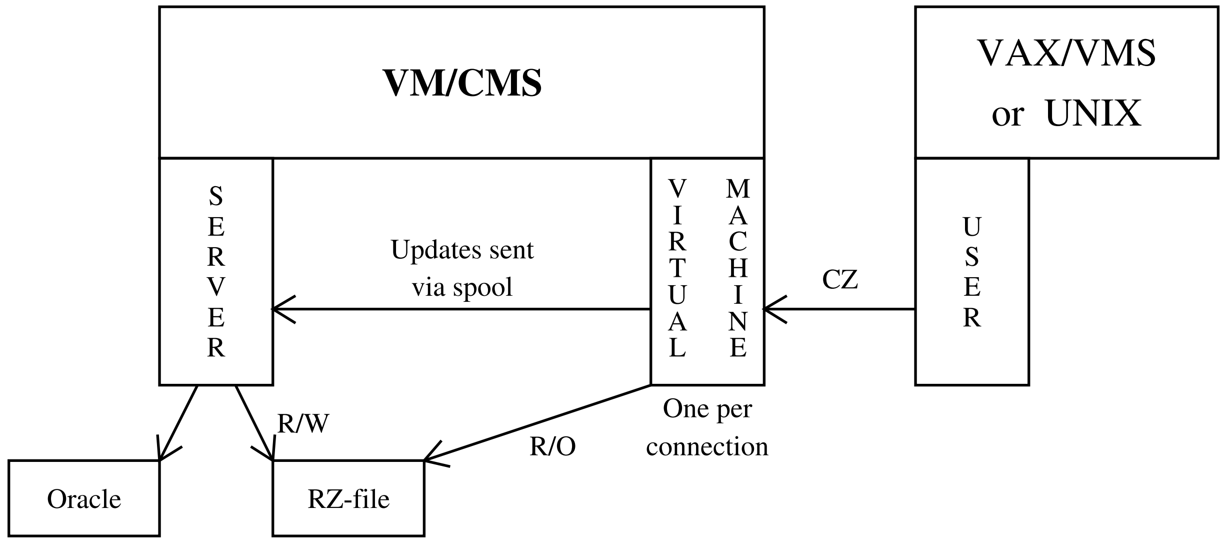


Figure 3.3: Interacting with the FATMEN database via the ZEBRA server



Chapter 4: File Catalogue Structure Overview

The FATMEN catalogue is maintained in a ZEBRA RZ (FORTRAN direct access) file. These files, one per experiment, are updated only by dedicated servers, again one per experiment. Updates are sent in FZ ASCII exchange format to these servers, which then read them in and apply them to the RZ database. Updates typically involve creation or deletion of files or directories.

4.1 The ORACLE database

In the design stage of FATMEN, it was felt essential that the contents of the RZ files be backed up in a relational database. Some hundred thousand or more entries were eventually expected in the file catalogue and it was decided to back this data up in a reliable database with journaling, rollback and other recovery mechanisms. Normally, the full features of a relational database are not required, although they may well be exploited by the database manager to perform auditing and repair operations on the database. In addition, complicated queries can be performed much more easily using ORACLE than with the RZ database. These might be of use to generate a special extract of the database, such as all datasets older than a certain date.

A third reason has since arisen for the use of a separate database and this is again that of data integrity. The code path used to maintain the relational database is completely separate from that used to update the RZ files, greatly reducing the chance of data loss due to software. **N.B. There is no requirement on any site to have ORACLE or any other relational database to run FATMEN. If selected, this code only ever runs in the server and is completely transparent to the user. The code involved will work with any relational database that provides a FORTRAN pre-compiler, including IBM's SQL/DS, DB2 etc.**

4.2 The ZEBRA RZ databases

The RZ structure, being based on the Unix file system with the addition of cycles or versions as in the VMS operating system, provides an excellent base for the FATMEN file catalogue. In an RZ database, a directory contains data identified by a number of keys. In the FATMEN file catalogue, these keys include the filename (i.e. that part of the generic name that follows the last slash, the pathname maps directly to the RZ pathname), the location code, medium type and copy level (original, copy, copy of a copy etc.). Further information, currently some 600 bytes per file, is stored in ZEBRA banks; one per file. Full details of the contents of the RZ and ORACLE databases are to be found in the users guide section of this manual. To summarize, they contain all information that is required to access transparently the data, plus ample space for user information, such as comments, bit encoded information, integer data etc. More information on ZEBRA and the RZ package are to be found in the ZEBRA users' guide.

4.3 Remote RZ databases

Remote RZ databases may be complete replicas of the central database or contain merely a subset of the information. The level of information in these remote copies is determined by an entry in a distribution list. When a server receives an update, it matches the pathname against a distribution list, and forwards the update to remote servers if the pathname matches. Thus, a server on LEPICS, the L3 IBM system in building 513 at CERN, may wish to receive information on all files in //CERN/L3, whereas a server on the ALEPH Wisconsin VAXcluster may only wish to receive information on files in //CERN/ALEPH/DST/WISC.

Up to 16 generic name patterns may be specified for each FATMEN server. Should the generic name of the update match any of these patterns, the update is forwarded to the remote server. The example below shows an entry for a remote FATMEN server at node UKACRL (RAL in the UK). All entries for simulated data (SIMD) will be sent to this node, plus all master and team DST information for real data.

Example of a NAMES file entry for the FATMEN servers

```
:nick FATSERVERS
              :list.fats1 fats2 fats3
:nick FATS1
:userid.FMDELPHI
:node.UKACRL
              :DIR1.//cern/delphi/simd
              :DIR2.//cern/delphi/alld/mdst
              :DIR3.//cern/delphi/alld/tdst
```

4.4 Reliability of the file catalogue

It is clear that the file catalogue must be extremely reliable, if it is to be used for access to thousands, perhaps even hundreds of thousands, of datasets. To this end, once an entry has been added to the file catalogue it is guaranteed. This is performed using various levels of journaling, described below.

1 Updates to the file catalogue consist of separate files, in FZ format. If these updates cannot be successfully processed, they are stored in a special directory (or mini-disk), for subsequent automatic retry.

2 Once updates have been successfully added to the local RZ database, they are then sent to all other systems as indicated in a distribution list. Again, automatic retry is provided.

3 Once an update has been successfully added to the local database and sent to all remote servers, it is moved to a journal area. These journal files are purged regularly, based upon the interval between regular disk backups.

4 On systems where the SQL interface is enabled, updates are automatically added to both RZ and SQL databases.

5 In addition, the RZ databases will normally be backed up on a regular basis. This, together with the local journal files, allows a remote site to reconstruct its RZ database.

4.5 Restrictions of the file catalogue

The `ZEBRA RZ` package supports a maximum of 65000 records per file. As `FATMEN` uses `RZ` to store the catalogue information, the same restriction applies to `FATMEN`.

The default record size of a `FATMEN` catalogue is 1024 words. If a larger record size is used, one may store more entries, although there is an overhead concerned with directory records, which always take an integral number of records.

For efficiency reasons, one should not store too many entries per directory. Experience suggests that a few hundred to one-two thousand entries is a suitable number.

Part II

FATMEN – Tutorial

Chapter 5: A tutorial introduction to FATMEN

5.1 What is FATMEN?

The FATMEN package is a distributed File and Tape management system. It provides transparent access to file catalogue information and to data with no source code changes required when moving from one host to another. This is independent of the location of the data or of the host operating system.

All data is referenced by a Unix-like name, the so-called generic-name. The FATMEN system is responsible for matching this name against a physical dataset.

5.2 How does FATMEN help?

FATMEN removes the necessity of having to know details such as which tape or disk a file is on. In addition, commands or subroutine calls to access data are identical on all supported systems (currently VM/CMS, MVS, VAX/VMS and Unix). Thus, one can take the same code from machine to machine and access data in a transparent manner, although the underlying mechanism or even the copy of the data may be completely different.

In addition, through the FATMEN catalogue, users may list characteristic of a data set, such as a comment field, user words, file format etc. which otherwise would have to be recorded in a log book or NEWS item.

5.3 Explanation of terms

When using the FATMEN system, all data is referenced by a name known as the **generic-name**. The generic name has the form

```
//catalogue/experiment/dir1/dir2/.../dirn/filename
```

where the slash character (/) is a directory delimiter, as for Unix file names, **catalogue** indicates in which catalogue the file resides **experiment** is the name of the experiment to which the file belongs The rest of the file name is free format, although its total length may not exceed 255 bytes and each component may not exceed 20 characters. Typically, experiments will have conventions for the generic-name, but the sort of information that you might want to include in the generic-name is

- Real data, simulated (possibly also technical run, cosmics etc.)
- Beam particle (e.g. pi+, pi- etc.)
- Beam energy
- Target (for fixed target experiments)
- Period and run
- Magnet setting, if relevant
- Number of the pass through the reconstruction chain
- Level, i.e. DST, RAWDATA, etc.

Examples of catalogue names are CERN, for all CERN experiments, DESY, for experiments based at DESY, and so on. Examples of experiment name are L3, H1, CDF.

Note that the same generic-name can be used for more than one file. In this case, the files are all assumed to contain the same data, but may well reside on different media or in different locations. The file format may differ, for example, one copy might be in Zebra FZ native format and another in exchange format. We will see later how different entries with the same generic-name may be selected or listed.

Associated with each generic-name are a catalogue entry and a key vector which contains important information that can be used to make a first pass selection. The catalogue entry is in fact a Zebra bank stored in a Zebra RZ file, and the keys are the normal RZ keys. However, for all practical purposes it is not necessary to know the structure of the catalogue in such detail, particularly when using the FATMEN shell (interactive interface) or the so-called 'novice' FORTRAN callable interface, which both hide Zebra completely.

The key vector contains the filename, i.e. the part of the generic-name following the last slash, and information on the mediatype on which the data resides, the location of the file and the so-called copy level. By using these keys, it is possible to make a first pass selection of a file, or to view only a subset of a catalogue in a very efficient manner. For example, when working at ones home laboratory say in the United States, one is probably not interested in looking at catalogue entries corresponding to files located several thousand kilometers away in CERN, still less in trying to access them over the network.

The meanings of the various keys are experiment defined, except for the media type, which is defined as follows:

```
1: disk
2: 3480 cartridge
3: 3420 tape
4: 8200 Exabyte cartridge
```

The location code

The location code is one piece of information available to FATMEN to select the best available source of data. The following convention is used by OPAL:

```
         0=Cern Vault     CERNVM VXCERN CRAY SHIFT etc
         1=Cern Vault
         2=Cern Vault
        11=VXOPON         OPAL Online Vax cluster
        12=Online         OPAL (apollo) online facilities
        21=VXOPOF         OPAL Offline cluster
        31=SHIFT          SHIFT disk and archive storage
     33101=Saclay         Active cartridges
     33901=Saclay         'obsolete' cartridges
     44501=UKACRL         Active cartridges
     44901=UKACRL         'obsolete' cartridges
```

Even if the location code is not set, FATMEN will still be able to find 'the best' copy of a file. However, it is much more efficient to restrict the search by specifying one or more location codes, as this results in less I/O to the FATMEN catalogue and, more importantly, less queries to the Tape Management System (TMS).

The copy level

Initially, the copy level was defined as follows:

```
     0       original
     1       copy of an original
     2       copy of a copy
     ...
```

In fact, this has been of limited use and it is now more commonly used to indicate the data representation. The following definitions are used by OPAL and correspond to those used by the FPACK package developed at DESY.

```
    1       IEEE floating point format (used on Unix workstations and
                                       for Zebra exchange data format)
    2       IBM  floating point format
    3       VAX  floating point format
    4       IEEE floating point format, but byte-swapped
    5       Cray floating point format
    4       IEEE floating point format, but byte-swapped
```

5.4 Using the FATMEN catalogue - a simple example

The FATMEN catalogue can be used as a general purpose catalogue, for example, to store information about the records, cassettes and CDs that someone owns.

To create a catalogue, we just type

```
mkfatnew
```

We then give the name `MUSIC` as the name of the FATMEN system, and `CLASSICAL` as the name of the experiment. This will create a file called `MUSIC.FATRZ` and all generic names will start `//MUSIC/CLASSICAL`.

We can then catalogue our collection by composer, making directories such as `BACH`, `BEETHOVEN`, `CHOPIN` etc.

```
fm
FM>mkdir BACH
FM>mkdir BEETHOVEN
FM>mkdir CHOPIN
```

We may well group the music under the type of work, such as `CONCERTOS`, `SONATOS`, `SYMPHONIES`. We then have very readable generic-names, e.g. `//MUSIC/CLASSICAL/BEETHOVEN/SYMPHONIES/NUMBER9`, could correspond to Beethoven's 9th Symphony. Note that, using the FATMEN shell, we would never need to type the `//MUSIC/CLASSICAL` as this would be our 'home' directory. We can even return to it by typing `cd $HOME`.

We can now type commands such as:

```
FM>ls beethoven/symphonies/(7:9) | List works 7-9 inclusive
FM>ls b*/c*/*                   | All works by composers
                                  with names beginning B
                                  and categories beginning C.
FM>ls */* -gc | Display the full name and comment field of
                every entry
```

We can, of course, use other fields of the catalogue, such as the media type. Here we will use 1 to mean CD, 2 for cassette and 3 for LP. Thus, by default, FATMEN will preferentially find a CD before a cassette before an LP. We could also define the location code, such as

```
1 : home
2 : car
3 : boat
4 : pad in St. Tropez
```

Then, we can limit our searching to a subset of the catalogue.

```
FM>set/location 3
FM>ls */* -gc | Now we just see what we have on our boat
```

If we get a new DAT player, we may want to add this, say as media type 4. We can then redefine the search order with:

```
FM>set/media 1,4,2,3 | Access DAT after CD but before everything else
```

There are also other fields that could be of interest, such as the performance date, for which we could use the creation date field, the performers, e.g. Berlin Philharmonic, and the conducter. We could then use the search command to list all entries performed by the London Sympony Orchestra (for example).

Thus, we can make selections according to

– The generic-name, e.g. BACH_*/*, to list all works by the BACH family.
– The FATMEN keys, such as the location-code, media-type or copy-level (which could be important for convential cassettes!)
– The catalogue entry itself, such as performance date, composer, record company etc.

We will see later how to use the same features to manage physics data, which is of course our main concern.

5.5 Starting to work with the FATMEN system

Before you can start to work with the FATMEN system, your group must have been entered into the FATMEN system as detailed in the installation guide section of this manual. This involves

– Creating a set of ORACLE or SQL/DS tables for your experiment **N.B.This is only necessary if you wish to have the FATMEN catalogued backed up into a relational database. This option is typically only used at CERN.**
– Creating a service machine to manage the FATMEN database
– Registering this machine with the FATMEN master service machine

To get this done, just send a mail to JAMIE@CERNVM.

5.6 Adding information to the file catalogue

Once the above steps have been taken, one can procede to catalogue data in the database. This can be done through either the FORTRAN callable interface or the FATMEN shell. We will demonstrate the use of both and show how the FATMEN system can then be used to access the data referenced by these catalogue entries.

Adding existing data to the FATMEN catalogue

The following examples show how existing data may be catalogued within FATMEN. The examples are taken from the Fermilab CDF experiment.

Information on existing data is kept in flat files, as shown below.

```
R015881E.TOP04   |CED984|FNALH   |MUA21|04-SEP-1989:06:53:19.60|GOOD|
R015882B.TOP04   |CED984|FNALH   |MUA21|04-SEP-1989:08:59:03.96|GOOD|
R015882C.TOP04   |CED984|FNALH   |MUA21|04-SEP-1989:10:25:58.47|GOOD|
R015882D.TOP04   |CED984|FNALH   |MUA21|04-SEP-1989:11:50:21.63|GOOD|
R015898A.TOP04   |CED984|FNALH   |MUA21|04-SEP-1989:13:26:08.62|GOOD|
R015910A.TOP04   |CED984|FNALH   |MUA21|04-SEP-1989:14:13:17.86|GOOD|
R015911A.TOP04   |CED984|FNALH   |MUA21|04-SEP-1989:15:43:56.84|GOOD|
```

where the fields are dataset name (filename), tape number (vsn), nodename, device name on which the tape was written, date and time and status.

The easiest way to add this information is to write a small command file that converts the file into a KUMAC file. This can be done as follows.

```
$       open/read in top04.tape
$       open/write out addfat.kumac
$loop:
$       read/end=eof in line
$       line=f$edit(line,"TRIM,COMPRESS,COLLAPSE")
$       dsn =f$element(0,"|",line)
$       vsn =f$element(1,"|",line)
$       node=f$element(2,"|",line)
$       comm=f$element(5,"|",line)
$       fname=f$element(0,".",dsn)
$       temp =dsn - fname - "."
$       gname="//fnal/cdf/dst/" + temp + "/" + fname
$       write out "ADD/TAPE ''vsn' ''vsn' 1 ''gname' ''dsn' _"
$       write out "YBB 0 ''node' ''comm' F 0 0 200 3"
$       goto loop
$eof:   close in
$       close out
```

This generates the following KUMAC file.

```
ADD/TAPE CED984 CED984 1 //fnal/cdf/dst/TOP04/R015880A R015880A.TOP04 _
UN 0 FNALH GOOD F 0 0 200 3
ADD/TAPE CEH060 CEH060 1 //fnal/cdf/dst/TOP04/R015881B R015881B.TOP04 _
UN 0 FNALH GOOD F 0 0 200 3
ADD/TAPE CED984 CED984 1 //fnal/cdf/dst/TOP04/R015881C R015881C.TOP04 _
UN 0 FNALH GOOD F 0 0 200 3
ADD/TAPE CED984 CED984 1 //fnal/cdf/dst/TOP04/R015881D R015881D.TOP04 _
UN 0 FNALH GOOD F 0 0 200 3
ADD/TAPE CED984 CED984 1 //fnal/cdf/dst/TOP04/R015881E R015881E.TOP04 _
UN 0 FNALH GOOD F 0 0 200 3
ADD/TAPE CED984 CED984 1 //fnal/cdf/dst/TOP04/R015882B R015882B.TOP04 _
UN 0 FNALH GOOD F 0 0 200 3
ADD/TAPE CED984 CED984 1 //fnal/cdf/dst/TOP04/R015882C R015882C.TOP04 _
UN 0 FNALH GOOD F 0 0 200 3
ADD/TAPE CED984 CED984 1 //fnal/cdf/dst/TOP04/R015882D R015882D.TOP04 _
UN 0 FNALH GOOD F 0 0 200 3
ADD/TAPE CED984 CED984 1 //fnal/cdf/dst/TOP04/R015898A R015898A.TOP04 _
UN 0 FNALH GOOD F 0 0 200 3
ADD/TAPE CED984 CED984 1 //fnal/cdf/dst/TOP04/R015910A R015910A.TOP04 _
UN 0 FNALH GOOD F 0 0 200 3
ADD/TAPE CED984 CED984 1 //fnal/cdf/dst/TOP04/R015911A R015911A.TOP04 _
UN 0 FNALH GOOD F 0 0 200 3
```

To add this to FATMEN we now type

```
EXEC ADDFAT
```

from within the FATMEN shell.

We could also do the same thing using the FORTRAN interface. This would have the advantage that we could also convert the date and time of creation from VAX format and add that to the catalogue, as is shown in the following FORTRAN program.

The ADDTEST fortran program

```
      PROGRAM ADDTEST
*
*     Add stuff to FATMEN catalogue, using CDF tape log files
*     A similar function can be performed by using ADDFAT.COM,
*     followed by ADDFAT.KUMAC in the FM shell.
*
      CHARACTER*256 GENAME,DSN,CHLINE
      CHARACTER*80  COMM
      CHARACTER*8   HOST
      CHARACTER*23  VAXDAT
      CHARACTER*6   VSN
      CHARACTER*4   FFORM,RECFM,CHOPT
      CHARACTER*3   MONTHS(12),CHMON
      DATA          MONTHS( 1)/'JAN'/,MONTHS( 2)/'FEB'/,
     +              MONTHS( 3)/'MAR'/,MONTHS( 4)/'APR'/,
     +              MONTHS( 5)/'MAY'/,MONTHS( 6)/'JUN'/,
     +              MONTHS( 7)/'JUL'/,MONTHS( 8)/'AUG'/,
     +              MONTHS( 9)/'SEP'/,MONTHS(10)/'OCT'/,
     +              MONTHS(11)/'NOV'/,MONTHS(12)/'DEC'/
*
* Start of FATMEN sequence FATPARA
*
** ***     Data set bank mnemonics
*
*
*          Keys
      PARAMETER ( MKSRFA= 1, MKFNFA= 2, MKCLFA=7, MKMTFA=8
     1           ,MKLCFA= 9, MKNBFA=10, NKDSFA=10 )
*
** ***     Bank offsets
*
      PARAMETER ( MFQNFA=  1, MHSNFA= 65, MCPLFA= 67, MMTPFA= 68
     1           ,MLOCFA= 69, MHSTFA= 70, MHOSFA= 74
     2           ,MVSNFA= 77, MVIDFA= 79, MVIPFA= 81, MDENFA= 82
     3           ,MVSQFA= 83, MFSQFA= 84, MSRDFA= 85, MERDFA= 86
     4           ,MSBLFA= 87, MEBLFA= 88, MRFMFA= 89, MRLNFA= 90
     5           ,MBLNFA= 91, MFLFFA= 92, MFUTFA= 93, MCRTFA= 94
     6           ,MCTTFA= 95, MLATFA= 96, MCURFA= 97, MCIDFA= 99
     7           ,MCNIFA=101, MCJIFA=103, MFPRFA=105, MSYWFA=106
     8           ,MUSWFA=116, MUCMFA=126, NWDSFA=145
     9           ,MFSZFA=MSYWFA,MUSCFA=MSYWFA+1)
*
* End of FATMEN sequence FATPARA
*
      COMMON /USRLNK/LUSRK1,LUSRBK,LADDBK,LUSRLS
```

```
      DIMENSION     NFAT(NWDSFA)

*
*     Initialise FATMEN & Zebra...
*
      CALL FMSTRT(1,2,'//FNAL/CDF',IRC)
      CALL FMLOGL(0)
*
*     Open the data file...
*
      OPEN(3,FORM='FORMATTED',STATUS='OLD')
*
*     Now process the data...
*
10    CONTINUE
      READ(3,'(A)',END=99) CHLINE
*     PRINT *,'Processing ',CHLINE(1:LENOCC(CHLINE))

*123456789_123456789_123456789_123456789_123456789_123456789_123456789_
*R015880A.TOP04   |CED984|FNALH   |MUA21|04-SEP-1989:01:31:48.11|GOOD

*
*     Convert date and time...
*
      VAXDAT = CHLINE(41:60)
      READ(VAXDAT,'(I2,1X,A3,1X,I4,1X,I2,1X,I2)')
     +     IDAY,CHMON,IYEA,IHOU,IMIN
      IMON   = ICNTH(CHMON,MONTHS,12)

      IYEA   = MOD(IYEA,1900)

      ID     = IYEA*10000 + IMON*100 + IDAY
      IT     = IHOU*100   + IMIN
*
*     and pack for insertion into FATMEN bank...
*
      CALL FMPKTM(ID,IT,IP,IRC)

      GENAME = '//FNAL/CDF/DST/'//CHLINE(10:14)//'/'//CHLINE(1:8)

      VSN    = CHLINE(19:24)

      DSN    = CHLINE(1:14)

      HOST   = CHLINE(26:31)

      IFILE  = 1

      CHOPT  = 'N'
*
*     Change later to YBB...
*
      FFORM  = 'UN'

      ICOPY  = 0
```

```
      RECFM  = 'V'

      COMM   = CHLINE(65:68) // ' ' // CHLINE(35:39)

      CALL FMADDT(GENAME,VSN,VSN,IFILE,
     +            DSN,FFORM,ICOPY,HOST,RECFM,0,0,0,0,COMM,
     +            IVECT,CHOPT,IRC)

*
*     Now get back the bank into a vector and modify the fields
*     that we could not via FMADDT
*
      CALL FMPEEK(GENAME,NFAT,' ',IRC)
*
*     Media type 3 = 3420
*
      NFAT(MMTPFA) = 3
*
*     these should be zero anyway...
*
      CALL VZERO(NFAT(MUSWFA),10)
*
*     Creation date...
*
      NFAT(MCRTFA) = IP

      CALL FMPOKE(GENAME,NFAT,'P',IRC)

      GOTO 10

99    CLOSE(3)
*
      CALL FMEND(IRC)
      PRINT *,'Return code ',IRC,' from FMEND'
*
      END
```

Adding and referencing data using the FATMEN shell

Let us assume that the tapes have volume serial numbers (VSN - the magnetically recorded label) JS1001 to JS1010 inclusive. The visual identifiers (VID) are different - CIN136 to CIN145 inclusive. As this type of VID implies, these are actually IBM 3480 cartridges, rather than the older open reel tape. The file identifier of all the datasets is the same - RAWDATA. In order to catalogue these files we must first establish a table linking each file or tape with a generic-name. Let us assume that the generic-name starts with `//CERN/OPAL/LEPD/RAWD/P51989`. This implies that the data are real LEP data (as opposed to simulated data), in raw data format, i.e. no filtering or reconstruction. The data were recorded in period 5 in 1989.

The FATMEN shell does not allow all fields of the file catalogue to be entered. We are able to specify the following fields:

1 Volume sequence number (VSN)
2 Visual identifier (VID)

3 File sequence number (FSEQ)
4 File identifier or dataset name (DSN)
5 File format (FX, FZ etc.)
6 Copy level (is this a copy or an original?)
7 Host name
8 Comment

Suppose we now enter this information into a disk file, each line representing a different tape. The resultant file might look like the following:

```
JS1001 CIN136 1 RAWDATA FX 0 VXOPON 'Early physics run - some dectectors out'
JS1002 CIN137 1 RAWDATA FX 0 VXOPON 'Early physics run - some dectectors out'
JS1003 CIN138 1 RAWDATA FX 0 VXOPON 'Early physics run - some dectectors out'
JS1004 CIN139 1 RAWDATA FX 0 VXOPON 'Early physics run - some dectectors out'
JS1005 CIN140 1 RAWDATA FX 0 VXOPON 'Early physics run - some dectectors out'
JS1006 CIN141 1 RAWDATA FX 0 VXOPON 'Early physics run - some dectectors out'
JS1007 CIN142 1 RAWDATA FX 0 VXOPON 'Early physics run - some dectectors out'
JS1008 CIN143 1 RAWDATA FX 0 VXOPON 'Early physics run - some dectectors out'
JS1009 CIN144 1 RAWDATA FX 0 VXOPON 'Early physics run - some dectectors out'
JS1010 CIN145 1 RAWDATA FX 0 VXOPON 'Early physics run - some dectectors out'
```

As we first wish to add this data using the FATMEN shell, we must edit the file to include the generic-name of each dataset. To do this, we will create a file with filetype KUMAC, so that we can then execute it directly from within the FATMEN shell. First, we include a command to change the current directory to `//CERN/OPAL/LEPD/RAWD/P51989`. This is done using the command `cd`. Note that there is no need to type the `//CERN/OPAL` as we enter at this level by default. Then, between the file sequence number and dataset name, we insert the generic-name. For simplity, we will refer to these files as FILE1 to FILE10. We also add the command `ADD/TAPE` at the beginning of each line to instruct the FATMEN shell to add this information to the database.

Resultant file `TESTFAT KUMAC` **(or** `testfat.kumac`**)**

```
cd lepd/rawd/p51989

ADD/TAPE JS1001 CIN136 1 TAPE1 RAWDATA FX 0 VXOPON 'Early physics run - some detectors out'
ADD/TAPE JS1002 CIN137 1 TAPE2 RAWDATA FX 0 VXOPON 'Early physics run - some detectors out'
ADD/TAPE JS1003 CIN138 1 TAPE3 RAWDATA FX 0 VXOPON 'Early physics run - some detectors out'
ADD/TAPE JS1004 CIN139 1 TAPE4 RAWDATA FX 0 VXOPON 'Early physics run - some detectors out'
ADD/TAPE JS1005 CIN140 1 TAPE5 RAWDATA FX 0 VXOPON 'Early physics run - some detectors out'
ADD/TAPE JS1006 CIN141 1 TAPE6 RAWDATA FX 0 VXOPON 'Early physics run - some detectors out'
ADD/TAPE JS1007 CIN142 1 TAPE7 RAWDATA FX 0 VXOPON 'Early physics run - some detectors out'
ADD/TAPE JS1008 CIN143 1 TAPE8 RAWDATA FX 0 VXOPON 'Early physics run - some detectors out'
ADD/TAPE JS1009 CIN144 1 TAPE9 RAWDATA FX 0 VXOPON 'Early physics run - some detectors out'
ADD/TAPE JS1010 CIN145 1 TAPE10 RAWDATA FX 0 VXOPON 'Early physics run -some dectectors out'
```

We can now procede to run this macro, by typing the following commands:

```
FM
EXEC TESTFAT
END
```

Now that these files have been catalogued, we can look at the catalogue information using the `ls` command. This command allows us to list the various files and there attributes. For example, the command `ls tape% -a` would list all details (option a) for the files tape1-tape9. Tape10 would not be listed as the % character only matches against a single character. We must use the syntax `ls tape* -a` to see also this file.

Of perhaps more interest is the ability to be able to access the data itself by using the generic name. This is performed by using the `find` command. Thus, we could type the commands

```
fm
cd lepd/rawd/p51989
find tape7 iofile13
end
```

This would initiate an input tape stage of the volume matching the generic name

```
//CERN/OPAL/LEPD/RAWD/P51989/TAPE7
```

Once the stage has completed, a FORTRAN program can read the data on IOFILE13.

Adding data using the FORTRAN callable interface

The following example shows how data may be added to the catalogue using the FORTRAN interface. The `FMADDD` and `FMADDT` routines provide the same functionality as the shell ADD/DISK and ADD/TAPE commands.

Adding information to the catalogue

```
      PROGRAM ADDTEST
      CHARACTER*256 GENAME,DSN
      CHARACTER*80  COMM
      CHARACTER*8   HOST,CHUSER
      CHARACTER*4   FFORM,RECFM,CHOPT
*
*     Sequence FATPARA from PATCH FATCDES on FATMEN pam
*
+CDE,FATPARA.

      CALL FMSTRT(1,2,'//CERN/CNDIV',IRC)
      CALL FMLOGL(3)

      GENAME = '//CERN/CNDIV/JAMIE/ULF'
      DSN    = '<JAMIE.192>BOX.SET'
      FFORM  = 'AS'
      HOST   = 'CERNVM'
      RECFM  = 'V'
      COMM   = 'ADDED VIA NEW ADDTEST FORTRAN'
      CHOPT  = 'N'
*
*     CHOPT = N : do not add this entry to the catalogue
*     this allows us to modify other fields before sending
*     the update to the server.
*
```

```
      CALL FMADDD(GENAME,DSN,FFORM,0,HOST,RECFM,80,11,483,2,COMM,
     +            IVECT,CHOPT,IRC)
*
*     Update user file format - this field is not accessible
*     directly via FMADDD
*
*     Two ways of doing this are shown.
*
+SELF,IF=PEEK.
*
*     Get contents of bank into vector IVECT
*
      CALL FMPEEK(GENAME,IVECT,' ',IRC)
*
*     Update user file format
*
      CALL UCTOH('NFF',IVECT(MFUTFA),4,3)
*
*     Copy vector back and update catalogue
*
      CALL FMPOKE(GENAME,IVECT,'P',IRC)
+SELF,IF=-PEEK.
      CALL FMPUTC(-1,'NFF',MFUTFA,3,IRC)
*
*     Now update the catalogue
*         Options: N - ignore IVECT
*                  P - add entry via FMPUT
*
      CALL FMPOKE(GENAME,IVECT,'NP',IRC)
+SELF.
      CALL FMEND(IRC)
      PRINT *,'Return code ',IRC,' from FMEND'
*
      END
```

5.7 Access to data using the FORTRAN callable interface

The following example shows how a dataset may be accessed through the FORTRAN interface.

Access to data

```
      CHARACTER*80 GENAME
      COMMON /QUEST/IQUEST(100)
*
*     FATMEN keys. These will be returned by the call to FMOPEN
*
      PARAMETER (LKEYFA=10)
      DIMENSION KEY(LKEYFA)
*
*     Units for reading the FATMEN catalogue and for
*     sending updates to the server
*
      LUNRZ = 1
      LUNFZ = 2
*
```

```
*     Initialise FATMEN novice interface
*
      CALL FMSTRT(LUNRZ,LUNFZ,'//CERN/OPAL',IRC)
*
      GENAME = '//CERN/OPAL/DDST/PASS3/FYZ1/P18R1929/C01'
      LG = LENOCC(GENAME)
*
*     Access and open the file. R indicates Read mode
*     and F instructs FMFILE to issue the appropriate
*     call to FZFILE for this dataset.
*
      LBANK = 0
      CALL FMFILE(11,GENAME(1:LG),'RF',IRC)
      IF(IRC.NE.0) THEN
         PRINT *,'Return code ',IRC,' from FMFILE'
         GOTO 10
      ELSE
*
*     Now process the file, reading just the FZ headers
*
         CALL READRZ(11)
      ENDIF
*
*     Now close this file, issuing FZENDI (option E),
*     drop staging disk (option D)
*
      CALL FMFEND(11,GENAME(1:LG),'ED',IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMFEND'
1     CONTINUE
10    CONTINUE
*
      END

      SUBROUTINE READFZ(LUN)
      COMMON/QUEST/IQUEST(100)
      CHARACTER*8  DELTIM
      DIMENSION    IUHEAD(400)
      DIMENSION    IOCR(100)
      PARAMETER (JBIAS=2)

      NREC = 0
      CALL FMRTIM(DELTIM)
      CALL TIMED(T)
   1  CONTINUE
      NHEAD = 400
      IXDIV = 0
      CALL FZIN(LUN,IXDIV,LSUP,JBIAS,'S',NHEAD,IUHEAD)
      IF(IQUEST(1).LT.4) THEN
         NREC = NREC + 1
         GOTO 1
      ENDIF

      PRINT *,'READFZ. end after ',NREC,' records, IQUEST(1) = ',
     +        IQUEST(1)
      CALL FMRTIM(DELTIM)
      CALL TIMED(T)
      PRINT *,'FMFZLP. Elapsed time = ',DELTIM,
```

```
     +         ' CP time = ',T,' sec.'
      END
```

5.8 Access to data step by step

The following examples go through some of the various options available when accessing a dataset. The simplest example as already been given, and is access to a dataset using the FATMEN shell.

We just type

```
FM> find generic-name unit
```

If unit is numeric, it will be interpreted as a FORTRAN unit number. Thus, after typing

```
find //cern/cndiv/jamie/test 11
```

we would have the following.

- On VM systems, a FILEDEF on unit FT11F001 pointing to the dataset referenced by //cern/cndiv/jamie. Any mini-disk links and accesses or stage operations would have been performed automatically, so that a subsequent FORTRAN program could just open unit 11 and read.
- On VMS systems, a logical name FOR011. Once again, our FORTRAN program can just open unit 11 and read.
- On Unix systems, a soft link fort.11.

One could also specify a unit such as VM11F001, if the file were to be read by VMIO, or any character string that would be used as a logical name or symbolic link, e.g.

```
find //cern/cndiv/jamie/test mydat
```

Our FORTRAN program could then open the named file MYDAT. (On VM or MVS systems, this corresponds to the DDNAME, rather than the file name.)

Normally, we will wish to issue the FIND command from FORTRAN, as this is more powerful. Although a routine `FMFIND` exists, we will describe the `FMOPEN` routine, which includes all of the functionality of `FMFIND` and more.

Our call to `FMOPEN` looks like

```
      CALL FMOPEN('//CERN/CNDIV/JAMIE/TEST',CHLUN,LBANK,CHOPT,IRC)
```

Here, the generic name and unit are specified exactly as in the shell, and are both of type character. This permits the use of units such as VM11F001, MYDAT and so on. Again, if we call `FMOPEN` with

```
      CALL FMOPEN('//CERN/CNDIV/JAMIE/TEST','11',LBANK,CHOPT,IRC)
```

the correct FORTRAN name on the system in question would be used. The exception to this is when the generic name in question points to a file which should not be processed by FORTRAN. In this case, FMOPEN will automatically perform the correct operation. For example, EPIO files on the IBM should be read with IOPACK, hence FMOPEN will build a DDNAME of IOFILE11. (This was also true for Zebra exchange format files on the IBM prior to version 3.67, when FORTRAN I/O became the default).

Unlike the shell, FMOPEN will not only perform operations such as staging the file as required, but will also issue the correct OPEN. This may be overridden by the character option parameter CHOPT.

FMOPEN options

In the case of files to be processed by the Zebra FZ or RZ package, we can ask FMOPEN to perform the FZFILE or RZFILE call. For example,

```
CALL FMOPEN('//CERN/CNDIV/JAMIE/TEST','11',LBANK,'F',IRC)
```

where F indicates that a call to FZFILE is to be issued. FATMEN obtains the correct parameters for the call to FZFILE from the catalogue.

Another interesting option is the S option. This will instruct FATMEN to update the catalogue with the file size. This is useful for tape files, as future accesses will request a staging disk of the correct size, and hence use the system more efficiently.

We can also request that an automatic duplicate of the file is made into the robot at CERN (SMCF), via the D option. This option requires a pool of tapes gg_FAT1, e.g. XX_FAT1, in the TMS. Naturally, if a robot copy already exists a new one will not be made.

Access to tape data

For a dataset on a tape volume to be accessible, a device capable of reading or writing the volume must exist on the local node, or on a server node, in the case of remote access to data or VAXcluster systems.

Suppose we wished to read the dataset with generic name //CERN/CNDIV/JAMIE/TEST. A copy of this dataset might exist on tape volume 123456. If this volume required a device of tape CT1, the FATMEN software will attempt to determine if such a device exists on the local node. Where remote tape access is available, it will check if such a device exists on one of the server nodes. This is done as follows.

VM/CMS systems running HEPVM software

A entry must exist in the file SETUP NAMES for the device CT1 (in this example). **(Not yet implemented)**

VAX/VMS systems running VAXTAP software

A logical name in the system table SETUP_CT1S (in this example) must exist for locally attached tapes. Served tapes must be defined in the file SETUP_EXE:TPSERV.CONF

Cray Unicos systems

An entry for the corresponding device must exist in the file /etc/shift.conf.

Unix systems running the SHIFT tape software

An entry for the corresponding device must exist in the file /etc/shift.conf or in the file /etc/TPCONFIG.

Access to remote data

The above examples will work on both local and remote data, without any change. Note that a call to FMOPEN on an Apollo in Helsinki will not result in a cartridge being mounted in the robot at CERN. Remote access to tape data must be explicitly enabled. It is currently enabled in the SHIFT facility, where the data is staged via the CRAY, and on the L3 Apollos, where the data is staged through LEPICS.

Access to remote disk data requires explicit selection of a copy of a dataset. This may be done as shown below.

Example of using the FMSELK routine

```
*     Argument declarations
      PARAMETER (LKEYFA=10)
      PARAMETER (MAXKEY=999)
      DIMENSION INKEYS(LKEYFA),OUKEYS(LKEYFA,MAXKEY)
*     The following statements will select all datasets
*     with copy level (MKCLFA) of 1 (i.e. a copy of an original file),
*     media type of 1 (i.e. disk) and location code of 1 (i.e. CERN)
      INKEYS(MKCLFA) = 1
      INKEYS(MKMTFA) = 1
      INKEYS(MKLCFA) = 1
      CALL FMSELK('//CERN/CNDIV/JAMIE/TEST',INKEYS,OUKEYS,NFOUND,MAXKEY,IRC)
      IF(NFOUND.GT.0) THEN
*     Just take the first one found which matches
         CALL FMGETK('//CERN/CNDIV/JAMIE/TEST',LBANK,OUKEYS(1,1),IRC)
*     Now pass the bank to FMOPEN.
         CALL FMOPEN('//CERN/CNDIV/JAMIE/TEST','11',LBANK,'F',IRC)
      ENDIF
```

In the current version this dataset would only be accessible if mounted via NFS.

5.9 The relationship between generic names, keys vectors and Zebra banks

As explained above, there can be multiple copies of a file with the same generic name. Typically, these copies will reside on different media, in different locations or have different data representations.

An entry exists in the FATMEN catalogue for each copy of a file. This entry consists of a Zebra bank and an associated vector known as the KEYS vector. One can use the keys vector, explicitly or implicitly, to select a particular copy of a file. More details on how this is done are given below.

Many of the FATMEN callable routines have the generic name, Zebra bank address and keys vector as arguments. If the bank address is zero, the FATMEN software retrieves the bank corresponding to the specified generic name from the FATMEN catalogue. In the case of multiple entries for the same generic name, the entry which is returned is determined by the rules described below. The exception to this rule is the routine FMGETK, which returns the entry corresponding to the key vector specified.

The FATMEN selection rules

By default, the FATMEN selection is as follows:

1 The catalogue is scanned for entries matching a given generic name. No check is made on location code, or copy level, but the different media types are processed in numerical order 1 - 4. (Media types 1 - 4 are disk, 3480 cartridge, 3420 tape and Exabyte 8200 cassette respectively).

2 If a entry is found for a given media type that is accessible, that entry is taken and the search stops.

3 For disk files, the host name in the catalogue must match the current host name for the file to be deemed accessible.

4 In the case of VAXclusters, the node name in the catalogue may be the VAXcluster alias or the name of any member of the cluster.

5 In the case of tape files, an entry on a robotically mounted volume is taken in preference over one on a manually mounted volume.

6 If the interface to the Tape Management System (TMS) is enabled, the volume must exist and be in an active vault.

Note that for systems such as Apollo, SHIFT etc. the node name that the FATMEN software uses can be set using an environmental variable. Thus on the various SHIFT nodes at CERN (shift1, shd01 etc.) the node name is set to SHIFT.

One can set ranges of valid location codes and copy levels using the routines `FMSETL` and `FMSETC` respectively. (The corresponding shell commands are `SET/LOCATION` and `SET/COPYLEVEL`). If ranges for either of these keys are set, then only entries with keys that match will be considered for selection. Note that setting a range of location codes can result in significantly faster selection time in the case of multiple entries, particurly if multiple TMS queries are avoided.

For example, if one makes 30 copies of every DST tape for export to outside laboratories, one can avoid up to 29 TMS queries by using different location codes at these sites.

In addition to ranges of location codes and copy levels, one may also set ranges of media types with `FMSETM` or `SET/MEDIATYPE`. Here, the order is also significant, thus `SET/MEDIATYPE 2,4,1` will look first on 3480 cartridge, then Exabyte 8200 cassette and finally disk.

In some cases, a more powerful selection technique is needed. For example, one may want to set the search order to

1 Local disk, native data format

2 Robotically mounted tape, native data format

3 Local disk, exchange data format

4 Robotically mounted tape, exchange data format

to avoid the overhead of conversion between data representation types. This can be achieved using the FORTRAN routine `FMSETK`. This routine is described further in the user guide section of this manual.

5.10 Using FATMEN to make copies of datasets

FATMEN can make copies of a dataset with automatic update of the catalogue. This is available both through the shell and the FORTRAN interface. The shell version provides an interface to the file transfer routines of CSPACK (the same as those used by ZFTP) and hence permits the network transfer of files. The files are transferred from disk to disk. If the local file resides on tape, it is first staged to disk. For reasons of program size this facility is not yet enabled on VM or MVS machines, but just for VMS and Unix systems. An interface to remote tapes is not yet provided.

Both the shell COPY command and the FORTRAN routine `FMCOPY` permit conversion of data representation and record format during copy. That is, one may copy an input Zebra FZ binary exchange file to an output Zebra FZ native file, or an input file with VBS format to an output file with record format U.

The following example shows the use of FMCOPY. This example was written for DELPHI, but similar programs are in use by CPLEAR, L3 and OPAL.

Example of copy data with FMCOPY

```
**********************************************************************
* PROGRAM DELRCOPY                                                   *
* ================                                                   *
* Make copies of all files corresponding to generic names on unit 10 *
* into the robot (SMCF) allocating volumes from pool XX_RAWD         *
**********************************************************************
      PARAMETER (LURCOR=100000)
      COMMON/CRZT/IXSTOR,IXDIV,IFENCE(2),LEV,LEVIN,BLVECT(LURCOR)
      DIMENSION    LQ(999),IQ(999),Q(999)
      EQUIVALENCE (IQ(1),Q(1),LQ(9)),(LQ(1),LEV)
      COMMON /USRLNK/LUSRK1,LUSRBK,LUSRLS
      COMMON /QUEST/IQUEST(100)
*
* Start of FATMEN sequence FATPARA
*
** ***     Data set bank mnemonics
*
*          Keys
      PARAMETER ( MKSRFA= 1, MKFNFA= 2, MKCLFA=7, MKMTFA=8
     1           ,MKLCFA= 9, MKNBFA=10, NKDSFA=10 )
*
** ***     Bank offsets
*
      PARAMETER ( MFQNFA=  1, MHSNFA= 65, MCPLFA= 67, MMTPFA= 68
     1           ,MLOCFA= 69, MHSTFA= 70, MHOSFA= 74
     2           ,MVSNFA= 77, MVIDFA= 79, MVIPFA= 81, MDENFA= 82
     3           ,MVSQFA= 83, MFSQFA= 84, MSRDFA= 85, MERDFA= 86
     4           ,MSBLFA= 87, MEBLFA= 88, MRFMFA= 89, MRLNFA= 90
     5           ,MBLNFA= 91, MFLFFA= 92, MFUTFA= 93, MCRTFA= 94
     6           ,MCTTFA= 95, MLATFA= 96, MCURFA= 97, MCIDFA= 99
     7           ,MCNIFA=101, MCJIFA=103, MFPRFA=105, MSYWFA=106
     8           ,MUSWFA=116, MUCMFA=126, NWDSFA=145
     9           ,MFSZFA=MSYWFA,MUSCFA=MSYWFA+1)

* End of FATMEN sequence FATPARA
      CHARACTER*6  DENS
      CHARACTER*8  LIB
      CHARACTER*4  LABTYP
      CHARACTER*1  MNTTYP
      CHARACTER*8  MODEL
      CHARACTER*7  ROBMAN(2)
      DATA         ROBMAN(1)/'-Robot '/,ROBMAN(2)/'-Manual'/
      PARAMETER (LKEYFA=10)
      PARAMETER (MAXFIL=1000)
      DIMENSION KEYS(LKEYFA,MAXFIL)
      DIMENSION JSORT(MAXFIL)
      DIMENSION KEYSIN(LKEYFA),KEYSOU(LKEYFA,MAXFIL)
      CHARACTER*255 FILES(MAXFIL)
      CHARACTER     THRONG*8, DSN*8, VSN*6, VID*6
      CHARACTER*80  TOPDIR,GENAME
      CHARACTER*26  CHOPT
*
```

```
*     Initialise ZEBRA
*
      CALL MZEBRA(-3)
      CALL MZSTOR(IXSTOR,'/CRZT/','Q',IFENCE,LEV,BLVECT(1),BLVECT(1),
     +            BLVECT(5000),BLVECT(LURCOR))
      CALL MZLOGL(IXSTOR,-3)
*
* *** Define user division and link area like:
*
      CALL MZDIV  (IXSTOR, IXDIV, 'USERS', 50000, LURCOR, 'L')
      CALL MZLINK (IXSTOR, '/USRLNK/', LUSRK1, LUSRLS, LUSRK1)
*
*     Units for FATMEN RZ/FZ files
*
      LUNRZ = 1
      LUNFZ = 2
*
*     Initialise FATMEN
*
      CALL FMINIT(IXSTOR,LUNRZ,LUNFZ,'//CERN/DELPHI',IRC)
      CALL FMLOGL(1)
      IDEBFA = 2
      NPROC  = 0

10    CONTINUE
      CALL TIMEL(T)
      IF(T.LT.100) THEN
         PRINT *,'Stopping due to time limit'
         GOTO 99
      ENDIF
      READ(10,9001,END=99) GENAME
9001  FORMAT(A80)
*     GENAME = FILES(JSORT(I))
      LGN = LENOCC(GENAME)
      PRINT *,'Processing ',GENAME(1:LGN)
      IROBOT = 0
*
*     First, check that a robot copy does not already exist
*
      CALL VBLANK(KEYSIN(2),5)
      LFN = INDEXB(GENAME(1:LGN),'/') + 1
*
*     Don't compare copy level or location code
*
      KEYSIN(MKCLFA) = -1
      KEYSIN(MKLCFA) = -1
*
*     Restrict search to 3480s
*
      KEYSIN(MKMTFA) = 2
      CALL FMSELK(GENAME(1:LGN),KEYSIN,KEYSOU,NMATCH,MAXFIL,IRC)
      IF(IRC.NE.0) THEN
         PRINT *,'Return code ',IRC,' from FMSELK'
         PRINT *,'Skipping ',GENAME(1:LGN)
         GOTO 10
      ENDIF
      IF(IDEBFA.GE.2)
```

```
     +PRINT *,'Found ',nmatch,' matches for media type 2'
      DO 30 J=1,NMATCH
      CALL FMQVOL(GENAME(1:LGN),LBANKR,KEYSOU(1,J),
     +            LIB,MODEL,DENS,MNTTYP,LABTYP,IC)

      IF(INDEX(LIB,'*Unknown') .NE.0) THEN
         IF(IDEBFA.GE.0) PRINT *,'Cannot determine mount type'
      ENDIF

      IF(MNTTYP.EQ.'R') THEN
         IF(IDEBFA.GE.0) PRINT *,'Robot copy already exists'
         IROBOT = 1
      ENDIF

30    CONTINUE
      IF(IROBOT.EQ.0) THEN
         PRINT *,'Robot copy does not exist for ',GENAME(1:LGN)
*
*     Now make a robot copy
*     First, allocate a new tape
*
         CALL FMALLO('3480','38K',' ','SMCF_1','XX_RAWD',
     +               LBANKR,' ',VSN,VID,IRC)
         IF(IRC.NE.0) THEN
            PRINT *,'Cannot allocate new tape - STOP!'
            GOTO 99
         ENDIF
*
*     Display the bank
*
         CALL FMSHOW(GENAME(1:LGN),LBANKR,KEYSOU(1,J),'A',IRC)
         LZERO = 0
         CALL VZERO(KEYSIN,10)
*
*     and make the copy
*
         CALL FMLOGL(3)
         CALL FMCOPY(GENAME(1:LGN),LZERO,KEYSIN,
     +               GENAME(1:LGN),LBANKR,KEYSOU(1,J),' ',IRC)
         IF(IRC.NE.0) THEN
            PRINT *,'Error from FMCOPY - STOP!'
            GOTO 99
         ENDIF
         CALL MZDROP(IXSTOR,LBANKR,' ')
         CALL MZDROP(IXSTOR,LZERO,' ')
         NPROC = NPROC + 1
      ENDIF
      GOTO 10
99    CONTINUE
      PRINT *,'Made ',NPROC,' copies, ',T,' seconds left'
*
*     Terminate cleanly
*
      CALL FMEND(IRC)
      END
```

5.11 Using the TMS tag information

When FATMEN is installed with the TMS option, access to the TMS tag information is possible. The TMS keeps two tags for each volume. As FATMEN already contains a fair amount of user information in the catalogue, it is not recommended that these tags be used to store further such information. Firstly, the TMS tags are only available for data on tape and secondly, access to the TMS tags is slower and less efficient than to information in the FATMEN catalogue. It can be useful to set the text tag to the generic name. If, for example, the generic name

```
//cern/l3/prod/data/sdsutt/cc00ftgu
```

points to the volume XP3088 one could set the text tag on this volume to be

```
//cern/l3/prod/data/sdsutt/cc00ftgu
```

This means that if we find the tape XP3088 somewhere and want to know what is on it, we can query the TMS and have a pointer to the entry in the FATMEN catalogue. Alternatively, one could use the command

```
SEARCH */* VID=XP3088
```

The TMS text tag can be set to the generic name with the shell command

```
tag //cern/l3/prod/data/sdsutt/cc00ftgu -s
```

5.12 Using tape pools within the TMS

The following is the recommended method of using tape pools within the TMS.

- When a new tape is required, it is first allocated out of the specified pool using the subroutine `FMALLO`.
- The job that allocates this volume should then write the data and, if successful, move the volume to another named pool, optionally write-locking it.
- If the job fails, it should return the volume to the pool from which it came.
- When the data is no longer required, the entry can be deleted from the catalogue with the tape volume automatically write-enabled and freed using the shell rm command.

Thus, a FORTRAN program might look like the following.

Example of using TMS tape pools

```
*
*     Allocate a new 3480 from the pool XX_FREE
*
      CALL FMALLO('3480','38K',' ','SMCF_1','XX_FREE',
     +LBANK,' ',VSN,VID,IRC)
*
*     Now write the data ...
*
...
...
*
```

```
*     OK - write-lock the tape and move to XX_DSTS
*     set TMS tag to generic name
*
10    CONTINUE
      CALL FMPOOL(GENAM,LBANK,KEYS,'XX_DSTS','LS',IRC)
      RETURN
*
*     Error - free tape
*     delete TMS tag
*
20    CONTINUE
      CALL FMPOOL(GENAM,LBANK,KEYS,'XX_FREE','D',IRC)
      RETURN
      END
```

5.13 Using a Zebra link area to protect the addresses of FATMEN banks

The addresses of FATMEN banks obtained from the FATMEN RZ file, e.g. using the routines FMGET, FMGETK are stored in a link area by the FATMEN software. However, only the most recent bank address is saved - these routines will DROP any previous bank upon entry. Should you wish to retain the addresses of multiple banks, the following method may be used.

Example of using a link area

```
      COMMON /USRLNK/LUSRK1,LBANKS,LUSRLS
      DIMENSION LBANKS(NBANKS)
      CHARACTER*255 GENAM(NBANKS)
      PARAMETER (LKEYFA=10)
      DIMENSION KEYS(LKEYFA)
+CDE,FATBANK.
*
      CALL MZLINK (IXSTOR, '/USRLNK/', LUSRK1, LUSRLS, LUSRK1)
*
      DO 10 I=1,NBANKS
      CALL FMGET(GENAM(I)(1:LENOCC(GENAM(I))),LBANK,KEYS,IRC)
      CALL ZSHUNT(IXSTOR,LBANK,LBANKS(I),1,0)
*
*     FMGET will DROP any bank at LTDSFA - including the one
*     we just got!
*
      LTDSFA = 0

10    CONTINUE
```

5.14 Using the routine FMALLO to allocate a tape

The following example is taken from the CERN specific routine FMSMCF. This routine is called from FMOPEN if the option D is specified. It performs the following functions:

1 Checks to see if a copy of the specified dataset already exists in the robot (SMCF)

2 If not, a tape is allocated from the special pool gg_FAT1, e.g. WS_FAT1.

3 The FMCOPY routine is then invoked to copy the data and update the catalogue.

As soon as the catalogue has been updated, subsequent accesses will by default take the copy in the robot.

Using FMALLO to allocate a tape

```
      SUBROUTINE FMSMCF(GENAME,LBANK,IRC)
*
*     Routine to make a copy of the dataset STAGEd in into the robot
*     using FMCOPY option 'S' (STAGE CHANGE)
*
+CDE,FATBANK.
+CDE,FATPARA.
+CDE,TMSDEF.
      CHARACTER*(*) GENAME
      CHARACTER*6   VSN,VID,CHACC
      PARAMETER (LKEYFA=10)
      DIMENSION KEYS(LKEYFA),KEYSR(LKEYFA)
      PARAMETER       (MAXKEY=1000)
      DIMENSION KEYSOU(LKEYFA,MAXKEY),KEYSIN(LKEYFA)
      INTEGER   FMACNT
*
      LGN = LENOCC(GENAME)
*
*     Save old bank address
*
      LOLDFA = LBANK
      LTDSFA = 0
*
*     First, check that a robot copy does not already exist
*
      CALL UCOPY(KEYS,KEYSIN,10)
*
*     Don't compare copy level or location code
*
      KEYSIN(MKCLFA) = -1
      KEYSIN(MKLCFA) = -1
      CALL FMSELK(GENAME(1:LGN),KEYSIN,KEYSOU,NMATCH,MAXKEY,IRC)
      IF(IDEBFA.GE.2)
     +PRINT *,'FMSMCF. found ',nmatch,' matches for media type 2'
      DO 10 I=1,NMATCH
      CALL FMGETK(GENAME(1:LGN),LBANKR,KEYSOU(1,I),IRC)
      CALL UHTOC(IQ(L+KOFUFA+MVIDFA),4,VID,6)
      LVID = LENOCC(VID)
      CALL FMQTMS(VID(1:LVID),LIB,MODEL,DENS,MNTTYP,LABTYP,IC)
      IF(MNTTYP.EQ.'R') THEN
         IF(IDEBFA.GE.0) PRINT *,'FMSMCF. robot copy already exists'
         RETURN
      ENDIF
10    CONTINUE
*
*     Lift new bank for the robot copy
*
      CALL FMLIFT(GENAME(1:LGN),KEYSR,'DISK',' ',IRC)
      CALL FMLINK(GENAME(1:LGN),LBANKR,' ',IRC)
*
*     Blindly copy old bank into new...
*
      CALL UCOPY(IQ(LBANK+KOFUFA+MFQNFA),IQ(LBANKR+KOFUFA+MFQNFA),
     +           NWDSFA)
*
*     and the keys...
```

```
*
      CALL UCOPY(KEYS,KEYSR,10)
*
*     Set last access date, date of cataloging and use count
*
      CALL DATIME(IDATE,ITIME)
      CALL FMPKTM(IDATE,ITIME,IPACK,IRC)
      IQ(LBANKR+KOFUFA+MCTTFA) = IPACK
      IQ(LBANKR+KOFUFA+MLATFA) = IPACK
      IQ(LBANKR+KOFUFA+MUSCFA) = 1
*
*     Now, allocate new tape
*
      IC = FMACNT(CHACC)
      CALL FMALLO('3480','38K',' ','SMCF_1',CHACC(5:6)//'_FAT1',
     +LBANKR,' ',VSN,VID,IRC)
      IF(IRC.NE.0) THEN
         IF(IDEBFA.GE.0) PRINT *,'FMSMCF. Cannot allocate robot tape'
         RETURN
         ELSE
         IF(IDEBFA.GE.0) PRINT *,'FMSMCF. allocated ',VSN,' ',VID,
     +                            ' (VSN/VID)'
      ENDIF
*
*     Do the copy
*
      CALL FMCOPY(GENAME,LOLDFA,KEYS,GENAME,LBANKR,KEYSR,'S',IRC)
      IF(IRC.NE.0) PRINT *,'FMSMCF. return code ',IRC,' from FMCOPY'
*
*     Restore bank address
*
      LBANK = LOLDFA
      END
```

5.15 Processing multiple entries

One may frequently wish to perform the same operation on multiple datasets, or on multiple catalogue entries. For example, we may wish to reset the user words for all entries corresponding to Monte Carlo data. This can be done in a simple way by

- Using the FMLFIL routine to generate a list of entries.
- Loop over all entries found
- Read each entry from the catalogue
- Replace the user vector
- Update the catalogue

The following example shows how this can be done using the novice interface.

Modifying the user words

```
      PARAMETER     (MAXFIL=100)
      PARAMETER     (LKEYFA=10)
      CHARACTER*255 CHFILES(MAXFIL),GENAME
      DIMENSION     KEYS(LKEYFA,MAXFIL)
*
** ***     Data set bank mnemonics
*
*          Keys
      PARAMETER ( MKSRFA= 1, MKFNFA= 2, MKCLFA=7, MKMTFA=8
     1           ,MKLCFA= 9, MKNBFA=10, NKDSFA=10 )
*
** ***     Bank offsets
*
      PARAMETER ( MFQNFA=  1, MHSNFA= 65, MCPLFA= 67, MMTPFA= 68
     1           ,MLOCFA= 69, MHSTFA= 70, MHOSFA= 74
     2           ,MVSNFA= 77, MVIDFA= 79, MVIPFA= 81, MDENFA= 82
     3           ,MVSQFA= 83, MFSQFA= 84, MSRDFA= 85, MERDFA= 86
     4           ,MSBLFA= 87, MEBLFA= 88, MRFMFA= 89, MRLNFA= 90
     5           ,MBLNFA= 91, MFLFFA= 92, MFUTFA= 93, MCRTFA= 94
     6           ,MCTTFA= 95, MLATFA= 96, MCURFA= 97, MCIDFA= 99
     7           ,MCNIFA=101, MCJIFA=103, MFPRFA=105, MSYWFA=106
     8           ,MUSWFA=116, MUCMFA=126, NWDSFA=145
     9           ,MFSZFA=MSYWFA,MUSCFA=MSYWFA+1)

      PARAMETER (LURCOR=200000)
      COMMON/FAT/IXSTOR,IXDIV,IFENCE(2),LEV,LEVIN,BLVECT(LURCOR)
      DIMENSION    LQ(999),IQ(999),Q(999)
      EQUIVALENCE (IQ(1),Q(1),LQ(9)),(LQ(1),LEV)
*
      COMMON /QUEST/IQUEST(100)
      DIMENSION     IVECT(10)
*
*     Initialise FATMEN and Zebra
*
      LUNRZ = 1
      LUNFZ = 2
      CALL FMSTRT(LUNRZ,LUNFZ,'//CERN/OPAL',IRC)

      GENAME = '//CERN/OPAL/SIMD/DDST/PASS3/*/*'
      LG = LENOCC(GENAME)
*
*     Find all entries that match
*
      ICONT = 0
      IFLAG = 1
10    CONTINUE
      CALL FMLFIL(GENAME(1:LG),CHFILES,KEYS,NFOUND,MAXFIL,ICONT,IRC)
*
      DO 20 J=1,NFOUND
      LF = LENOCC(CHFILES(J))
      LBANK = 0
      PRINT *,'Processing ',CHFILES(J)(1:LF)
*
*     Read entry from catalogue
*
      CALL FMGETK(CHFILES(J)(1:LF),LBANK,KEYS(1,J),IRC)
```

```
*
*     here we could add checks on the bank contents
*
*     Store vector IVECT at offset MUSWFA, length 10
*
      CALL FMPUTV(LBANK,IVECT,MUSWFA,10,IRC)
*
*     and write back to the catalogue
*
      CALL FMMOD(CHFILES(J)(1:LF),LBANK,IFLAG,IRC)
*
*     drop bank
*
      CALL MZDROP(IXSTOR,LBANK,' ')
20    CONTINUE

      IF(ICONT.NE.0) GOTO 10

      CALL FMEND(IRC)
      END
```

The following example finds all generic-names that match a pattern containing wild cards, and then deletes all entries corresponding to tapes that are mounted robotically. The tape volumes are write-enabled and moved to a TMS pool so that they can be allocated for future use.

Example of processing multiple entries in FORTRAN

```
      PARAMETER (LURCOR=200000)
      COMMON/CRZT/IXSTOR,IXDIV,IFENCE(2),LEV,LEVIN,BLVECT(LURCOR)
      DIMENSION    LQ(999),IQ(999),Q(999)
      EQUIVALENCE (IQ(1),Q(1),LQ(9)),(LQ(1),LEV)
*
      COMMON /USRLNK/LUSRK1,LUSRBK,LUSRLS
*
      COMMON /QUEST/IQUEST(100)
*
* Start of FATMEN sequence FATPARA
*
** ***     Data set bank mnemonics
*
*          Keys
      PARAMETER ( MKSRFA= 1, MKFNFA= 2, MKCLFA=7, MKMTFA=8
     1           ,MKLCFA= 9, MKNBFA=10, NKDSFA=10 )
*
** ***     Bank offsets
*
      PARAMETER ( MFQNFA=  1, MHSNFA= 65, MCPLFA= 67, MMTPFA= 68
     1           ,MLOCFA= 69, MHSTFA= 70, MHOSFA= 74
     2           ,MVSNFA= 77, MVIDFA= 79, MVIPFA= 81, MDENFA= 82
     3           ,MVSQFA= 83, MFSQFA= 84, MSRDFA= 85, MERDFA= 86
     4           ,MSBLFA= 87, MEBLFA= 88, MRFMFA= 89, MRLNFA= 90
     5           ,MBLNFA= 91, MFLFFA= 92, MFUTFA= 93, MCRTFA= 94
     6           ,MCTTFA= 95, MLATFA= 96, MCURFA= 97, MCIDFA= 99
     7           ,MCNIFA=101, MCJIFA=103, MFPRFA=105, MSYWFA=106
     8           ,MUSWFA=116, MUCMFA=126, NWDSFA=145
```

```
     9          ,MFSZFA=MSYWFA,MUSCFA=MSYWFA+1)

* End of FATMEN sequence FATPARA
      CHARACTER*6  DENS
      CHARACTER*8  LIB
      CHARACTER*4  LABTYP
      CHARACTER*1  MNTTYP
      CHARACTER*8  MODEL
      CHARACTER*7  ROBMAN(2)
      DATA         ROBMAN(1)/'-Robot '/,ROBMAN(2)/'-Manual'/
      PARAMETER (LKEYFA=10)
      PARAMETER (MAXFIL=3000)
      DIMENSION KEYS(LKEYFA,MAXFIL)
      CHARACTER*255 FILES(MAXFIL)
      CHARACTER*8   THRONG
      CHARACTER*255 TOPDIR
      CHARACTER*26  CHOPT
      CHARACTER*8   DSN
*
*
*     Initialise ZEBRA
*
      CALL MZEBRA(-3)
      CALL MZSTOR(IXSTOR,'/CRZT/','Q',IFENCE,LEV,BLVECT(1),BLVECT(1),
     +            BLVECT(5000),BLVECT(LURCOR))
      CALL MZLOGL(IXSTOR,-3)

*
* *** Define user division and link area like:
*
      CALL MZDIV  (IXSTOR, IXDIV, 'USERS', 50000, LURCOR, 'L')
      CALL MZLINK (IXSTOR, '/USRLNK/', LUSRK1, LUSRLS, LUSRK1)
*
*     Units for FATMEN RZ/FZ files
*
      LUNRZ = 1
      LUNFZ = 2
*
*     Initialise FATMEN
*
      CALL FMINIT(IXSTOR,LUNRZ,LUNFZ,'//CERN/delphi',IRC)
      CALL FMLOGL(1)
*
*     Get list of file names
*
      JCONT = 0
1     CONTINUE
      CALL FMLFIL('//CERN/DELPHI/P01_*/RAWD/NONE/Y90V00/E*/L*/*',
     +FILES,KEYS,NFOUND,MAXFIL,JCONT,IRC)
*
*     IRC = -1 indicates that there are more files found than
*     fit in FILES(MAXFIL). Calling FMLFIL again with JCONT^=0 will
*     return the next MAXFIL matches
*
      IF(IRC.EQ.-1) THEN
        JCONT = 1
      ELSE
```

```
        JCONT = 0
      ENDIF

      PRINT *,NFOUND,' files found'

      DO 10 I=1,NFOUND
      LENF = LENOCC(FILES(I))
      PRINT *,'Processing ',FILES(I)(1:LENF)
      LBANK = 0
      CALL FMQMED(FILES(I)(1:LENF),LBANK,KEYS(1,I),IMEDIA,IROBOT,IRC)
      IF(IROBOT.NE.1) GOTO 10
*
*     Display media information and the full generic-name for this entry
*
      CALL FMSHOW(FILES(I)(1:LENF),LBANK,KEYS(1,I),'MG',IRC)
*
*     Unlock (write-enable) corresponding tape volume
*
      CALL FMULOK(FILES(I)(1:LENF),LBANK,KEYS(1,I),' ',IRC)
      IF(IRC.NE.0) THEN
         PRINT *,'Return code ',IRC,' from FMULOK for ',
     +   FILES(I)(1:LENF)
         GOTO 10
      ENDIF
*
*     Move to pool XX_DSTS.
*
      CALL FMPOOL(FILES(I)(1:LENF),LBANK,KEYS(1,I),
     +            'XX_RAWD',' ',IRC)
      IF(IRC.NE.0) THEN
         PRINT *,'Return code ',IRC,' from FMPOOL for ',
     +   FILES(I)(1:LENF)
         GOTO 10
      ENDIF
*
*     and remove the entry from the FATMEN catalogue
*
      CALL FMRM(FILES(I)(1:LENF),LBANK,KEYS(1,I),IRC)
      IF(IRC.NE.0) THEN
         PRINT *,'Return code ',IRC,' from FMRM for ',
     +   FILES(I)(1:LENF)
         GOTO 10
      ENDIF
10    CONTINUE
*
*     any more files?
*
      IF(JCONT.NE.0) GOTO 1
*
*     Terminate cleanly
*
      CALL FMEND(IRC)

      END
```

5.16 Deleting multiple files using the FATMEN shell

Neither the FORTRAN routine FMRM nor the shell commmand rm accept wild-cards in the generic-name. This is deliberate and protects against mis-use such as

```
rm *
```

or worse

```
rm */*
```

which would remove all files in all directories, subject to the normal protection rules, and generate a lot of work for the servers.

Multiple files can be deleted as shown in the previous FORTRAN example, or via a 2-stage operation in the shell. If, for example, one wished to delete all files belonging to JAMIE, one could type

Using the SEARCH command to prepare a macro

```
FM>SEARCH */* USER=JAMIE -D OUTPUT=DELETE.KUMAC
FM>EXEC DELETE
```

To approximate to the previous FORTRAN example, one could use

Using the SEARCH command to delete robot tapes

```
FM>SEARCH //CERN/DELPHI/P01_*/RAWD/NONE/Y90V00/E*/L*/* VID=I* -D _
   OUTPUT=DELROB.KUMAC
FM>EDIT DELROB | Now add -FU to each line
FM>EXEC DELROB
```

Note that the removal of entries from the catalogue should always be done with extreme care.

5.17 Access to TMS tag information

The following program shows how the TMS tags associated with a tape volume may be accessed. As usual in the FATMEN system, all access is based on the generic-name.

Access to TMS tag information from FORTRAN

```
      PARAMETER (LURCOR=200000)
      COMMON/CRZT/IXSTOR,IXDIV,IFENCE(2),LEV,LEVIN,BLVECT(LURCOR)
      DIMENSION    LQ(999),IQ(999),Q(999)
      EQUIVALENCE (IQ(1),Q(1),LQ(9)),(LQ(1),LEV)
*
      COMMON /USRLNK/LUSRK1,LUSRBK,LUSRLS
*
      COMMON /QUEST/IQUEST(100)
*
* Start of FATMEN sequence FATPARA
*
** ***     Data set bank mnemonics
*
```

```
*          Keys
      PARAMETER ( MKSRFA= 1, MKFNFA= 2, MKCLFA=7, MKMTFA=8
     1           ,MKLCFA= 9, MKNBFA=10, NKDSFA=10 )
*
** ***     Bank offsets
*
      PARAMETER ( MFQNFA=  1, MHSNFA= 65, MCPLFA= 67, MMTPFA= 68
     1           ,MLOCFA= 69, MHSTFA= 70, MHOSFA= 74
     2           ,MVSNFA= 77, MVIDFA= 79, MVIPFA= 81, MDENFA= 82
     3           ,MVSQFA= 83, MFSQFA= 84, MSRDFA= 85, MERDFA= 86
     4           ,MSBLFA= 87, MEBLFA= 88, MRFMFA= 89, MRLNFA= 90
     5           ,MBLNFA= 91, MFLFFA= 92, MFUTFA= 93, MCRTFA= 94
     6           ,MCTTFA= 95, MLATFA= 96, MCURFA= 97, MCIDFA= 99
     7           ,MCNIFA=101, MCJIFA=103, MFPRFA=105, MSYWFA=106
     8           ,MUSWFA=116, MUCMFA=126, NWDSFA=145
     9           ,MFSZFA=MSYWFA,MUSCFA=MSYWFA+1)

* End of FATMEN sequence FATPARA
      PARAMETER (LKEYFA=10)
      DIMENSION KEYS(LKEYFA)
      CHARACTER*80  GENAM
      CHARACTER*255 CHTAGS
*
*
*     Initialise ZEBRA
*
      CALL MZEBRA(-3)
      CALL MZSTOR(IXSTOR,'/CRZT/','Q',IFENCE,LEV,BLVECT(1),BLVECT(1),
     +            BLVECT(5000),BLVECT(LURCOR))
      CALL MZLOGL(IXSTOR,-3)

*
* *** Define user division and link area like:
*
      CALL MZDIV  (IXSTOR, IXDIV, 'USERS', 50000, LURCOR, 'L')
      CALL MZLINK (IXSTOR, '/USRLNK/', LUSRK1, LUSRLS, LUSRK1)
*
*     Units for FATMEN RZ/FZ files
*
      LUNRZ = 1
      LUNFZ = 2
*
*     Initialise FATMEN
*
      CALL FMINIT(IXSTOR,LUNRZ,LUNFZ,'//CERN/CNDIV',IRC)
      CALL FMLOGL(0)
*
*     Get list of file names
*
      GENAM = '//CERN/CNDIV/JAMIE/OUT'
      LG    = LENOCC(GENAM)
      CHTAGS = 'Archive tape for FATMEN source'
*
*     Set tag - default is Text tag
*
      CALL FMTAGS(GENAM(1:LG),LBANK,KEYS,CHTAGS,'S',IRC)
*
```

```
*     Read back and print tag
*
      CALL FMTAGS(GENAM(1:LG),LBANK,KEYS,CHTAGS,'G',IRC)
      PRINT *,IRC,CHTAGS(1:LENOCC(CHTAGS))
*
*     Terminate cleanly
*
      CALL FMEND(IRC)

      END
```

5.18 Run time tailoring of the FATMEN system

Certain features of the FATMEN system may be reconfigured at run-time, both via the FORTRAN callable interface or interactive shell. A description of what can be tailored is given below.

Host name and account fields

The FATMEN software attempts to determine various information, such as the current host name and account, by calling system routines. In some cases, it may be desirable to override the values returned. This can be done by setting variables (environmental variables in Unix, global symbols in VAX/VMS).

For example, the SHIFT facility at CERN is composed of a number of systems (shift1, shd01 etc.). By default, a file created on a given system would not be accessible on another as the node name check would fail. To override this, the variable `FMHOST` is set, as below.

```
setenv FMHOST shift
```

The account field can be set in a similar way, e.g.

```
FMACNT:==JDSCT
```

for a VAX/VMS system.

Media attributes

If FATMEN has been installed using the TMS flag, media attributes are obtained from the TMS (Tape Management System). (A preliminary interface to the VMTAPE package also exists and is selected via the flag VMTAPE).

On other systems, the defaults can be overridden globally, or on a per volume basis. Ideally, the media attributes for a given site should be automatically selected via installation flags. However, the `FMEDIA` callable routine and MEDIA shell command permit the default values to be overridden at any time.

For example, to change the generic device name of media type 2, which defaults at CERN to CT1, the following command could be used.

```
MEDIA 2 3480 TA90
```

Tailoring the FATMEN selection

By default, all entries in the FATMEN catalogue are visible and may be examined using the shell command ls. One may limit the range of entries that can be seen by defining a list of location codes, copy levels and media types. These lists also affect those files that may be accessed, and the selection procedure itself.

For example, an experiment which computes at several laboratories will probably have copies of the tapes containing the DST information at each site. The shell command

```
set/location 1-3
```

would prevent any catalogue entries with a location code outside the range 1-3 from being visible. This is particularly useful for large collaborations that typically have many copies of each dataset.

It is also important to set the correct range for efficiency in data access. In the case where an experiment has multiple copies of a file on different tapes, each at a different laboratory, FATMEN will normally issue a TMS query for each volume to see if it is accessible. Setting the location code correctly reduces the number of TMS queries and hence improves data access time.

In the case of the location code or copy level, only those entries with a value in the defined range will be visible or accessible. In the case of the medium type, the order of the values is also important. By default, FATMEN first looks for a disk file, then a copy on a 3480 cartridge and so on. The shell command

```
set/media 5,1,2
```

would cause FATMEN to first look for a copy on an Exabyte 8500 cartridge, then disk and finally a 3480 cassette, assuming the default medium attributes.

5.19 Plotting information from the FATMEN catalogue

The following program shows how certain information, such as the file size, number of days since last access, etc. can be histogrammed using HBOOK and saved in a file for further processing with PAW.

Before running, the variable THRONG should be set to the FATMEN group that is to be processed, as shown below.

Setting the `THRONG` variable on various systems

```
Unix systems:        THRONG=OPAL;export THRONG  (Bourne and Korn shells)
-------------        setenv THRONG OPAL         (C shell)

VMS systems:         throng==opal
------------

VM/CMS systems:      SETENV THRONG OPAL
---------------
```

Plotting FATMEN information with HBOOK

```
      PARAMETER (LURCOR=200000)
      COMMON/CRZT/IXSTOR,IXDIV,IFENCE(2),LEV,LEVIN,BLVECT(LURCOR)
      DIMENSION    LQ(999),IQ(999),Q(999)
      EQUIVALENCE (IQ(1),Q(1),LQ(9)),(LQ(1),LEV)
*
      COMMON /USRLNK/LUSRK1,LUSRBK,LUSRLS
*
      COMMON /QUEST/IQUEST(100)
      CHARACTER*8   THRONG
*
* Start of FATMEN sequence FATPARA
*
** ***     Data set bank mnemonics
*
*          Keys
      PARAMETER ( MKSRFA= 1, MKFNFA= 2, MKCLFA=7, MKMTFA=8
     1           ,MKLCFA= 9, MKNBFA=10, NKDSFA=10 )
*
** ***     Bank offsets
*
      PARAMETER ( MFQNFA=  1, MHSNFA= 65, MCPLFA= 67, MMTPFA= 68
     1           ,MLOCFA= 69, MHSTFA= 70, MHOSFA= 74
     2           ,MVSNFA= 77, MVIDFA= 79, MVIPFA= 81, MDENFA= 82
     3           ,MVSQFA= 83, MFSQFA= 84, MSRDFA= 85, MERDFA= 86
     4           ,MSBLFA= 87, MEBLFA= 88, MRFMFA= 89, MRLNFA= 90
     5           ,MBLNFA= 91, MFLFFA= 92, MFUTFA= 93, MCRTFA= 94
     6           ,MCTTFA= 95, MLATFA= 96, MCURFA= 97, MCIDFA= 99
     7           ,MCNIFA=101, MCJIFA=103, MFPRFA=105, MSYWFA=106
     8           ,MUSWFA=116, MUCMFA=126, NWDSFA=145
     9           ,MFSZFA=MSYWFA,MUSCFA=MSYWFA+1)

* End of FATMEN sequence FATPARA
*KEEP,FATBUG.
      COMMON /FATUSE/ IDEBFA, IDIVFA, IKDRFA, KOFSFA, KOFUFA, LBFXFA
     +              , LSAVFA, LTOPFA, LBBKFA, LBGNFA, LTDSFA, LBDSFA
```

```
     +              , LPRTFA, NTOPFA, LUFZFA, IOUPFA, IOBKFA, IODSFA
     +              , LLNLFA, LLNHFA
*KEND.
      CHARACTER*8   DSN
      EXTERNAL      UROUT
*
*
*     Initialise ZEBRA
*
      CALL MZEBRA(-3)
      CALL MZSTOR(IXSTOR,'/CRZT/','Q',IFENCE,LEV,BLVECT(1),BLVECT(1),
     +            BLVECT(5000),BLVECT(LURCOR))
      CALL MZLOGL(IXSTOR,-3)

*
* *** Define user division and link area like:
*
      CALL MZDIV  (IXSTOR, IXDIV, 'USERS', 50000, LURCOR, 'L')
      CALL MZLINK (IXSTOR, '/USRLNK/', LUSRK1, LUSRLS, LUSRK1)
*
*     Units for FATMEN RZ/FZ files
*
      LUNRZ = 1
      LUNFZ = 2
      CALL GETENVF('THRONG',THRONG)
      LTH = LENOCC(THRONG)
*
*     Initialise FATMEN
*
      CALL FMINIT(IXSTOR,LUNRZ,LUNFZ,'//CERN/'//THRONG(1:LTH),IRC)
      CALL FMLOGL(0)
*
*     Initialise HBOOK
*
      CALL HLIMIT(-20000)
*
*     Book histograms
*
      CALL HBOOK1(1,'File Size (MB)',50,0.,200.,0.)
      CALL HBOOK1(2,'Number of accesses',50,0.,50.,0.)
      CALL HBOOK1(3,'Number days since last access',50,0.,300.,0.)
      CALL HBOOK1(4,'Number days since catalogued',50,0.,300.,0.)
      CALL HBOOK1(5,'Number days since created',50,0.,300.,0.)
      CALL HBOOK1(6,'Medium',5,0.,5.,0.)
      CALL HIDOPT(0,'BLAC')
*
*     Loop over all files
*
      CALL FMLOOP('//CERN/*/*',-1,UROUT,IRC)
*
*     Print and store the histograms
*
      CALL HPRINT(0)
      CALL HRPUT(0,'FATTUPLE.'//THRONG(1:LTH),'N')
*
*     Terminate cleanly
*
```

```
      CALL FMEND(IRC)
      END

      SUBROUTINE UROUT(PATH,KEYS,IRC)
*
* Start of FATMEN sequence FATPARA
*
** ***     Data set bank mnemonics
*
*          Keys
      PARAMETER ( MKSRFA= 1, MKFNFA= 2, MKCLFA=7, MKMTFA=8
     1           ,MKLCFA= 9, MKNBFA=10, NKDSFA=10 )
*
** ***     Bank offsets
*
      PARAMETER ( MFQNFA=  1, MHSNFA= 65, MCPLFA= 67, MMTPFA= 68
     1           ,MLOCFA= 69, MHSTFA= 70, MHOSFA= 74
     2           ,MVSNFA= 77, MVIDFA= 79, MVIPFA= 81, MDENFA= 82
     3           ,MVSQFA= 83, MFSQFA= 84, MSRDFA= 85, MERDFA= 86
     4           ,MSBLFA= 87, MEBLFA= 88, MRFMFA= 89, MRLNFA= 90
     5           ,MBLNFA= 91, MFLFFA= 92, MFUTFA= 93, MCRTFA= 94
     6           ,MCTTFA= 95, MLATFA= 96, MCURFA= 97, MCIDFA= 99
     7           ,MCNIFA=101, MCJIFA=103, MFPRFA=105, MSYWFA=106
     8           ,MUSWFA=116, MUCMFA=126, NWDSFA=145
     9           ,MFSZFA=MSYWFA,MUSCFA=MSYWFA+1)

* End of FATMEN sequence FATPARA
      PARAMETER (LURCOR=200000)
      COMMON/CRZT/IXSTOR,IXDIV,IFENCE(2),LEV,LEVIN,BLVECT(LURCOR)
      DIMENSION    LQ(999),IQ(999),Q(999)
      EQUIVALENCE (IQ(1),Q(1),LQ(9)),(LQ(1),LEV)
      CHARACTER*(*) PATH
      PARAMETER     (LKEYFA=10)
      DIMENSION     KEYS(LKEYFA)
      DIMENSION     NDAYS(3)
      COMMON/QUEST/IQUEST(100)
      IRC   = 0
      LBANK = 0
      LP    = LENOCC(PATH)
      CALL FMGETK(PATH(1:LP),LBANK,KEYS,IRC)
*
*     Fill histograms
*
      IF(IQ(LBANK+MFSZFA).NE.0)
     +CALL HFILL(1,FLOAT(IQ(LBANK+MFSZFA)),0.,1.)
      IF(IQ(LBANK+MUSCFA).NE.0)
     +CALL HFILL(2,FLOAT(IQ(LBANK+MUSCFA)),0.,1.)
      CALL FMDAYS(PATH(1:LP),LBANK,KEYS,NDAYS,' ',IRC)
      CALL HFILL(3,FLOAT(NDAYS(3)),0.,1.)
      CALL HFILL(4,FLOAT(NDAYS(2)),0.,1.)
      CALL HFILL(5,FLOAT(NDAYS(1)),0.,1.)
      CALL HFILL(6,FLOAT(IQ(LBANK+MMTPFA)),0.,1.)
      CALL MZDROP(IXSTOR,LBANK,' ')
      END
```

Part III

FATMEN – User Guide

Chapter 6: Introduction to the FATMEN File System User Interface

The user may interface to the file system on one of two ways: either through FORTRAN callable routines, or by executing commands at the command line. It is expected that most access to data will be via the callable interface, although the command line interface will be useful for searching through the directory structure.

Although the FATMEN file catalogue makes use of ZEBRA for the management of the data structures, no knowledge of ZEBRA is required to use the interface routines, unless the user wishes to override the system defaults If the user needs to override default values, or access fields in the FATMEN catalogue, the PATCHY sequence `FATPARA` must be included in his routines. See page 50 for an example of how to extract the parameter offsets and page 233 for a description of the parameters and their meanings. This sequence defines the offsets into the FATMEN banks, e.g.

```
      CHARACTER*8 CHOST
*
*     Get name of host on which this file is stored
*
      CALL UHTOC(IQ(LFAT+MHSNFA),CHOST,4,8)
*
*     Check file sequence number
*
      ISEQ = IQ(LFAT+MFSQFA)
*
*     That's about as complicated as it gets.
*
```

6.1 Parameter offsets

The offsets to the various fields in the FATMEN banks are defined by the PATCHY sequence `FATPARA`. This sequence may be extracted as shown in the following example.

Extracting the sequence FATPARA

```
ypatchy fatmen.cards fatpara.f :go <<!
+use,*fatpara.
+exe.
+pam,11,t=c,a. fatmen.cards
+quit
!
```

6.2 Recent changes to FATMEN routines

Changes to the FATMEN code are logged in a patch named HISTORY on the FATMEN PAM. This patch lists improvements, bug fixes and outstanding work and can be extracted using the following patchy cradle:

Extracting the FATMEN history patch

```
+USE,HISTORY,T=LIS.
+PAM.
+QUIT.
```

6.3 Calling sequences and return codes

In all the routines listed below, a return code of zero indicates success and non-zero returns codes indicate failure. Some routines use the IQUEST vector to return error information. To test on results returned via IQUEST, the following definition should be added to the calling routine.

```
COMMON/QUEST/IQUEST(100)
```

In the following descriptions, the generic name argument `GENAM` or path name PATH should be declared as `CHARACTER*255`. All other arguments are of type INTEGER unless otherwise specified. Arguments followed by an asterisk (*) are output parameters. Arguments both preceeded and followed by asterisks are input parameters which are overwritten on output. All other arguments are input only. Optional arguments are specified by passing an INTEGER zero or a blank character, depending on the datatype.

Generic names and path names

Given the generic name `//CERN/DELPHI/P01_ALLD/CDST/S2PR/Y91V02/E091.3/L0678/R023808F01C1` the path name is `//CERN/DELPHI/P01_ALLD/CDST/S2PR/Y91V02/E091.3/L0678`. That is, the path name contains only the names of the directories where a file resides, whereas the generic name contains the full name.

Chapter 7: The FATMEN Fortran callable interface routines

7.1 Novice interface routines

These routines require no knowledge of Zebra to use FATMEN. They may be used in conjunction with the other FATMEN routines, provided that the same Zebra store is used.

Initialise FATMEN system

CALL **FMSTRT** (LUNRZ,LUNFZ,CHFAT,IRC*)

LUNRZ	Integer variable specifying the FORTRAN logical unit for the FATMEN RZ file.
LUNFZ	Integer variable specifying the FORTRAN logical unit to be used to send updates to the FATMEN server.
CHFAT	Character variable to specify the database and group name in the form `//database/group`, e.g. `//CERN/DELPHI`.
IRC	Integer variable in which the return code is returned.

This routine initialises ZEBRA, unless it has already been initialised, and then the FATMEN system. The store `/FAT/` in sequence FAT is used for this purpose. LUNRZ and LUNFZ are the logical units that will be used to access the database for reading and writing. If read-only access is required, LUNFZ should be set to 0. The database name CHFAT is a character string indicating the name of the database that is to be accessed. This field must be of the form '`//CERN/experiment`' for CERN experiments, '`//DESY/experiment`' for DESY experiments and so on.

On VM systems, the virtual card punch is used to communicate updates with the service machine that handles the database. As the punch may be in use for other purposes, both positive and negative values of LUNFZ are foreseen. If `LUNFZ > 0`, the punch will be used directly, which implies that it cannot be used by the calling programme for any other purpose. If `LUNFZ < 0`, a temporary file will be created and sent via SENDFILE to the server. The disk with most free space that is accessed in WRITE mode will be used for this purpose. If `LUNFZ > 0`, FATMEN will assume that it can write directly to the PUN device.

Example of using the `FMSTRT` routine

```
*     Initialise FATMEN for group CPLEAR
      CALL FMSTRT(1,2,'//CERN/CPLEAR',IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMSTRT'
```

Access a dataset

CALL **FMFILE** (LUN,GENAM,CHOPT,IRC*)

LUN	Integer variable specifying the FORTRAN logical unit to be used to access the dataset.
GENAM	Character variable specifying the generic name of the file to be accessed.
CHOPT	Character variable to specify the required options, as for the routine `FMOPEN` (see on Page 68.)
IRC	Integer variable in which the return code is returned.

This routine accesses the dataset referenced by GENAM on FORTRAN unit LUN.

Example of using the FMFILE routine

```
      CALL FMFILE(11,
     +'//CERN/CPLEAR/REAMS/M2T3/RAWD/NONE/NONE/R08288',' ',IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMFILE'
```

Deaccess a dataset

CALL **FMFEND** (LUN,GENAM,CHOPT,IRC*)

LUN	Integer variable specifying the FORTRAN logical unit used to access the dataset.
GENAM	Character variable specifying the generic name of the file accessed.
CHOPT	Character variable to specify the required options, as for the routine FMCLOS (see on Page 70.)
IRC	Integer variable in which the return code is returned.

This routine deaccesses the dataset referenced by GENAM on FORTRAN unit LUN.

Example of using the FMFEND routine

```
      CALL FMFEND(11,
     +'//CERN/CPLEAR/REAMS/M2T3/RAWD/NONE/NONE/R08288',' ',IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMFEND'
```

Add a tape file

CALL **FMADDT** (GENAM,VSN,VID,FSEQ,DSN,FFORM,CPLEV,HOST,RECFM,LRECL,LBLOCK,
FSIZE,MEDIA,COMM,IVECT,CHOPT,IRC*)

GENAM	Character variable specifying the generic name of the file to be added.
VSN	Character variable specifying the volume serial number or magnetically recorded label of the tape on which the file resides.
VID	Character variable specifying the visual identifier or the 'sticky' or external label of the tape on which the file resides. This field may also be given in the 'extended VID' form, e.g. IN2P3.EP1234 if a VID prefix is required.
FSEQ	Integer variable specifying the file sequence number on the specified tape volume of the file to be added.
DSN	Character variable specifying the dataset name of the file to be added.
FFORM	Character variable specifying the logical format of the file to be added, such as FX, EP, RZ etc.
CPLEV	Integer variable specifying the copy level (or data representation type) of the file to be added.
HOST	Character variable specifying the name of the host through which the dataset is to be accessed. For tape volumes this is typically the same as the node on which the dataset was created as it is not used to check access to the data.
RECFM	Character variable specifying the record format of the file to be
LRECL	Integer variable specifying the record length of the file to be added, in units of 32 bit words.
LBLOCK	Integer variable specifying the block size of the file to be added, in units of 32 bit words.
FSIZE	Integer variable specifying the size of the file in Megabytes.

MEDIA Integer variable specifying the media type, where 2=3480, 3=3420, 4=8200 etc. For tapes which are known to the TMS, this information is ignored.

COMM Character variable containing the user comment for this entry.

IVECT Integer array of length 10 containing the user words for this entry.

CHOPT Character variable specifying the required options (none at present).

- the entry is added using FMPUT (see Page 60).
- M the entry is added using FMMOD (see Page 60).
- N the entry is not added to the catalogue but may be further manipulated with the FMPEEK/FMPOKE routines described below.

IRC Integer variable in which the return code is returned.

This routine adds an entry to the FATMEN catalogue.

Example of using the FMADDT routine

```
      CALL FMADDT('//CERN/CNDIV/JAMIE/TEST','I29021','I29021',1
     +'FATMEN.CARDS','AS',0,'CERNVM','FB',20,800,1,2,
     +'Backup of source to FATMEN',IVECT,' ',IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMADDT'
```

Add a disk file

CALL **FMADDD** (GENAM,DSN,FFORM,CPLEV,HOST,RECFM,LRECL,LBLOCK, FSIZE,COMM,IVECT,CHOPT,IRC*)

GENAM Character variable specifying the generic name of the file to be added.

DSN Character variable specifying the dataset name of the file to be added.

FFORM Character variable specifying the logical format of the file to be added, such as FX, EP, RZ etc.

CPLEV Integer variable specifying the copy level (or data representation type) of the file to be added.

HOST Character variable specifying the name of the host on which the dataset resides.

RECFM Character variable specifying the record format of the file to be

LRECL Integer variable specifying the record length of the file to be added, in units of 32 bit words.

LBLOCK Integer variable specifying the block size of the file to be added, in units of 32 bit words.

FSIZE Integer variable specifying the size of the file in Megabytes.

COMM Character variable containing the user comment for this entry.

IVECT Integer array of length 10 containing the user words for this entry.

CHOPT Character variable specifying the required options.

- the entry is added using FMPUT (see Page 60).
- M the entry is added using FMMOD (see Page 60).
- N the entry is not added to the catalogue but may be further manipulated with the FMPEEK/FMPOKE routines described below.

IRC Integer variable in which the return code is returned.

This routine adds an entry to the FATMEN catalogue.

Example of using the FMADDD routine

```
      CALL FMADDD('//CERN/CNDIV/JAMIE/VAX2',
     +'DISK$CERN:[JAMIE.ZFTP]FXFILE.DAT',
     +'FX',0,'VXCERN','F',8100,8100,10,
     +'Test Zebra exchange format file for ZFTP',IVECT,' ',IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMADDD'
```

Return information on FATMEN entry

CALL **FMPEEK** (GENAM,IVECT,CHOPT,IRC*)

GENAM Character variable specifying the generic name of the file on which information is to be returned.

IVECT Integer array of length NWDSFA in which the FATMEN information is returned.

CHOPT Character variable specifying the required options.

- the last bank created by a call to FMADDT and FMADDD is returned.
- G the information is obtained from the FATMEN catalogue using the default FATMEN selection. (Note that there may be no entry with the current selection criteria - see the description of FMGET on page 59 for details).
- A the first entry found in the catalogue is returned.
- N the next entry found in the catalogue is returned, which may be the first if no previous call with CHOPT = 'A' has been issued.
- D Drop bank after unpacking

IRC Integer variable in which the return code is returned.

This routine returns in the vector IVECT the contents of the FATMEN bank associated with the specified generic name.

Example of using the FMPEEK routine

```
CALL FMPEEK('//CERN/CNDIV/JAMIE/VAX2',IVECT,'N',IRC)
IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMPEEK'
```

Add entry to catalogue

CALL **FMPOKE** (GENAM,IVECT,CHOPT,IRC*)

GENAM Character variable specifying the generic name of the file.

IVECT Integer array of length NWDSFA in which the FATMEN information is contained.

CHOPT Character variable specifying the required options.

- the information is simply copied into this bank.
- N the vector IVECT is ignored. This is useful if the bank has been modified using the FMPUTC/I/V routines.
- M the information is added to the catalogue using FMMOD. (see on Page 60).
- R the entry is replaced using FMMOD (see on Page 60).
- P the information is added to the catalogue using FMPUT. (see Page 60).

IRC Integer variable in which the return code is returned.

This routine copies the vector IVECT into the FATMEN bank reserved for manipulation with FMADDD/FMADDT, FMPEEK/FMPOKE.

Example of using the FMPOKE routine

```
      CALL FMPOKE('//CERN/CNDIV/JAMIE/VAX2',IVECT,'M',IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMPOKE'
```

7.2 Routines that manipulate the FATMEN catalogue

Initialise FATMEN system

CALL **FMINIT** (IXSTOR*,LUNRZ,LUNFZ,DBNAME,IRC*)

IXSTOR Integer variable to return the number of the store initialised for FATMEN.
LUNRZ Integer variable specifying the FORTRAN logical unit for the FATMEN RZ file.
LUNFZ Integer variable specifying the FORTRAN logical unit to be used to send updates to the FATMEN server.
DBNAME Character variable to specify the database and group name in the form //database/group, e.g. //CERN/DELPHI.
IRC Integer variable in which the return code is returned.

This routine initialises the FATMEN system. LUNRZ and LUNFZ are the logical units that will be used to access the database for reading and writing. If read-only access is required, LUNFZ should be set to 0. The database name DBNAME is a character string indicating the name of the database that is to be accessed. This field must be of the form '//CERN/experiment'.

On VM systems, the virtual card punch is used to communicate updates with the service machine that handles the database. As the punch may be in use for other purposes, both positive and negative values of LUNFZ are foreseen. If LUNFZ > 0, the punch will be used directly, which implies that it cannot be used by the calling programme for any other purpose. If LUNFZ < 0, a temporary file will be created and sent via SENDFILE to the server. The disk with most free space that is accessed in WRITE mode will be used for this purpose. If LUNFZ > 0, FATMEN will assume that it can write directly to the PUN device.

Example of using the FMINIT routine

```
*     Initialise FATMEN for group CPLEAR
      CALL FMINIT(IXSTOR,1,2,'//CERN/CPLEAR',IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMINIT'
```

Terminate FATMEN package

CALL **FMEND** (IRC*)

IRC* Integer variable in which the return code is returned.

This routine should be called when no further access to the FATMEN file catalogue is required, normally at program termination. This routine automatically calls the routine FMUPDT to ensure that any outstanding updates are sent to the server.

Example of using the routine FMEND

```
      CALL FMEND(IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMEND'
```

After a call to FMEND, a further call to FMINIT may be made, to look at the same or different FATMEN catalogue.

Set logging level of FATMEN package

CALL **FMLOGL** (LEVEL)

LEVEL Integer variable to set the level of logging required.

- -3 Suppress all log messages
- -2 Error messages
- -1 Terse logging
- 0 Normal (FMINIT, FMEND etc.)
- 1 Log calls to FATMEN routines (FORTRAN callable interface)
- 2 Log to monitor FATMEN internal decisions, such as selection of a dataset.
- 3 Debug messages

This routine establishes the LEVEL of diagnostic printing from the FATMEN package.

Example of using the routine FMLOGL

```
*     Set maximum logging level to monitor FATMEN progress
      CALL FMLOGL(3)
```

Control updating mode

CALL **FMUPDT** (MAX,NGROUP,IFLAG,IRC*)

MAX Integer variable specifying the maximum number of updates that may be performed. If this number is exceeded, the program will be terminated by a call to the Zebra routine ZFATAM.

NGROUP Integer variable specifying the number of updates that are to be grouped together.

IFLAG Integer variable which allows outstanding updates to be sent or purged, or to reset the system defaults.

IRC Integer variable specifying the return code.

This routine controls the updating mode of the FATMEN package. MAX is the maximum number of updates that may be issued by a single job. NGROUP is the number of updates to send together. If IFLAG=-1, the system defaults of MAX=999, NGROUP=0 (send each update separately) will be applied. If IFLAG=0, MAX and NGROUP will be reset as specified, with any outstanding updates sent immediately. If IFLAG=1, MAX and NGROUP will be reset as specified, with any outstanding updates purged.

Example of using the FMUPDT routine

```
*     Reset updating mode to the system defaults
      CALL FMUPDT(MAX,NGROUP,-1,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMUPDT
*     Cancel any outstanding updates and limit future updates to 10
      CALL FMUPDT(10,NGROUP,1,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMUPDT
```

Purge old entries from catalogue

CALL **FMPURG** (PATH,KEYSEL,MAXSIZ,MINACC,MAXDAYS,MINCPS,LUNPUR,CHOPT,IRC*)

PATH Character variable specifying the path name to be purged.

KEYSEL Integer array of length 10 specifying the KEY selection to be applied.

MAXSIZ Integer variable specifying the file size threshold, in Megabytes. Files with size <= MAXSIZ escape deletion.

MINACC Integer variable specifying the number of accesses. Files with accessed => MINACC escape deletion.

MAXDAYS Integer variable specifying the number of number of days permitted since last access. Files with accessed <= MAXDAYS ago escape deletion.

MINCPS Integer variable specifying the minimum number of copies required for a file to be candidate for deletion. Files with <= MINCPS escape deletion.

LUNPUR Integer variable specifying the FORTRAN logical unit to be used

CHOPT Character variable specifying the required options.

- P print usage statistics on these files
- K write a FATMEN KUMAC on LUNPUR to remove these entries
- R remove these entries

IRC Integer variable in which the return code is returned.

This routine searches for files matching the specified path and KEYSEL selection for entries eligible for deletion. A check may be bypassed by coding a -1 for the corresponding parameter. Thus, to purge files without checking on the number of accesses, specify a -1 for MINACC.

Example of using the routine FMPURG

```
      DIMENSION KEYSEL(10)
*     Restrict search to 3480s
      KEYSEL(MKMTFA) = 2
*     Restrict search to location code 1
      KEYSEL(MKLCFA) = 1
*     Make no check on Copy Level
      KEYSEL(MKLCFA) = -1
*
*     Print all files eligible for purge that are 80 MB or larger,
*     have been accessed less than 10 times, have not been accessed
*     in the last 60 days, and for which at least 3 copies exist.
*
      CALL FMPURG('//CERN/CNDIV/J*',KEYSEL,80,10,60,3,0,'P',IC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMPURG'
```

Get information on named file

CALL **FMGET** (GENAM,LBANK*,KEYS*,IRC*)

GENAM	Character variable of maximum length 255 to specify the generic name.
LBANK	Integer variable to return the address of the bank corresponding to the generic name GENAM.
KEYS	Integer array of length 10 to return the keys vector associated with the specified generic name.
IRC	Integer variable in which the return code is returned.

This routine returns the address of the bank LBANK and the keys vector KEYS for the specified generic name GENAM. Should multiple entries exist for the given generic name GENAM, the FATMEN system will choose the most appropriate, according to a simple algorithm. Should a specific copy be required, the routine `FMGETK` should be used. **Warning: any bank at the address `LBANK` will be dropped by this routine on input. See the example on page 35 for an example of using a link area to save the address of multiple banks, e.g. when calling this routine in a loop.** The procedure for selecting a given copy of a dataset is as follows:

- The routine `FMSELK` is used to select disk files residing at the current location which match the specified generic name.
- The first dataset which resides on the current node is taken.
- Should no such dataset exist, the procedure is repeated for datasets residing on 3480 cartridges.
- The first matching dataset residing on a cartridge in an accessible tape robot is taken.
- Should no such dataset exist, the first matching dataset on a manually mounted 3480, if any, will be taken.

Where access to remote data is available, the procedure will then continue, searching for datasets on remote disks, in remote robots and finally on remote manually mounted tapes.

Example of using the `FMGET` routine

```
*     Select a copy of a dataset using default FATMEN selection.
      CALL FMGET('//CERN/CNDIV/CHRIS/TAPE8',LBANK,KEYS,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMGET'
```

Get information on named file with key selection

CALL **FMGETK** (GENAM,LBANK*,*KEYS*,IRC*)

GENAM	Character variable of maximum length 255 to specify the generic name.
LBANK	Integer variable to return the address of the bank corresponding to the generic name GENAM.
KEYS	Integer array of length 10 to return the keys vector associated with the specified generic name. The keys vector is also used to select a particular copy of a dataset.
IRC	Integer variable in which the return code is returned.

This routine returns the address of the bank LBANK for the combination of the generic name GENAM and key vector KEYS specified. This allows the user to select a particular copy of a file. **Warning: any bank at the address LBANK will be dropped by this routine on input. See the example on page 35 for an example of using a link area to save the address of multiple banks, e.g. when calling this routine in a loop.**

Example of using the FMGETK routine

```
      Select a given copy of a dataset.
*     The vector MYKEYS was obtained from FMSELK.
      CALL FMGETK('//CERN/CNDIV/CHRIS/TAPE8',LBANK,MYKEYS(1,1),IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMGETK'
```

Add entry to FATMEN catalogue

CALL **FMPUT** (GENAM,LBANK,IRC*)

GENAM Character variable of maximum length 255 to specify the generic name.

LBANK Integer variable to input the address of the bank corresponding to the generic name specified.

IRC Integer variable in which the return code is returned.

This routine enters the information in the bank with address LBANK for the specified generic name GENAM in the database. If a matching dataset already exists for this generic name, nothing will be added to the database. In all other cases a new entry will be made. Before the entry is sent to the FATMEN server, the routine FMVERI is automatically called. Should FMVERI return a non-zero return code, the entry will not be sent to the server. This is to ensure that the FATMEN catalogue is correctly updated and that the data can be successfully retrieved. See the description of the FMCOMP routine on page 85 for details of the association of generic names and datasets.

Example of using the FMPUT routine

```
      CALL FMPUT('//CERN/CHARM2/TEST/DST1/ELEC/H020/NOM/E02/FILE1',
     +LBANK,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMPUT'
```

Modify existing entry

CALL **FMMOD** (GENAM,LBANK,IFLAG,IRC*)

GENAM Character variable of maximum length 255 to specify the generic name.

LBANK Integer variable to input the address of the bank corresponding to the generic name specified.

IFLAG Integer variable to control the mode of operation.

- 0 the entry for the generic name GENAM will be added if it does not exist, or replaced if it does.
- 1 the entry will be replaced if it exists but otherwise not added.

IRC Integer variable in which the return code is returned.

This routine is similar top the routine FMPUT, except that it also allows an existing entry to be modified. Before the entry is sent to the FATMEN server, the routine FMVERI is automatically called. Should FMVERI return a non-zero return code, the entry will not be sent to the server. This is to ensure that the FATMEN catalogue is correctly updated and that the data can be successfully retrieved. See the description of the FMCOMP routine on page 85 for details of the association of generic names and datasets.

Example of using the FMMOD routine

```
*
*     Update an existing entry with the number of Megabytes written
      IQ(LBANK+MFSZFA) = 200
      CALL FMMOD('//CERN/CNDIV/CHRIS/TAPE8',LBANK,1,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMMOD
```

Create a new FATMEN bank

```
CALL FMBOOK (GENAM,KEYS*,LADDR*,*LSUP*,JBIAS,IRC*)
```

GENAM Character variable of maximum length 255 to specify the generic name.
KEYS Integer array of length 10 to return the keys vector associated with the specified generic name.
LADDR Integer variable to return the address of the bank created.
LSUP if JBIAS < 0: address of the supporting up bank
if JBIAS = 0: address of the supporting previous bank
if JBIAS > 0: link bias in the supporting bank
JBIAS if JBIAS < 1: link bias in the supporting bank
if JBIAS = 1: create top-level bank
if JBIAS = 2: create stand-alone bank
IRC Integer variable in which the return code is returned.

This routine will create a new ZEBRA bank for the specified generic name GENAM and fill in default values. The user may then modify these before committing the changes via FMPUT. If IRC is non-zero, the IQUEST vector will contain the error condition signaled by MZBOOK. See the description of MZBOOK in the ZEBRA manual.

The address of the bank lifted is the users responsibility and should be saved in a link area.

Example of using the FMBOOK routine

```
*
      CALL FMBOOK(GNAME,KEYS,LUSRBK,LSUP,JBIAS,IRC)
```

FMBOOK returns warning conditions using the IQUEST vector. The following conditions may be reported:

IQUEST(11)0 if this generic name does not exist, 1 otherwise.
IQUEST(12)0 if the corresponding directories already exist, 1 otherwise.

Create a link to an existing catalogue entry

```
CALL FMLN (CHSRCE,CHTRGT,CHCOMM,IVECT,CHOPT,IRC*)
```

CHSRCE Character variable of maximum length 255 to specify the generic name of the link to an existing object.
CHTRGT Character variable of maximum length 255 specifying an existing generic name.
CHCOMM Character variable of maximum length 80 specifying the comment to be associated to the link.

IVECT Vector of 10 user words to be associated to the link.

CHOPT Character variable specifying the required options.

- C set the comment field to the string specified in CHCOMM
- U set the user words to the values in the vector IVECT

IRC Integer variable in which the return code is returned.

Use the FMLN routine to make a link to an existing catalogue entry.

In Unix parlence, the sourcefile is the real file and the targetfile the link that points to it. To avoid even further confusion, the same terminology is adopted here.

If the existing entry is itself a link, the link will point to the source of that link.

A link is identified by having location code 0. The source file name is stored in the FATMEN bank at the offset MFQNFA.

Links can be useful in the following scenario. DELPHI write single file 3480 cassettes which contain more than one run. There is a generic name for each run that points to the same file. Additionally, there is also a so-called SUMT entry (for summary tape). This can cause house keeping problems, particularly when making copies for data export.

A solution to the above problem is to make the run specific generic names *links*. Only the SUMT entries are copied, moved or deleted. When a file is accessed via a link, the link is automatically resolved. The FATMEN selection then procedes as if the sourcefile name had been given.

A limitation of the current implementation is that it does not cater for the situation when a single run is copied to hotter media, e.g. disk. This would involve some reworking of the selection logic and remains pending demand from actual usage.

Remove entry from FATMEN catalogue

```
CALL FMRM (GENAM,LBANK*,KEYS,IRC*)
```

GENAM Character variable of maximum length 255 to specify the generic name.

LBANK Integer variable to input the address of the bank corresponding to the generic name specified.

KEYS Integer array of length 10 to pass the keys vector associated with the specified generic name.

IRC Integer variable in which the return code is returned.

This routine marks the entry generic name GENAM for deletion. An entry is uniquely identified by the following information: Host name, DSN (for disk files), Location, VID, VSN, File sequence number (for tape files), contained in the bank at address LBANK. Files may only be deleted by the creator. If a non-zero key vector KEYS is input, KEYS(1) will be used to select a specific copy of a file for deletion. If only one match for the specified name exists, KEYS(1) may be zero. If more than one entry for the specified generic name exists, KEYS(1) must specify the copy that is to be deleted.

Example of using the FMRM routine

```
      DIMENSION KEYS(10)
      CALL VZERO(KEYS,10)
      LBANK = 0
      CALL FMRM('//CERN/CHARM2/TEST/DST1/ELEC/H020/NOM/E02/FILE1',
     +LBANK,KEYS,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMRM'
```

Remove a link from a FATMEN catalogue

```
CALL FMRMLN (CHLINK,LUN,CHFILE,CHOPT,IRC*)
```

CHLINK Character variable specifying the path to be searched for dangling links, containing wild-cards as necessary

LUN Unit number on which the file CHFILE is written, with option F

CHFILE Character variable specifying the name of the file to be written, with option F.

CHOPT Character variable specifying the required options.

- P Print the names of dangling links
- D Write the names of dangling links in the form `rm generic-name ksn`
- R Remote dangling links
- F Redirect output to the file CHFILE on the unit LUN

IRC Integer variable in which the return code is returned.

Make directory

```
CALL FMKDIR (CHDIR,IRC*)
```

CHDIR Character variable of maximum length 255 to specify the name of the directory to be created.

IRC Integer variable in which the return code is returned.

This routine creates the specified directory CHDIR.

Example of using the routine FMKDIR

```
      CALL FMKDIR('//CERN/ALEPH/MC/TEST',IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMKDIR
```

FMKDIR returns warning conditions using the IQUEST vector. The following conditions may be reported:

IQUEST(12)0 if the specified directory already exists, 1 otherwise.

7.3 Routines to modify the contents of the FATMEN banks

Set contents of FATMEN bank

```
CALL FMFILL (GENAM,*LBANK*,*KEYS*,CHOPT,IRC*)
```

GENAM Character variable of maximum length 255 to specify the generic name.

LBANK Integer variable to return the address of the bank corresponding to the input generic name GENAM. If LBANK is non-zero, the bank at this address will be used.

KEYS Integer array of length 10 to return the keys vector associated with the specified generic name. If the keys vector is non-zero, it will be used by FMGETK to return the bank corresponding to a particular copy of a dataset.

CHOPT Character variable to specify the options desired.

- A set all fields
- C clear comment field

- `F` zero file attributes, such as start/end record and block
- `K` reset keys to match generic name, default copy level, media type, location
- `L` clear logical attributes, such as FATMEN file format
- `M` clear media attributes, such as VSN, VID, file sequence number for tape files, set host type and operating system for disk files.
- `N` clear dataset name on disk/tape of this file
- `O` set owner, node and job of creator etc.
- `P` clear physical attributes, such as record format etc.
- `S` clear security details of this file (protection)
- `T` set date and time of creation, last access etc.
- `U` clear user words.
- `Z` display ZEBRA bank with FMSHOW.

`IRC` Integer variable in which the return code is returned.

This routine sets the contents of the FATMEN bank corresponding to the input generic names GENAM, or at the address LBANK if non-zero. This routine either sets fields that can be automatically obtained, such as the date and time or current node, or sets them to zero or blanks as appropriate (user comment, vsn, vid etc.)

Example of using the `FMFILL` routine

```
*     Update owner information and time fields.
*     Z option causes the resultant bank to be displayed
*     via the routine FMSHOW.
*
      CALL FMFILL('//CERN/L3/PROD/DATA/SDSUEE/CC00DCVY',
     +LBANK,KEYS,'OTZ',IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMFILL'
```

Insert character data into FATMEN bank

`CALL` **FMPUTC** `(LBANK,STRING,ISTART,NCH,IRC*)`

`LBANK` Integer variable containing the address of the bank to be updated, or -1, when called from the novice interface.

`STRING` Character variable containing the data to be inserted into the bank.

`ISTART` Offset at which this data should be written, e.g. MUSCFA

`NCH` Number of characters to write.

`IRC` Integer variable in which the return code is returned.

This routine updates the FATMEN bank at address LBANK with the character data in the string STRING. Attempts to write outside of bank boundaries will generate an error and a non-zero value of the return code IRC.

When called from the novice interface, LBANK should be set to -1.

Example of using the FMPUTC routine

```
      CALL FMPUTC(LBANK,'I28901',MVSNFA,6,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMPUTC'
```

Read character data from FATMEN bank

CALL **FMGETC** (LBANK,STRING*,ISTART,NCH,IRC*)

LBANK	Integer variable containing the address of the bank from which information is to be retrieved, or -1, when called from the novice interface.
STRING	Character variable in which the character data is returned.
ISTART	Offset at which this data should be read, e.g. MUSCFA
NCH	Number of characters to read.
IRC	Integer variable in which the return code is returned.

This routine returns character information from the FATMEN bank at address LBANK in the string STRING. Attempts to read outside of bank boundaries will generate an error and a non-zero value of the return code IRC.

When called from the novice interface, LBANK should be set to -1.

Example of using the FMGETC routine

```
      CALL FMGETC(LBANK,VSN,MVSNFA,6,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMGETC'
```

Insert integer vector into FATMEN bank

CALL **FMPUTV** (LBANK,IVECT,ISTART,NWORDS,IRC*)

LBANK	Integer variable containing the address of the bank to be updated, or -1, when called from the novice interface.
IVECT	Integer vector containing the data to be inserted into the bank.
ISTART	Offset at which this data should be written, e.g. MUSWFA
NCH	Number of words to write.
IRC	Integer variable in which the return code is returned.

This routine updates the FATMEN bank at address LBANK with the data in the vector IVECT. Attempts to write outside of bank boundaries will generate an error and a non-zero value of the return code IRC.

When called from the novice interface, LBANK should be set to -1.

Example of using the FMPUTV routine

```
*     Insert user words vector into FATMEN bank
      CALL FMPUTV(LBANK,IWORDS,MUSWFA,10,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMPUTV'
```

Read integer vector from FATMEN bank

`CALL` **FMGETV** `(LBANK,IVECT*,ISTART,NWORDS,IRC*)`

`LBANK`	Integer variable containing the address of the bank from which information is to be retrieved, or -1, when called from the novice interface.
`IVECT`	Integer vector in which the data is returned.
`ISTART`	Offset at which this data should be read, e.g. MUSWFA
`NCH`	Number of characters to read.
`IRC`	Integer variable in which the return code is returned.

This routine returns the information from the FATMEN bank at address LBANK in the vector IVECT. Attempts to read outside of bank boundaries will generate an error and a non-zero value of the return code IRC.

When called from the novice interface, LBANK should be set to -1.

Example of using the `FMGETV` routine

```
      CALL FMGETV(LBANK,IWORDS,MUSWFA,10,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMGETV'
```

Insert integer value into FATMEN bank

`CALL` **FMPUTI** `(LBANK,IVAL,IOFF,IRC*)`

`LBANK`	Integer variable containing the address of the bank to be updated, or -1, when called from the novice interface.
`IVAL`	Integer variable containing the data to be inserted into the bank.
`IOFF`	Offset at which this data should be written, e.g. MUSWFA
`IRC`	Integer variable in which the return code is returned.

This routine updates the FATMEN bank at address LBANK with the data in the variable IVAL. Attempts to write outside of bank boundaries will generate an error and a non-zero value of the return code IRC.

Example of using the `FMPUTI` routine

```
*     Insert use count into FATMEN bank
      CALL FMPUTI(LBANK,NUSE,MUSCFA,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMPUTI'
```

Read integer value from FATMEN bank

```
CALL FMGETI (LBANK,IVAL*,IOFF,IRC*)
```

`LBANK` Integer variable containing the address of the bank from which information is to be retrieved, or -1, when called from the novice interface.

`IVAL` Integer variable in which the data is returned.

`ISTART` Offset at which this data should be read, e.g. MUSWFA

`IRC` Integer variable in which the return code is returned.

This routine returns the information from the FATMEN bank at address LBANK in the variable IVAL. Attempts to read outside of bank boundaries will generate an error and a non-zero value of the return code IRC.

When called from the novice interface, LBANK should be set to -1.

Example of using the FMGETI routine

```
      CALL FMGETI(LBANK,NUSE,MUSCFA,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMGETI'
```

7.4 Routines that provide access to the data

Find existing dataset and associate with logical unit

```
CALL FMFIND (GENAM,DDNAME,*LBANK*,IRC*)
```

`GENAM` Character variable of maximum length 255 to specify the generic name.

`DDNAME` Character variable of maximum length 8 to specify the FORTRAN logical unit.

`LBANK` Integer variable to return the address of the bank corresponding to the generic name GENAM.

`IRC` Integer variable in which the return code is returned.

This routine returns the bank address LBANK of the most suitable copy of the data referenced by the specified generic name GENAM. **N.B. if LBANK is non-zero, the bank at this address will be used. This allows the user to make their own selection, e.g. by first calling `FMGET`, `FMGETK` or `FMSELM`, or to override some parameters in the Zebra bank for the specified generic name. If you do not wish to use this facility, you must drop the bank at LBANK using MZDROP.**

When the calling program receives control, it may open the file on the logical unit corresponding to the DDNAME specified and read the data. Should the required data reside on tape, the FATMEN system will check access to the tape using the Tape Management System and STAGE the data on to disk.

The ddname, of type CHARACTER, may be specified as 'nn', e.g. 1 or 99, or as any valid logical unit such as FT93F005, IOFILE11, FOR003, fort.20 etc. If a one or two digit character is specified, it will be converted to the format used by FORTRAN on the host machine. (For files in EP or FX format on VM/CMS systems, the format IOFILEnn will be used). If the input bank-address is non-zero, the information in the bank to which it points will be used to accessed the data. If the bank-address is zero, `FMLIFT` will automatically call `FMGET`.

On VM/CMS systems only, a DDNAME of NOWAIT will result in a STAGE request with the NOWAIT option. This may be used to initialise the input staging of the tape **i+1** while tape **i** is being processed. See the tutorial section of this manual for an example of using the NOWAIT option.

Example of using routine FMFIND

```
*     Find a dataset using the default FATMEN selection
      CALL FMFIND('//CERN/CNDIV/CHRIS/TAPE8','IOFILE13',LFAT,IRC)
      IF(IRC.NE.0) PRINT *,'Return code',IRC,' from FMFIND.'
```

Create new dataset

CALL **FMMAKE** (GENAM,DDNAME,*LBANK*,IRC*)

GENAM	Character variable of maximum length 255 to specify the generic name.
DDNAME	Character variable of maximum length 8 to specify the FORTRAN logical unit.
LBANK	Integer variable to return the address of the bank corresponding to the generic name GENAM.
IRC	Integer variable

This subroutine creates a new disk or tape dataset, according to the contents of the bank passed at LBANK Before calling FMMAKE, the user should call FMBOOK (see on Page 61) and then set the various fields as required.

See the description of the FMFIND routine for information on the DDNAME parameter. **N.B. if LBANK is non-zero, the bank at this address will be used. This allows the user to make their own selection, e.g. by first calling FMGET, FMGETK or FMSELM, or to override some parameters in the Zebra bank for the specified generic name. If you do not wish to use this facility, you must drop the bank at LBANK using MZDROP.**

Example of using the routine FMMAKE

```
*     Create dataset on IOFILE11.
*     The bank at address LBANK was created by FMLIFT
      CALL FMMAKE('//CERN/ALEPH/MDST/KELLNER/RUN123','IOFILE11',LBANK,IRC)
```

Open a dataset for read or write

CALL **FMOPEN** (GENAM,DDNAME,*LBANK*,CHOPT,IRC*)

GENAM	Character variable of maximum length 255 to specify the generic name.
DDNAME	Character variable to specify the FORTRAN logical unit to be opened.
LBANK	Integer variable to input the address of the bank corresponding to the generic name specified.
CHOPT	Character variable to specify the options desired.
IRC	Integer variable in which the return code is returned.

This routine accesses a dataset via its generic name GENAM and opens it on the logical unit specified by the character DDNAME. The following values are allowed for **CHOPT**.

- D Make a duplicate into the SMCF robot (CERN only)
- E When used with option T, add END option to SETUP command
- F Issue call to FZFILE If option F is specified, IQUEST(10) may be set to indicate which form of I/O should be used. The default is FORTRAN I/O.

```
IQUEST(10) = 1 : C I/O
                 Corresponds to option L in FZFILE
IQUEST(10) = 2 : VM only : VMIO
                 (via FZHOOK to FATMEN routine)
IQUEST(10) = 3 : Package I/O, i.e. IOPACK on VM or MVS systems
                 Corresponds to option Y in FZFILE
```

H Stage wHole tape. This option is ignored unless running on a VAX/VMS system using the VAXTAP [13] package.
K KEEP option on STAGE OUT
L Override DCB (record format, record length, block length) in tape label with information in FATMEN bank/catalogue. (Default on output and for VM disk files, should only be used on input if tape labels are bad).
N Do not specify DSN or FILEID on STAGE or LABELDEF command
O Override SIZE information with value specified in IQUEST(11)
P AutoPut option in STAGE OUT
Q Queue stage request (e.g. NOWAIT option for VMSTAGE). This option queues the staging request and then returns without waiting for the staging operation to complete.
R Read access
S Update FATMEN catalogue with file size, as obtained from staging system if filesize field is zero and return code from staging system is zero.
T Read/write directly to tape (i.e. do not use STAGE).
U OPEN will be performed by User
V As S, but even if file size is non-zero. If the file size in the catalogue and that returned by VMSTAGE disagree, a warning message will be issued, but the information obtained from VMSTAGE will nevertheless be used to update the catalogue.
W Write access
X Direct access file
Y Do not issue STAGE command, but write to file on unit DDNAME
Z Issue RZOPEN and RZFILE

N.B. if LBANK is non-zero, the bank at this address will be used. This allows the user to make their own selection, e.g. by first calling FMGET, FMGETK or FMSELM, or to override some parameters in the Zebra bank for the specified generic name. If you do not wish to use this facility, you must drop the bank at LBANK using MZDROP.

Example of using the FMOPEN routine

```
      CALL FMOPEN('//CERN/CHARM2/TEST/DST1/ELEC/H020/NOM/E02/FILE3',
     +            '3',LBANK,'R',IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMOPEN
```

Option D, when used with option R, will call the routine FMSMCF to automatically copy the data into the robot. Note that this facility is currently CERNVM specific, and requires that

1 The dataset resides on tape and is successfully STAGEd in by FMOPEN (i.e. option T not allowed).
2 No dataset with the specified generic name currently exists in the robot.
3 A pool of tapes gg_FAT1 exists (i.e. WS_FAT1).

Option Y is valid for stage operations only. Instead of issuing the stage command, it will be written to the file specified by the DDNAME parameter. No further action will be taken.

Shift/CORE specific considerations

When running on systems that have the SHIFT/CORE software installed, the default access method for ZEBRA FZ files is via the XYOPEN/XYREAD interface to **RFIO**. FATMEN calls the FZ routine FZHOOK to channel I/O through a special routine that interfaces to **RFIO**. Alternatively, one may select the C I/O option as described above. In this case, the **CFIO** routines will be used. On SHIFT/CORE systems, these are compiled in such a way as to interface to **RFIO**.

To ensure that all staged files are placed in a common directory, one should set the environmental variables **DPMUSER** and **STAGE_USER** as shown below.

Example of setting the Shift environmental variables

```
setenv DPMUSER pubxx
setenv STAGE_USER pubxx
```

Close file opened via FATMEN

CALL **FMCLOS** (GENAM,DDNAME,LBANK,CHOPT,IRC*)

GENAM	Character variable of maximum length 255 to specify the generic name.
DDNAME	Character variable to specify the name of the logical unit to be closed. The DDNAME is specified as for the routines FMOPEN, FMFIND and FMMAKE.
LBANK	Integer variable to specify the address of the bank to be written to the FATMEN catalogue.
CHOPT	Character variable to specify the required options.
IRC	Integer variable in which the return code is returned.

This routine closes a file IUNIT previously opened via FMOPEN. If LBANK is non-zero, the information concerning the specified generic name GENAM is written back to the database. This would typically be used by a production job, which would update the bank information with the number of records and total amount of data written, before updating the file catalogue. The following values are allowed for **CHOPT**.

C The data, if STAGEd, is cleared.

D DROP the Staging or maxi-disk (VM), Dismount the tape volume (option T on FMOPEN) or Deassign the logical name created by FMOPEN (VMS).

E Call the appropriate Zebra termination routine for this file. If the file was opened via a call to FMOPEN, FMCLOS will automatically call FZENDI, FZENDO or RZEND as appropriate.

F Update the bank at LBANK with the file size as obtained from FZINFO.

N Do not issue close (FORTRAN, VMIO or other) for output file

P For output staging only: request that the writing of the data to tape begins (STAGE PUT).

U Update FATMEN catalogue with bank at LBANK

W Wait for STAGE PUT command to complete

Z Drop bank at LBANK with MZDROP

Example of using the routine `FMCLOS`

```
*
      CALL FMCLOS('//CERN/CNDIV/CHRIS/TAPE8','FT11F002',LBANK,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMCLOS'
```

Copy a dataset and update the FATMEN catalogue

`CALL` **FMCOPY** `(GN1,*LBANK1*,*KEYS1*,GN2,*LBANK2*,*KEYS2*,CHOPT,IRC*)`

`GNn` Character variable of maximum length 255 to specify the generic names of the files to be copied.

`LBANKn` Integer variable(s) to return the addresses of the banks corresponding to GN1 and GN2.

`KEYSn` Integer array of length 10 to return the keys vectors associated with the specified generic names.

`CHOPT` Character variable specifying the required options.

`IRC` Integer variable in which the return code is returned.

This routine copies the dataset pointed to by the generic name GN1 to the location pointed to by GN2. (Typically GN1 will be the same as GN2). If either LBANK1 or LBANK2 is non-zero, the bank(s) at the corresponding address will be used. If either LBANK1 or LBANK2 are zero, the routine `FMGETK` will be used to obtain the corresponding bank. Should KEYS1(1) or KEYS2(1) be zero, the default FATMEN selection will apply. If CHOPT='S', the input file will first be STAGEd and then the STAGE CHANGE command used to write the output tape (assuming that input and output media are tape). In all other cases, the copy will be performed using the VMIO package.
The following options may be specified:

A Input data is already STAGEd.
C Use STAGE CHANGE / stagewrt/ WRTAPE to write the output file.
F Copy file using FZIN/FZOUT (permits change of FZ formats)
K Queue copy for transfer by `CHEOPS` using `FMCOPQ`
R Skip Zebra start of run / end of run records (with F)
S Stage the input file
Z Stage the output file

Example of using the `FMCOPY` **routine**

```
*     In this example, both datasets are already catalogued
*     IN points to the FATMEN source (FATMEN CARDS)
*     OUT points to a robot tape. FMCOPY copies IN to OUT
*     converting RECFM F to RECFM FB (due to catalogue entries)
      CALL FMCOPY('//CERN/CNDIV/JAMIE/IN' ,LBANK1,KEYS1,
     +            '//CERN/CNDIV/JAMIE/OUT',LBANK2,KEYS2,'S',IRC)
```

N.B. The `FMCOPY` routine allocates FORTRAN logical units to perform the copy using the routine `FMGLUN` (see on Page 107). These are freed immediately after use by `FMFLUN`. For `FMCOPY` to work, first allocate some units for use by FATMEN using the routine `FMSETU` (see on Page 107).

On VM systems, FATMEN uses the VMIO logical units corresponding to the two units declared in `FMSTRT` (see on Page 52) or `FMINIT` (see on Page 56). For example, if `FMINIT` is called with logical units 1 and 2, `FMCOPY` will use `VM01F001` and `VM02F001`.

When the copy is performed by STAGE CHANGE (stagewrt on Cray, WRTAPE on VAX), no additional logical units are required.

Queue a copy request

```
CALL FMCOPQ (GN1,*LBANK1*,*KEYS1*,GN2,*LBANK2*,*KEYS2*,CHOPT,IRC*)
```

- `GNn` Character variable of maximum length 255 to specify the generic names of the files to be copied.
- `LBANKn` Integer variable(s) to return the addresses of the banks corresponding to GN1 and GN2.
- `KEYSn` Integer array of length 10 to return the keys vectors associated with the specified generic names.
- `CHOPT` Character variable specifying the required options.
- `IRC` Integer variable in which the return code is returned.

This routine is called by `FMCOPY` when option K is specified. Rather than perform the copy directly, it is queued for transfer by `CHEOPS`. This is performed by writing a request file in a special queue directory. `FMCOPQ` may only be used at sites that are `CHEOPS` partners.

Copy a dataset over the network and update the FATMEN catalogue

```
CALL FMRCOP (GN1,*LBANK1*,*KEYS1*,GN2,*LBANK2*,*KEYS2*,CHOPT,IRC*)
```

- `GNn` Character variable of maximum length 255 to specify the generic names of the files to be copied.
- `LBANKn` Integer variable(s) to return the addresses of the banks corresponding to GN1 and GN2.
- `KEYSn` Integer array of length 10 to return the keys vectors associated with the specified generic names.
- `CHOPT` Character variable specifying the required options.
- `IRC` Integer variable in which the return code is returned.

This routine is called by `FMCOPY` when it determines that a network copy is required.

7.5 Routines to select or list catalogue entries

Check whether generic name already exists

```
CALL FMEXST (GENAM,IRC*)
```

- `GENAM` Character variable of maximum length 255 to specify the generic name.
- `IRC` Integer variable in which the number of occurances of GENAM are returned.

This routine returns the number of occurances of the input generic name GENAM in the FATMEN database. A return code of 0 indicates, therefore that no matches are found, whereas a return code of 3 would indicate that 3 copies were found. Routine `FMSELK` (see on Page 82) can be used to count the number of files with certain characteristics, such as location code or media type.

Example of using the `FMEXST` routine

```
      CALL FMEXST('//CERN/CNDIV/JAMIE/IN',IRC)
      PRINT *,IRC,' occurances in FATMEN database'
```

List files in specified directory

```
CALL FMLS (GENAM,CHOPT,IRC*)
```

GENAM Character variable of maximum length 255 to specify the generic name.
CHOPT Character variable to specify the options desired.
IRC Integer variable in which the return code is returned.

This routine lists the files according to the specified generic name GENAM. Examples of valid generic-names are:

```
//cern/cndiv/goossens/*,
//cern/cndiv/goossens/dcf/m%%%,
//cern/cndiv/chris/tape8.
```

The generic-name may also contain numeric ranges specified as (mm:nn), such as (13:147) in the field following the last delimiter (/). See on Page 81 for more details. If `IRC < 0`, then the specified pathname does not exist or is invalid. Otherwise, IRC returns the number of files found (which may be 0).

Example of using the FMLS routine

```
*     List media details of files M%%% in //CERN/CNDIV/GOOSSENS/DCF
      CALL FMLS('//CERN/CNDIV/GOOSENS/DCF/M%%%','M',IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMLS'
```

The allowed values of CHOPT and their meanings are as follows:

A list all attributes, except DZSHOW (option Z).
B brief (80) column listing
C display comment field associated with file
D to be used to generate a KUMAC file to remove (delete) the entry
E extended (132) column listing
F list file attributes, such as start/end record and block
G list the full generic name of each file
H write header line]useful in the case of output redirection
I sort generic names in Increasing order
J Just show entries that are accessible
K list keys associated with this file (copy level, media type, location)
L list logical attributes, such as FATMEN file format (ZEBRA exchange etc.)
M list media attributes, such as VSN, VID, file sequence number for tape files, host type and operating system for disk files.
N lists dataset name on disk/tape of this file
O list owner, node and job of creator etc.
P list physical attributes, such as record format etc.
Q obtain volume information from Tape Management System (TMS) if the entry corresponds to a tape file, and if the TMS option is installed.
R display where the data Reside.

`Disk files` Displays if the file is accessible and the access method, if known. (e.g. local disk, NFS, AFS, DFS etc.)

`Tape files` Displays if the associated volume is in an active library, i.e. not archived, whether it is staged, and whether a device of the appropriate type exists on the local node or is served.

`S` lists security details of this file (protection)

`T` list date and time of creation, last access etc.

`U` list user words.

`X` display only one entry per generic name (implies I)

`W` list generic names across the page (default is one name per line)

`Z` dump ZEBRA bank with DZSHOW.

Display contents of FATMEN bank

`CALL` **FMSHOW** `(GENAM,*LBANK*,*KEYS*,CHOPT,IRC*)`

`GENAM` Character variable of maximum length 255 to specify the generic name.

`LBANK` Integer variable to return the address of the bank corresponding to the input generic name GENAM. LBANK can have the following values:

- `>0` The bank at this address will be used.
- `0` The corresponding bank will be retrieved from the FATMEN catalogue
- `<0` The last bank referenced by one of the novice interface routines will be used.

`KEYS` Integer array of length 10 to return the keys vector associated with the specified generic name. If the keys vector is non-zero, it will be used by `FMGETK` to return the bank corresponding to a particular copy of a dataset.

`CHOPT` Character variable to specify the options desired.

`IRC` Integer variable in which the return code is returned.

This routine displays the contents of the FATMEN bank corresponding to the input generic names GENAM, or at the address LBANK if non-zero. See the description of the **ls** command for a description of the CHOPT argument. It is envisaged that this routine will be tailored to suit the requirements of individual experiments.

Example of using the `FMSHOW` routine

```
*
      CALL FMSHOW('//CERN/CNDIV/GOOSENS/DCF/M123',LBANK,KEYS,'M',IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMSHOW'
```

The allowed values of CHOPT and their meanings are

`A` list All attributes, except DZSHOW (option Z).

`B` Brief (80) column listing

`C` display Comment field associated with file

`D` to be used to generate a KUMAC file to remove (Delete) the entry

`E` Extended (132) column listing

`F` list File attributes, such as start/end record and block

G list the full Generic name of each file
H write Header line - useful in the case of output redirection
I sort generic names in Increasing order
J Just show entries that are accessible
K list Keys associated with this file (copy level, media type, location)
L list Logical attributes, such as FATMEN file format (ZEBRA exchange etc.)
M list Media attributes, such as VSN, VID, file sequence number for tape files, host type and operating system for disk files.
N lists dataset Name on disk/tape of this file
O list Owner, node and job of creator etc.
P list Physical attributes, such as record format etc.
Q obtain volume information from Tape Management System (TMS) if the entry corresponds to a tape file, and if the TMS option is installed.
R display where the data Reside.

  `Disk files` Displays if the file is accessible and the access method, if known. (e.g. local disk, NFS, AFS, DFS etc.)

  `Tape files` Displays if the associated volume is in an active library, i.e. not archived, whether it is staged, and whether a device of the appropriate type exists on the local node or is served.

S lists Security details of this file (protection)
T list date and time of creation, last access etc.
U list user words.
X display only one entry per generic name (implies I)
W list generic names across the page (default is one name per line)
Z dump ZEBRA bank with DZSHOW.

Count file names

CALL **FMFILC** (GENAM,NFILES*,IRC*)

GENAM Character variable of maximum length 255 to specify the generic name of interest.
NFILES Integer variable in which the number of matching files is returned.
IRC Integer variable in which the return code is returned.

This routine returns the number of files that match the specified generic name. The generic name is interpreted as for the FMLS routine (see on Page 73. That is, any characters following the last slash are assumed to be the filename, those preceeding the last slash are the pathname.

Example of using the FMFILC routine

```
      CALL FMFILC('//CERN/DELPHI'//
     +            '/ALLD/RAWD/CERN/V00%/E09%.00/P01R*/NONE/RUN*',
     +            NFILES,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMFILC'
```

Scan FATMEN directory structure

```
CALL FMSCAN (PATH,NLEVEL,UROUT,IRC*)
```

`PATH`	Character variable of maximum length 255 to specify the path name of interest.
`NLEVEL`	Integer variable specified the number of levels to descend in the directory structure.
`UROUT`	Address of a user routine that is called for each matching directory.
`IRC`	Integer variable in which the return code is returned.

This routine scans the directory names in the directory tree specified by PATH. The directory tree may contain wild-cards (* or %) in any position, or numeric ranges specified as (mm:nn), such as (13:147). See on Page 81 for more details. The path name PATH may also contain the wild-cards < or >. These wild cards may be used to find, for example, the highest PASS of a run. Assuming that, at a given level, the generic name has a field containing PASSn, e.g. PASS3, PASS1, specifying PASS> will cause `FMSCAN` to follow **ONLY** the PASS3 subdirectory

For each matching directory, `FMSCAN` will call the specified user routine with the matching directory name. Upon exit from the user routine, a non-zero return code will cause `FMSCAN` to stop the search.

On entry to the user routine, IQUEST contains the following information:

`IQUEST(10)`	number of elements in path passed to `FMSCAN`
`IQUEST(12)`	number of subdirectories
`IQUEST(13)`	number of elements in current path
`IQUEST(14)`	number of keys
`IQUEST(15)`	number of elements per key

In addition, on return from `FMSCAN`, IQUEST(11) contains the total number of directories processed.

Example of calling the routine `FMSCAN`

```
      EXTERNAL FUSCAN
*
*     This should follow the path PASS5/GPMH/P5
*
      ICONT = 0
 1    CONTINUE
      CALL FMSCAN('//CERN/OPAL/PROD/PASS>/GPMH/P<',99,FUSCAN,IRC)
```

Example of a user exit routine

```
      SUBROUTINE UROUT(PATH,IRC)
      CHARACTER*(*) PATH
      COMMON/QUEST/IQUEST(100)
      IRC = 0
*
*     Set IRC^=0 to stop directory scan
*
      PRINT *,'<< ',PATH(1:LENOCC(PATH))
      END
```

Loop through FATMEN file names

```
CALL FMLOOP (GENAM,NLEVEL,UROUT,IRC*)
```

GENAM	Character variable of maximum length 255 to specify the generic name of interest.
NLEVEL	Integer variable specified the number of levels to descend in the directory structure.
UROUT	Address of a user routine that is called for each matching file.
IRC	Integer variable in which the return code is returned.

This routine is similar to FMSCAN, but expects a complete generic name on input, rather than just a path name.

For each matching generic name, FMLOOP will call the specified user routine with the generic name and keys vector. Upon exit from the user routine, a non-zero return code will cause FMLOOP to stop the search.

Upon entry to the user routine, IQUEST will contain the following information.

IQUEST(10)	number levels in initial path
IQUEST(11)	number of directories found
IQUEST(12)	number of subdirectories at this level
IQUEST(13)	number of elements in path name
IQUEST(14)	number of keys
IQUEST(15)	number of words per key
IQUEST(16)	incremental count of number of keys selected
IQUEST(17)	number of this key vector

This routine respects the values set for medium type, location code and copylevel by the routines FMSETM, FMSETL and FMSETC respectively.

Upon return from the user routine, IRC < 0 will cause FMLOOP to skip to the next directory, IRC > 0 will cause FMLOOP to return.

Example of calling the routine FMLOOP

```
      EXTERNAL      UROUT
*
*     List all files in all directories beginning with a J
*     down to the maximum (99) number of levels
*
      CALL FMLOOP('//CERN/CNDIV/J*/*',99,UROUT,IRC)
```

Example of a user exit routine

```
      SUBROUTINE UROUT(GENAM,KEYS,IRC)
      CHARACTER*(*) GENAM
      PARAMETER     (LKEYFA=10)
      DIMENSION     KEYS(LKEYFA)
      IRC = 0
      PRINT *,'Generic name = ',GENAM(1:LENOCC(GENAM))
      PRINT *,'Keys: '
      CALL FMPKEY(KEYS,LKEYFA)
      END
```

Return directory names in directory structure

CALL **FMLDIR** (PATH,DIRS*,NFOUND,MAXDIR,*ICONT*,IRC*)

PATH	Character variable of maximum length 255 to specify the path name of interest.
DIRS	Character array of length 255 characters and dimension MAXFIL to return the names of the directories found.
NFOUND	Integer variable to return the actual dimension of DIRS.
MAXFIL	Integer constant to specify the dimension of DIRS.
ICONT	Integer variable to indicate whether this is a continuation of a previous call, or a new call. If more directories are found than can be returned in a single call, ICONT will be set to 1.
IRC	Integer variable in which the return code is returned.

This routine returns the directory names in the directory tree specified by PATH. The directory tree may contain wild-cards (* or %) in any position, or numeric ranges specified as (mm:nn), such as (13:147). See on Page 81 for more details. The path name PATH may also contain the wild-cards < or >. These wild cards may be used to find, for example, the highest PASS of a run. Assuming that, at a given level, the generic name has a field containing PASSn, e.g. PASS3, PASS1, specifying PASS> will cause FMLDIR to follow **ONLY** the PASS3 subdirectory structure. Although the method used by FMLDIR is slightly slower for a complete directory scan than FMLIST, it normal cases it will be much faster as it follows only the paths which match the input path name. NFOUND returns the actual number of directories returned. IRC = -1 : more directories found than can be returned in DIRS(MAXDIR) FMLDIR should be called again with ICONT=0 to get the next MAXDIR batch of DIRS.

Example of using the FMLDIR routine

```
      PARAMETER     (LKEYFA=10)
      PARAMETER     (MAXDIR=1000)
      CHARACTER*255 DIRS(MAXDIR)
*
*     This should follow the path PASS5/GPMH/P5
*
      ICONT = 0
 1    CONTINUE
      CALL FMLDIR('//CERN/OPAL'//
     +            '/PROD/PASS>/GPMH/P<',
     +            DIRS,NFOUND,MAXDIR,IRC)
      IF(IRC.EQ.-1) THEN
         ICONT = 1
      ELSE
         ICONT = 0
         IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMLDIR'
      ENDIF
```

Return file names in directory structure

```
CALL FMLFIL (GENAM,FILES*,KEYS*,NFOUND,MAXFIL,JCONT,IRC*)
```

`GENAM` Character variable of maximum length 255 to specify the generic name of interest.

`FILES` Character array of length 255 characters and dimension MAXFIL to return the names of the files found.

`KEYS` Integer matrix of first dimension LKEYFA and second dimension MAXFIL.

`NFOUND` Integer variable to return the actual dimension of FILES and then actual second dimension of KEYS.

`MAXFIL` Integer constant to specify the second dimension of KEYS.

`JCONT` Integer variable to indicate whether this is a continuation of a previous call, or a new call. If more directories are found than can be returned in a single call, JCONT will be set to 1.

`IRC` Integer variable in which the return code is returned.

This routine returns the file names in the directory tree specified by GENAM. The directory tree may contain wild-cards (* or %) in any position, or numeric ranges specified as (mm:nn), such as (13:147). See on Page 81 for more details. The generic name GENAM may also contain the wild-cards < or >. These wild cards may be used to find, for example, the highest PASS of a run. Assuming that, at a given level, the generic name has a field containing PASSn, e.g. PASS3, PASS1, specifying PASS> will cause `FMLFIL` to follow **ONLY** the PASS3 subdirectory structure. NFOUND returns the actual number of files returned. IRC = -1 : more files found than can be returned in FILES(MAXFIL) `FMLDIR` should be called again with ICONT=0 to get the next MAXFIL batch of FILES. *N.B. This routine returns ALL file names in in the order in which they were added to the catalogue.* See the routine on Page 80 for details of how to sort the FILES array.

Example of using the `FMLFIL` routine

```
      PARAMETER     (LKEYFA=10)
      PARAMETER     (MAXFIL=1000)
      CHARACTER*255 FILES(MAXFIL)
*
*     This should follow the path PASS5/GPMH/P5
*
*     Indicate that this is a new call
*
      JCONT = 0
 1    CONTINUE
      CALL FMLFIL('//CERN/OPAL/PROD/PASS>/GPMH/P<',
     +            FILES,KEYS,NFOUND,MAXFIL,JCONT,IRC)
      IF(IRC.EQ.-1) THEN
         JCONT = 1
      ELSE
         JCONT = 0
         IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMLFIL'
      ENDIF
*
*     Process this batch of files...
*
*     ...
*
```

```
*     Get next batch of files...
*
      IF(JCONT.NE.0) GOTO 1
```

Sort file names and keys

CALL **FMSORT** (FILES,KEYS,NFILES,JSORT*,IRC*)

FILES	Character array of maximum length 255 containing the file names to be sorted.
KEYS	Integer matrix of size (10,NFILES) containing the keys vectors associated with the file names in FILES.
NFILES	Integer variable containing the size of the array FILES.
JSORT	Integer array of length NFILES to return the sorted indices into FILES and KEYS.
IRC	Integer variable in which the return code is returned.

This routine returns in JSORT the sorted indices into FILES and KEYS in ascending order of FILES.

Example of using the FMSORT routine

```
      PARAMETER     (LKEYFA=10, MAXFIL=1000)
      CHARACTER*255 FILES(MAXFIL)
      INTEGER       KEYS(LKEYFA,MAXFIL),JSORT(MAXFIL)
*
      CALL FMSORT(FILES,KEYS,NFILES,JSORT,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMSORT'
*
*     Print sorted files and associated keys
*
      DO 10 I=1,NFILES
      PRINT *,FILES(JSORT(I))
10    CALL FMPKEY(KEYS(1,JSORT(I))),LKEYFA)
```

Rank generic names by tape volume and file sequence number

CALL **FMRANK** CHFILES,KEYS,NFILES,JSORT,CHOPT,IRC*)

CHFILES	Character array of maximum length 255 containing the file names to be ranked.
KEYS	Integer matrix of size (10,NFILES) containing the keys vectors associated with the file names in CHFILES.
NFILES	Integer variable containing the size of the array CHFILES.
JSORT	Integer array of length NFILES to return the sorted indices into CHFILES and KEYS.
CHOPT	Character variable specifying the required options
IRC	Integer variable in which the return code is returned.

This routine returns a index into the arrays CHFILES and KEYS ranked by tape volume (MVIDFA) and file sequence number (MFSQFA). A maximum of 100 entries can be processed in one call.

It is foreseen that this routine be used prior to calling FMOPEN with the Q option.

Match file name against pattern

CALL **FMATCH** (CHFILE,MATCH,IRC*)

CHFILE Character variable of maximum length 255 containing the file name to match.

MATCH Character variable of maximum length 255 to specify the pattern to match against.

IRC Integer variable in which the return code is returned.

This routine matches the input file name CHFILE against the pattern specified in MATCH and returns 1 in IRC if no match is found. MATCH may include wild-cards as in VAX/VMS or VM/CMS file names, where a "%" will match any single character and a "*" will match any number of characters. In addition, numeric ranges may be specified using the syntax (mm:nn), such as (1:135). This form can be useful for specifying a set of runs, LEP files etc. Note that if *mm* is shorter than *nn*, it will be left-padded with zeroes, thus (1:135) is equivalent to (001:135). CHFILE and MATCH are character variables and may be as long as 255 characters each.

Example of using the routine FMATCH

```
      CHARACTER*255 CHFILE,MATCH
*     Match against files RUN1-9 (or RUNx)
      CALL FMATCH(CHFILE(1:LENOCC(CHFILE)),'RUN%',IRC)
      IF(IRC.EQ.0) THEN
         PRINT *,'Matching dataset found - ',CHFILE(1:LENOCC(CHFILE))
      ELSE
         PRINT *,'Dataset ',CHFILE(1:LENOCC(CHFILE)),' does not match'
      ENDIF
```

Match multiple names against pattern

CALL **FMMANY** (MATCH,FILES,NFILES,NMATCH*,IRC*)

MATCH Character variable of maximum length 255 to specify the pattern to match against.

FILES Character array of maximum length 255 containing a list of file names to match.

NFILES Integer variable specifying the number of files in FILES.

NMATCH Integer variable returning the element of FILES which matches.

IRC Integer variable in which the return code is returned.

This routine performs the same wild-card matching as `FMATCH` described above. In addition, if a < or > is specified in the MATCH pattern, it will return in NMATCH the element of the character array FILES which matches.

Example of using the routine FMMANY

```
      CHARACTER*255 FILES(9),MATCH
      FILES(1) = 'PASS3'
      FILES(2) = 'TEST3'
      FILES(3) = 'RUN13'
      FILES(4) = 'HOTP'
      FILES(5) = 'STUFF'
      FILES(6) = 'PASS3'
      FILES(7) = 'PASS5'
      FILES(8) = 'PAZS9'
      FILES(9) = 'POSS9'
      MATCH    = 'P%SS>'
      CALL FMMANY(MATCH,FILES,9,NMATCH,IRC)
      IF(NMATCH.NE.0) PRINT *,FILES(NMATCH)
```

Print contents of FATMEN keys vector

```
CALL FMPKEY (KEYS,NKEYS)
```

KEYS Integer array of length 10 containing a FATMEN keys vector

NKEYS Integer constant containing the number of keys to be printed. This should always be set to 10.

This routine displays the contents of the FATMEN keys vector in the correct format, depending on the data type of the various elements (integer or hollerith).

Example of using the FMPKEY routine

```
      CALL FMPKEY(KEYS,10)
```

Select files using the FATMEN keys

```
CALL FMSELK (GENAM,INKEYS,OUKEYS*,NKEYS*,MAXKEY,IRC*)
```

GENAM Character variable of maximum length 255 to specify the generic name.

INKEYS Integer array of length 10 to input the keys vector to match those found for the specified generic name.

OUKEYS Integer array of size (10,MAXKEY) to return the keys vector that match the input keys and the specified generic name.

MAXKEY Integer constant to specify the maximum second dimension of the array OUKEYS.

IRC Integer variable in which the return code is returned.

This routine selects all datasets corresponding to the generic name GENAM on the basis of the keys vector INKEYS. By setting the relevant elements of INKEYS, datasets may be selected on media type, copy level and location code. Thus, by setting INKEYS(MKMTFA) to 1 (disk), all matching datasets on disk can be selected. If a key element is set to a negative value, it will not be used in the selection. Thus, by setting INKEYS(MKLCFA) to -1 and INKEYS(MKMTFA) to 2, all datasets with the specified generic name on 3480 cassette at any location will be returned. The number of datasets that match is returned in NKEYS, up to a maximum of MAXKEY. The array OUKEYS may then be used as input to the routine FMGETK.

Example of using the FMSELK routine

```
*     Argument declarations
      PARAMETER (LKEYFA=10)
      PARAMETER (MAXKEY=999)
      DIMENSION INKEYS(LKEYFA),OUKEYS(LKEYFA,MAXKEY)
*     The following statements will select all datasets
*     with copy level (MKCLFA) of 1 (i.e. a copy of an original file),
*     media type of 1 (i.e. disk) and location code of 1 (i.e. CERN)
      INKEYS(MKCLFA) = 1
      INKEYS(MKMTFA) = 1
      INKEYS(MKLCFA) = 1
      CALL FMSELK('//CERN/CNDIV/CHRIS/TAPE8',INKEYS,OUKEYS,MAXKEY,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMSELK'
```

Select files using the FATMEN bank information

```
CALL FMSELB (GENAM,INKEYS,NKEYS,UEXIT,ISEL*,IRC*)
```

GENAM Character variable of maximum length 255 to specify the generic name.

INKEYS Integer array of length 10 to input the keys vector to match those found for the specified generic name.

NKEYS Integer constant to specify the actual second dimension of the array INKEYS.

ISEL Integer variable to return the index of the selected copy is returned. The keys vector of the selected copy is INKEYS(1,ISEL).

IRC Integer variable in which the return code is returned.

This routine allows the user to select a particular copy of a data set according to information contained in the FATMEN bank for the specified generic name. INKEYS is a matrix of dimension (10,NKEYS) containing the keys vectors for the various data sets matching the generic name, GENAM. This matrix is produced by a call to `FMSELK` to perform the appropriate initial selection, such as all data sets matching the input generic name that are on disk etc. For each matching data set, `FMSELB` will call the user-specified exit routine, which much be declared EXTERNAL, in the following way:

```
CALL UEXIT(GENAM,LBANK,KEYS,NKEYS,N,ISEL*,IRC*)
```

were

GENAM Character variable of maximum length 255 to specify the generic name.

LBANK Integer variable containing the address of the bank corresponding to the specified generic name.

KEYS Integer array of length 10 containing the keys vector associated with the specified generic name.

NKEYS Integer variable specifying the number of matches for this generic name, i.e. the number of times that UEXIT will be called.

N Integer variable specifying the candidate number for this call, which is less than or equal to NKEYS.

ISEL Integer variable to return the index of the selected copy is returned. The keys vector of the selected copy is INKEYS(1,ISEL). After the last call to UEXIT for each generic name, ISEL should be 0, to indicate that none of the candidates was selected, or the number of the candidate that was chosen.

IRC Integer variable in which the return code is returned.

Example of using the `FMSELB` routine

```
*     Argument declarations
      PARAMETER (LKEYFA=10)
      PARAMETER (MAXKEY=999)
      DIMENSION INKEYS(LKEYFA,MAXKEY)
      CALL FMSELB('//CERN/CNDIV/CHRIS/TAPE8',INKEYS,NKEYS,ISEL,IRC)
      IF(IRC.NE.0) THEN
         PRINT *,'Return code ',IRC,' from FMSELK'
      ELSE
         PRINT *,'Copy ',ISEL,' selected'
      ENDIF
```

Select files using keys matrix

```
CALL FMSELM (GENAM,LBANK*,KEYS*,KEYM,NKEY,CHOPT,IRC*)
```

GENAM	Character variable of maximum length 255 to specify the generic name.
LBANK	Integer variable in which the address of the FATMEN bank is returned.
KEY	Integer array of length 10 in which the keys vector associated with the bank at LBANK is returned.
KEYM	Integer matrix of size (10,NKEY) containing the ordered selection criteria.
NKEY	Integer variable to specify the second dimension of the matrix KEYM.
CHOPT	Character variable containing the user specified options.
IRC	Integer variable in which the return code is returned.

This routine takes each row of the input matrix KEYM in turn, and attempts to find a matching dataset. If no matching dataset is found, it then proceeds to the next row. Currently, checks are only made on media type (MKMTFA), location code (MKLCFA) and copy level (MKCLFA). A value of -1 indicates that no check on this item is to be made.

The following options are available:

```
I - issue FORTRAN inquire for disk datasets (with full support
    for SHIFT files, Unix environmental variables, VM mini-disks,
    SFS pools etc.)
N - check host name for disk datasets against current host
M - require that tape datasets reside on manually mounted volumes
R - require that tape datasets reside on robotically mounted volumes
```

Example of using the FMSELM routine

```
*     Argument declarations
      PARAMETER (LKEYFA=10)
      PARAMETER (NKEY=3)
      DIMENSION KEYS(LKEYFA),KEYM(LKEYFA,NKEY)
*
*     Define search criteria: first, disk dataset in location 31,
*        with no check on copy level,
*                             next, dataset on media type 2 in location 1,
*                             next, dataset on any media in any location
*                             with copy level 1
*
      KEYM(MKMTFA,1) = 1
      KEYM(MKCLFA,1) = -1
      KEYM(MKLCFA,1) = 31

      KEYM(MKMTFA,2) = 2
      KEYM(MKCLFA,2) = -1
      KEYM(MKLCFA,2) = 1

      KEYM(MKMTFA,3) = -1
      KEYM(MKCLFA,3) = 1
      KEYM(MKLCFA,3) = -1
*
*     Options NM: No check on host name for disk files,
*                 Manually mounted tape required
```

```
*
      CALL FMSELM('//CERN/OPAL/DDST/PASS3/FYZ1/P20/R02222C01',
     +   LBANK,KEYS,KEYM,NK,'NM',IRC)

      PRINT *,'Return code from FMSELM = ',IRC

      IF(IRC.EQ.0)
     +   CALL FMSHOW('//CERN/OPAL/DDST/PASS3/FYZ1/P20/R02222C01',
     +                 LBANK,KEYS,'A',IRC)
```

Compare FATMEN entries

`CALL` **FMCOMP** `(GENAM1,*LBANK1*,*KEYS1*,GENAM2,*LBANK2*,*KEYS2*,IRC*)`

GENAMn Character variable of maximum length 255 to specify the generic names of the files to be compared.

LBANKn Integer variable(s) to return the addresses of the banks corresponding to GENAM1 and GENAM2.

KEYSn Integer array of length 10 to return the keys vectors associated with the specified generic names.

IRC Integer variable in which the return code is returned.

This routine compares the FATMEN entries pointed to by the generic names GENAM1 and GENAM2. If either LBANK1 or LBANK2 is non-zero, the bank(s) at the corresponding address will be used. If either LBANK1 or LBANK2 are zero, the routine FMGETK will be used to obtain the corresponding bank. Should KEYS1(1) or KEYS2(1) be zero, the default FATMEN selection will apply. A zero return code indicates that the comparison succeeded. Other values of IRC are given below.

1 Error obtaining bank corresponding to GENAM1 using FMGETK.
2 Error obtaining bank corresponding to GENAM2 using FMGETK.
3 The datasets referred to by GENAM1 and GENAM2 are on different media.
4 Both datasets are on disk but the comparison failed (either fully qualified dataset name or host name differ).
5 Both datasets are on tape but the comparison failed (VSN/VID/FILESEQ do not match).

Example of using the FMCOMP routine

```
*     Compare entries with following generic names
      CALL FMCOMP('//CERN/OPAL/EVKI/MW34MH/R1200/P01',LBANK1,KEYS1,
     +            '//CERN/OPAL/EVKI/JT72MH/R1100/P19',LBANK2,KEYS2,IRC)
      IF(IRC.NE.0) PRINT *,'Comparison failed'
```

7.6 User exits

See also the description of user exits for the FMQVOL and FMALLO routines.

Print user words and comment

```
CALL FMUPRT (GENAM,LBANK,KEYS,IUSER,COMM,IRC*)
```

GENAM Character variable specifying the generic name of the file.
LBANK Integer variable containing the address of the corresponding bank.
KEYS Integer array of length 10 containing the keys vector associated with the specified generic name.
IUSER Integer array of length 10 containing the user words.
COMM Character variable of length 80 in which the comment is passed.
IRC Integer variable in which the return code is returned.

This routine is called from FMSHOW to print the user words and comment in the required format. A dummy routine is provided in the standard library. This routine should be provided if the user words should be interpreted in a special way, e.g. as packed 6 bit integers etc.

Example of a user FMUPRT routine

```
      SUBROUTINE FMUPRT(GENAM,LBANK,KEYS,JUSER,COMM,IRC)
      DIMENSION KEYS(10),JUSER(10),IUSER(10),USER(10)
      EQUIVALENCE (USER(1),IUSER(1))
      CHARACTER*(*) GENAM,COMM
      LCOMM = LENOCC(COMM)
      PRINT *,COMM(1:LCOMM)
      CALL UCOPY(JUSER,IUSER,10)
*     User words are real
      PRINT *,USER
      END
```

User selection

```
CALL FMUSEL (GENAM,LBANK,KEYS,IRC*)
```

GENAM Character variable specifying the generic name of the file.
LBANK Integer variable containing the address of the corresponding bank.
KEYS Integer array of length 10 containing the keys vector associated with the specified generic name.
IRC Integer variable in which the return code is returned.

This routine allows the user to accept or reject FATMEN entries during the default selection procedure. The FATMEN software builds a list of candidates that match the selection criteria that are in force. For each candidate the FMUSEL routine is called in turn. If a non-zero return code is found, the corresponding entry is removed from the list of candidates.

Example of a user FMUSEL routine

```
      SUBROUTINE FMUSEL(GENAM,LBANK,KEYS,IRC)
      DIMENSION KEYS(10)
      CHARACTER*(*) GENAM
*
*     Example user selection routine to reject any
*     entries with a copy level indicating Cray data representation
*
      IF(KEYS(MKCLFA).EQ.4) THEN
         IRC = -1
      ELSE
         IRC = 0
      ENDIF
      END
```

7.7 Routines to allocate media and interface to the TMS

Allocate new piece of media

CALL **FMALLO** (MEDIA,DENS,COMPACT,LIB,POOL,LBANK,CHOPT,VSN*,VID*,IRC*)

MEDIA	Character variable of length 4 to specify the medium required.
DENS	Character variable specifying the required density. **Reserved for future use**
COMPACT	Character variable specifying whether compaction is required or not. **Reserved for future use**
LIB	Character variable to specify the library from which the allocated volume should come. At CERN, a library consists of a two letter experimental code followed by the tape store name, such as PV_DPVAULT, PV_ARCHIVE.
POOL	Character variable to specify the pool from which the allocated volume should come. Pools can be used to separate libraries into groups, such as EA0001-EA1000 for DSTs, EA1001-EA9999 for Rawdata etc.
LBANK	Integer variable to input the address of the bank corresponding to the generic name for which the medium is to be allocated
CHOPT	Character variable to specify the type of operation required.
VSN	Character variable of length 6 in which the VSN is returned.
VID	Character variable of length 6 in which the VID is returned.
IRC	Integer variable in which the return code is returned.

This routine allocates a new piece of medium of the type specified. The allocation is performed by calling the Tape Management System (TMS).

Example of using the routine FMALLO

```
      CHARACTER*6 VSN,VID
*
*     Allocate a 3480 from the pool XVPROD in the library SMCF_1
*
      CALL FMALLO('3480','38K',' ','SMCF_1','XVPROD',LBANK,' ',
     +VSN,VID,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMALLO'
```

Get volume from name pool with sufficient free space

```
CALL FMGVOL (GENAM,LBANK,KEYS,CHLIB,CHPOOL,CHFREE,CHOPT,IRC*)
```

- `GENAM` Character variable of maximum length 255 to specify the generic name.
- `LBANK` Integer variable to return the address of the bank corresponding to the generic name GENAM. If LBANK is non-zero on input, the information in the bank at LBANK will be used.
- `KEYS` Integer array of length 10 to return the keys vector associated with the specified generic name. If LBANK is zero, KEYS may be used to select a specific copy of a file.
- `CHLIB` Character variable specifying the `TMS` library in which the specified pool(s) reside
- `CHPOOL` Character variable specifying the `TMS` pool that is to be searched for a volume with sufficient space. The space requirement is taken from the file size field `MFSZFA` of the FATMEN bank or catalogue entry.
- `CHFREE` Character variable specifying the `TMS` pool from which a new volume is allocated in the case the the pool `CHPOOL` contains no volumes or no volumes with sufficient space.
- `CHOPT` Character variable specifying the options required.
- `IRC` Integer variable in which the return code is returned.

This routine allocates a tape volume with sufficient space for the current file. The FATMEN bank is automatically updated with the volume information (FATMEN fields `MVSNFA, MVIDFA` and `MFSQFA` in the case of successful operation.

```
CALL FMGVID (IFREE,IMEDIA,CHLIB,CHPOOL,CHFREE,CHVSN,CHVID,IFILE,CHOPT,IRC*)
```

- `IFREE` Integer variable specifying the space requirement in Megabytes
- `IMEDIA` Integer variable specifying the media type. This is used to obtain the high water mark and maximum capacity.
- `CHLIB` Character variable specifying the `TMS` library in which the specified pool(s) reside
- `CHPOOL` Character variable specifying the `TMS` pool that is to be searched for a volume with sufficient space.
- `CHFREE` Character variable specifying the `TMS` pool from which a new volume is allocated in the case the the pool `CHPOOL` contains no volumes or no volumes with sufficient space.
- `CHVSN` Character variable in which the `VSN` of the allocated volume is returned.
- `CHVID` Character variable in which the `VID` of the allocated volume is returned.
- `IFILE` Integer variable in which the `file sequence number` on the allocated volume is returned.
- `CHOPT` Character variable specifying the options required.
- `IRC` Integer variable in which the return code is returned.

Manipulate `VOLINFO` tag field

```
CALL FMVINF (CHVID,MB,NFILES,CHOPT,IRC*)
```

- `CHVID` Character variable specifying the volume for which the `VOLINFO` tag is to be treated.
- `MB` Integer variable in which the number of megabytes stored on the current volume is input or returned depending on the value of the `CHOPT` argument.
- `NFILES` Integer variable in which the number of files on the current volume is input or returned depending on the value of the `CHOPT` argument.

CHOPT Character variable specifying the options required.

- S Set the VOLINFO tag to contain the values specified in the MB and NFILES arguments
- G Return in the MB and NFILES arguments the current information in the VOLINFO tag
- I Increment the information in the VOLINFO tag. The space used is incremented by the value contained in the MB argument, the number of files is incremented by one and the NFILES argument is ignored.

IRC Integer variable in which the return code is returned.

Move volumes between TMS pools

```
CALL FMPOOL (GENAM,LBANK,KEYS,CHPOOL,CHOPT,IRC*)
```

GENAM Character variable of maximum length 255 to specify the generic name.

LBANK Integer variable to return the address of the bank corresponding to the generic name GENAM. If LBANK is non-zero on input, the information in the bank at LBANK will be used.

KEYS Integer array of length 10 to return the keys vector associated with the specified generic name. If LBANK is zero, KEYS may be used to select a specific copy of a file.

CHOPT Character variable specifying the options required.

- L write lock
- U unlock (write enable)
- P privileged]transfer even if not owner
- D delete the TMS tag
- S set the TMS tag to the generic name
- B binary tag
- T text tag (D)

IRC Integer variable in which the return code is returned.

This routine allows you to transfer tape volumes between named pools in the Tape Management System (TMS). In addition, one can set or delete the TMS text or binary tags associated with the volume.

As an example of its use, one might wish to do the following:

- Allocate a tape volume from a named pool, e.g. XX_FREE
- Write a DST
- If successful, move the volume to XX_DSTS, write lock it, and set the TMS text tag to be the FATMEN generic name.

Conversely, when returning the volume to XX_FREE, one would call FMPOOL with options 'UDT' to unlock (write-enable) the volume and to delete the TMS text tag.

Example of using the routine FMPOOL

```
      CHARACTER*6   VSN,VID
      CHARACTER*255 GENAM
*
*     Create new bank ...
*
      GENAM = '//CERN/DELPHI/DSTS/RUN123'
      CALL FMBOOK(GENAM,KEYS,LBANK,LSUP,JBIAS,IRC)
```

```
*
*     Allocate a 3480 from the pool XX_FREE in the library SMCF_1
*
      CALL FMALLO('3480','38K',' ','SMCF_1','XX_FREE',LBANK,' ',
     +VSN,VID,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMALLO'
*
* ... write the data ...
*
* ...
*
*     Now move the volume to XX_DSTS, write lock and set the TMS text
*     tag to the generic name
*
      CALL FMPOOL(GENAM,LBANK,KEYS,'XX_DSTS','LST',IRC)
```

Obtain volume characteristics

CALL **FMQVOL** (GENAM,LBANK,KEYS,LIB*,MODEL*,DENS*,MNTTYP*,LABTYP*,IRC*)

GENAM Character variable of maximum length 255 to specify the generic name.

LBANK Integer variable to return the address of the bank corresponding to the generic name GENAM.

KEYS Integer array of length 10 to return the keys vector associated with the specified generic name.

LIB Character variable to return the name of the library in which the specified volume resides.

MODEL Character variable to return the generic device type (e.g. CART, TAPE, SMCF) associated with VID, or the physical device type (e.g. 3480).

MNTTYP Character variable to indicate whether VID will be robotically or manually mounted. MNTTYP returns 'R' or 'M' respectively.

LABTYP Character variable to return the label type of the volume VID, e.g. SL, NL, BLP.

IRC Integer variable in which the return code is returned.

This routine interfaces to the local Tape Management System and returns information on a given volume. Interfaces currently exist to the HEPVM TMS and VMTAPE.

If FATMEN has been installed without the TMS option, then default values will be returned. See the description of the FMEDIA routine for information on setting these default values. To allow this default information to be overridden on a volume by volume basis, FMQVOL calls a user exit FMUVOL which has exactly the same calling sequence. A dummy FMUVOL routine exists in PACKLIB.

Example of using the routine FMQVOL

```
      CHARACTER*6  VID
* Definitions from FATMEN sequence TMSDEF
      CHARACTER*6  DENS
      CHARACTER*8  LIB
      CHARACTER*4  LABTYP
      CHARACTER*1  MNTTYP
      CHARACTER*8  MODEL
      CHARACTER*7  ROBMAN(2)
      DATA         ROBMAN(1)/'-Robot '/,ROBMAN(2)/'-Manual'/
*
*     Obtain characteristics of volume corresponding to generic
```

```
*     name   GENAM
*
      CALL FMQVOL(GENAM,LBANK,KEYS,LIB,MODEL,DENS,MNTTYP,LABTYP,IRC)
      IF(IRC.EQ.100) PRINT *,'Volume unknown to TMS'
            IF(IC.EQ.0) THEN
              ITYPE = 1
              IF(MNTTYP.EQ.'M') ITYPE = 2
              PRINT *,'Library = ',LIB,' model = ',MODEL//ROBMAN(ITYPE)
     +                     ,' density = ',DENS,' label type = ',LABTYP
              ENDIF
```

Example of a user coded `FMUVOL` routine

```
      SUBROUTINE FMUVOL(GENAM,LBANK,KEYS,LIB,MODEL,DENS,MNTTYP,LABTYP,IRC)
      CHARACTER*255 GENAM
      PARAMETER     (LKEYFA=10)
      DIMENSION     KEYS(LKEYFA)
      CHARACTER*6 VID
+CDE,FATTYP.
+CDE,TMSDEF.
      CALL FMGETC(LBANK,VID,MVIDFA,6,IRC)
*
*     Return codes (HEPVM TMS convention)
*                  0   ok
*                  8   Syntax error
*                  12  Access denied
*                  100 Volume does not exist
*                  312 Volume unavailable
*
*     The following test is CERN specific!!!
*
      IF((VID(1:1).EQ.'I').AND.(ICNUM(VID,2,6).EQ.7)) THEN
         LIB = 'SMCF_1'
         MODEL = 'SMCF'
         MNTTYP= 'R'
      ENDIF

      END
```

Obtain media information

`CALL` **FMQMED** `(GENAM,*LBANK*,*KEYS*,IMEDIA*,IROBOT*,IRC*)`

`GENAM`	Character variable of maximum length 255 to specify the generic name.
`LBANK`	Integer variable to return the address of the bank corresponding to the generic name GENAM. If LBANK is non-zero on input, the information in the bank at LBANK will be used.
`KEYS`	Integer array of length 10 to return the keys vector associated with the specified generic name. If LBANK is zero, KEYS may be used to select a specific copy of a file.
`IMEDIA`	Integer variable to return the media type of the file corresponding to the generic name GENAM.
`IROBOT`	Integer variable to indicate, for tape files only, whether the volume corresponding to the specified generic name is mounted robotically or not.

- `0` The volume is not mounted robotically
- `1` The volume is mounted robotically

`IRC` Integer variable in which the return code is returned.

Example of using the routine `FMQMED`

```
      PARAMETER (LURCOR=200000)
      COMMON/CRZT/IXSTOR,IXDIV,IFENCE(2),LEV,LEVIN,BLVECT(LURCOR)
      DIMENSION    LQ(999),IQ(999),Q(999)
      EQUIVALENCE (IQ(1),Q(1),LQ(9)),(LQ(1),LEV)
*
      COMMON /USRLNK/LUSRK1,LUSRBK,LUSRLS
*
      COMMON /QUEST/IQUEST(100)
*
* Start of FATMEN sequence FATPARA
*
** ***     Data set bank mnemonics
*
*          Keys
      PARAMETER ( MKSRFA= 1, MKFNFA= 2, MKCLFA=7, MKMTFA=8
     1           ,MKLCFA= 9, MKNBFA=10, NKDSFA=10 )
*
** ***     Bank offsets
*
      PARAMETER ( MFQNFA=  1, MHSNFA= 65, MCPLFA= 67, MMTPFA= 68
     1           ,MLOCFA= 69, MHSTFA= 70, MHOSFA= 74
     2           ,MVSNFA= 77, MVIDFA= 79, MVIPFA= 81, MDENFA= 82
     3           ,MVSQFA= 83, MFSQFA= 84, MSRDFA= 85, MERDFA= 86
     4           ,MSBLFA= 87, MEBLFA= 88, MRFMFA= 89, MRLNFA= 90
     5           ,MBLNFA= 91, MFLFFA= 92, MFUTFA= 93, MCRTFA= 94
     6           ,MCTTFA= 95, MLATFA= 96, MCURFA= 97, MCIDFA= 99
     7           ,MCNIFA=101, MCJIFA=103, MFPRFA=105, MSYWFA=106
     8           ,MUSWFA=116, MUCMFA=126, NWDSFA=145
     9           ,MFSZFA=MSYWFA,MUSCFA=MSYWFA+1)

* End of FATMEN sequence FATPARA
      CHARACTER*6  DENS
      CHARACTER*8  LIB
      CHARACTER*4  LABTYP
      CHARACTER*1  MNTTYP
      CHARACTER*8  MODEL
      CHARACTER*7  ROBMAN(2)
      DATA         ROBMAN(1)/'-Robot '/,ROBMAN(2)/'-Manual'/
      PARAMETER (LKEYFA=10)
      PARAMETER (MAXFIL=3000)
      DIMENSION KEYS(LKEYFA,MAXFIL)
      CHARACTER*255 FILES(MAXFIL)
      CHARACTER*8   THRONG
      CHARACTER*255 TOPDIR
      CHARACTER*26  CHOPT
      CHARACTER*8   DSN
*
*
*     Initialise ZEBRA
*
```

```
      CALL MZEBRA(-3)
      CALL MZSTOR(IXSTOR,'/CRZT/','Q',IFENCE,LEV,BLVECT(1),BLVECT(1),
     +            BLVECT(5000),BLVECT(LURCOR))
      CALL MZLOGL(IXSTOR,-3)
*
* *** Define user division and link area like:
*
      CALL MZDIV  (IXSTOR, IXDIV, 'USERS', 50000, LURCOR, 'L')
      CALL MZLINK (IXSTOR, '/USRLNK/', LUSRK1, LUSRLS, LUSRK1)
*
*     Units for FATMEN RZ/FZ files
*
      LUNRZ = 1
      LUNFZ = 2
*
*     Initialise FATMEN
*
      CALL FMINIT(IXSTOR,LUNRZ,LUNFZ,'//CERN/delphi',IRC)
      CALL FMLOGL(1)
*
*     Get list of file names
*
      JCONT = 0
1     CONTINUE
      CALL FMLFIL('//CERN/DELPHI/P01_*/RAWD/NONE/Y90V00/E*/L*/*',
     +FILES,KEYS,NFOUND,MAXFIL,JCONT,IRC)
      IF(IRC.EQ.-1) THEN
        JCONT = 1
      ELSE
        JCONT = 0
      ENDIF

      PRINT *,NFOUND,' files found'

      DO 10 I=1,NFOUND
      LENF = LENOCC(FILES(I))
      PRINT *,'Processing ',FILES(I)(1:LENF)
      LBANK = 0
      CALL FMQMED(FILES(I)(1:LENF),LBANK,KEYS(1,I),IMEDIA,IROBOT,IRC)
*
*     Remove this entry if it corresponds to a tape in a (the) robot
*
      IF(IROBOT.NE.1) GOTO 10
      CALL FMSHOW(FILES(I)(1:LENF),LBANK,KEYS(1,I),'MG',IRC)
      GOTO 10
*
*     Write enable the freed volume
*
      CALL FMULOK(FILES(I)(1:LENF),LBANK,KEYS(1,I),' ',IRC)
      IF(IRC.NE.0) THEN
         PRINT *,'Return code ',IRC,' from FMULOK for ',
     +   FILES(I)(1:LENF)
         GOTO 10
      ENDIF
*
*     and return it to the pool XX_DSTS
*
```

```
      CALL FMPOOL(FILES(I)(1:LENF),LBANK,KEYS(1,I),
     +            'XX_RAWD',' ',IRC)
      IF(IRC.NE.0) THEN
         PRINT *,'Return code ',IRC,' from FMPOOL for ',
     +   FILES(I)(1:LENF)
         GOTO 10
      ENDIF
      CALL FMRM(FILES(I)(1:LENF),LBANK,KEYS(1,I),IRC)
      IF(IRC.NE.0) THEN
         PRINT *,'Return code ',IRC,' from FMRM for ',
     +   FILES(I)(1:LENF)
         GOTO 10
      ENDIF
10    CONTINUE
      IF(JCONT.NE.0) GOTO 1
*
*     Terminate cleanly
*
      CALL FMEND(IRC)

      END
```

7.8 Software write lock a volume

CALL **FMLOCK** (GENAM,*LBANK*,*KEYS*,CHOPT,IRC*)

GENAM Character variable of maximum length 255 to specify the generic name.

LBANK Integer variable to return the address of the bank corresponding to the generic name GENAM. If LBANK is non-zero on input, the information in the bank at LBANK will be used.

KEYS Integer array of length 10 to return the keys vector associated with the specified generic name. If LBANK is zero, KEYS may be used to select a specific copy of a file.

CHOPT

IRC

This routine software write locks a volume by issuing a `TMS LOCK DISABLE WRITE VID` command.

7.9 Software write enable a volume

CALL **FMULOK** (GENAM,*LBANK*,*KEYS*,CHOPT,IRC*)

GENAM Character variable of maximum length 255 to specify the generic name.

LBANK Integer variable to return the address of the bank corresponding to the generic name GENAM. If LBANK is non-zero on input, the information in the bank at LBANK will be used.

KEYS Integer array of length 10 to return the keys vector associated with the specified generic name. If LBANK is zero, KEYS may be used to select a specific copy of a file.

CHOPT

IRC

This routine software write enables a volume by issuing a `TMS LOCK ENABLE WRITE VID` command.

7.10 Issue SYSREQ command

```
CALL FMSREQ (SERVICE,COMMAND,IRC*,REPLY*,*REPLEN*)
```

`SERVICE` Character variable giving the name of the service for which the command is intended. At CERN, the only valid service is `TMS`.

`COMMAND` Character variable containing the command that is to be issued.

`IRC` Integer variable in which the return code is returned. The return code maybe from `SYSREQ` itself, or from the service.

- `0` Success
- `2` Reply buffer too short. REPLY(REPLEN) contains the command that should be issued to receive the next batch of replies.

`REPLY` Character array of length `REPLEN` in which the reply, if any, is returned.

`REPLEN` Integer variable containing the length of the arrary `REPLY`. On output, it contains the number of elements that contain useful data (which may be zero).

This routine issues a `SYSREQ` command. It differs from the routine `SYSREQ` described in the `CSPACK` [5] manual in that it performs retry in case of network problems.

7.11 Set default media information

```
CALL FMEDIA (MFMMED,MFMTYP,MFMGEN,MFMSIZ,MFMDEN,MFMMNT,MFMLAB,NMEDIA,IRC*)
```

`MFMMED` Integer array of length NMEDIA giving the FATMEN media type (integer code).

`MFMTYP` Character array of length NMEDIA specifying the physical device type.

`MFMGEN` Character array of length NMEDIA specifying the generic device type.

`MFMSIZ` Character array of length NMEDIA specifying the capacity in Megabytes of this medium.

`MFMDEN` Character array of length NMEDIA specifying the density of this medium.

`MFMMNT` Character array of length NMEDIA specifying the default mount type (M=manual, R=robotic) of this medium.

`MFMLAB` Character array of length NMEDIA specifying the default label type (SL=IBM labels, AL=ansi labels, NL=no labels)

`NMEDIA` Integer variable specifying the length of the preceeding arrays.

`IRC` Integer variable in which the return code is returned.

This routine can be used to set the characteristics of the various types of medium. It is automatically called at `FATMEN` initialisation time by the routine `FMINIT`. New media may be added at runtime using the routine `FMAMED`.

Example using the FMEDIA routine

```
*KEEP,FATMED. Default media attributes
      PARAMETER     (NMEDIA=4)
*
*     FATMEN media type
*
      DIMENSION     MFMMED(NMEDIA)
*
*     Generic device type
*
      CHARACTER*8   MFMGEN(NMEDIA)
*
*     Physical device type
*
      CHARACTER*8   MFMTYP(NMEDIA)
*
*     Default density
*
      CHARACTER*8   MFMDEN(NMEDIA)
*
*     Media size in Megabytes
*
      CHARACTER*8   MFMSIZ(NMEDIA)
*
*     Default mount type
*
      CHARACTER*1   MFMMNT(NMEDIA)
*
*     Default label type
*
      CHARACTER*2   MFMLAB(NMEDIA)

      DATA          MFMGEN(1)/'DISK'/,MFMGEN(2)/'3480'/,
     +              MFMGEN(3)/'3420'/,MFMGEN(4)/'8MM '/

      DATA          MFMTYP(1)/'DISK'/,MFMTYP(2)/'CT1 '/,
     +              MFMTYP(3)/'TAPE'/,MFMTYP(4)/'8200'/

      DATA          MFMDEN(2)/'38K '/,MFMDEN(3)/'6250'/,
     +              MFMDEN(4)/'43200'/

      DATA          MFMSIZ(1)/'0'/,   MFMSIZ(2)/'200'/,
     +              MFMSIZ(3)/'200'/, MFMSIZ(4)/'2300'/

      DATA          MFMMNT(1)/'M'/,   MFMMNT(2)/'M'/,
     +              MFMMNT(3)/'M'/,   MFMMNT(4)/'M'/

      DATA          MFMLAB(1)/'  '/,  MFMLAB(2)/'SL'/,
     +              MFMLAB(3)/'SL'/,  MFMLAB(4)/'SL'/
*KEND
*
*     Set default media attributes
*
      DO 55 I=1,NMEDIA
      MFMMED(I) = I
   55 CONTINUE
*
```

```
*     For each media type (1,2,3,...) set
*         physical device type (disk, 3480, 3420,...)
*         generic  device type (disk, ct1,  tape,...)
*         capacity (MB)        (?, 200, 150,...)
*         density              (?, 38K, 6250,...)
*         mount type           (manual/robotic)
*         label type           (SL/NL/AL)
*
+SELF,IF=APOL3,LEPICS.
*
*     Still using old generic device names...
*
      MFMGEN(2) = 'CART'
+SELF,IF=VMTAPE.
*
*     Generic names for VMTAPE...
*
      MFMGEN(2) = '18TR'
      MFMGEN(3) = '9TR '
+SELF.
      CALL FMEDIA(MFMMED,MFMTYP,MFMGEN,MFMSIZ,MFMDEN,
     +            MFMMNT,MFMLAB,NMEDIA,IRC)
```

7.12 Add additional media definitions

CALL **FMAMED** (MFMMED,MFMTYP,MFMGEN,MFMSIZ,MFMDEN,MFMMNT,MFMLAB,NMEDIA,IRC*)

- `MFMMED` Integer array of length NMEDIA giving the FATMEN media type (integer code).
- `MFMTYP` Character array of length NMEDIA specifying the physical device type.
- `MFMGEN` Character array of length NMEDIA specifying the generic device type.
- `MFMSIZ` Character array of length NMEDIA specifying the capacity in Megabytes of this medium.
- `MFMDEN` Character array of length NMEDIA specifying the density of this medium.
- `MFMMNT` Character array of length NMEDIA specifying the default mount type (M=manual, R=robotic) of this medium.
- `MFMLAB` Character array of length NMEDIA specifying the default label type (SL=IBM labels, AL=ansi labels, NL=no labels)
- `NMEDIA` Integer variable specifying the length of the preceeding arrays.
- `IRC` Integer variable in which the return code is returned.

This routine can be used to define additional media types.

Get, Set or Delete TMS Tags

CALL **FMTAGS** (GENAM,*LBANK*,*KEYS*,*CHTAG*,CHOPT,IRC*)

- `GENAM` Character variable of maximum length 255 to specify the generic name.
- `LBANK` Integer variable to return the address of the bank corresponding to the generic name GENAM. If LBANK is non-zero on input, the information in the bank at LBANK will be used.
- `KEYS` Integer array of length 10 to return the keys vector associated with the specified generic name. If LBANK is zero, KEYS may be used to select a specific copy of a file.

CHTAG Character variable of length 255 in which the tag associated with the specified generic name is returned (option G), or which contains the tag to be set (option S). This argument is ignored in the case of option D.

CHOPT Character variable to specify the type of operation required.

- D delete the tag
- G get and display the tag
- S set the tag
- B to select the BINARY tag (stored as CHARACTER*255)
- T to select the TEXT tag (default)
- V to select the VOLINFO tag

IRC Integer variable in which the return code is returned.

This routine allows the TMS text or binary tag fields associated with a tape volume that corresponds to a specified generic name to be deleted, set or obtained.

Example of using the FMTAGS routine

```
*
*     Set tag
*
      GENAM = '//CERN/CNDIV/JAMIE/OUT'
      LG    = LENOCC(GENAM)
      CHTAGS = 'Archive tape for FATMEN source'
      CALL FMTAGS(GENAM(1:LG),LBANK,KEYS,CHTAGS,'S',IRC)
*
*     Get and print tag
*
      CALL FMTAGS(GENAM(1:LG),LBANK,KEYS,CHTAGS,'G',IRC)
      PRINT *,IRC,CHTAGS(1:LENOCC(CHTAGS))
*
*     Delete binary tag
*
      CALL FMTAGS(GENAM(1:LG),LBANK,KEYS,CHTAGS,'DB',IRC)
```

7.13 Routines to tailor FATMEN selection

By default, FATMEN makes no check on location code or copy level when accessing a dataset or listing a catalogue entry. When attempting to access a dataset, FATMEN searches for a copy first on disk, then 3480 cartridge, and so on. The following routines may be used to modify the selection procedure.

If a call to FMSETL or FMSETC is made, only entries with a location code or copy level that match one of those specified will be accessible (unless the key serial number is given explicitly) or visible via a call to FMLS. Routine FMSETM permits not only the range of medium types to be set but also the order. For example, a call to FMSETM with the following values will limit the selection to medium types 3, 5, 1 and 2, in that order.

```
      NMTP = 4
      MTP(1) = 3
      MTP(2) = 5
      MTP(3) = 1
      MTP(4) = 2
```

Alternatively, routine FMSETK can be used. This allows the user to declare a keys matrix and character option which will be used in all future file selections. That is, it overrides both the FATMEN default selection and that set by the routines FMSETC, FMSETL and FMSETM.

The difference between these three approaches is explained below. With the default FATMEN selection, no check is made on copy level or location code. FATMEN takes in turn media types 1 to 4 and looks for an accessible dataset. The first one found is taken, with the proviso that robotically mounted tapes are given preference over manually mounted tapes.

The FMSETC, FMSETL and FMSETM routines allow the user to specify allowed values for the copy level and location code, and the order in which the different media types are to be processed. This allows media types that are not in the default selection to be processed, and the order to be changed. However, the user is unable to indicate that, for example, a location code or 31 is preferable to one of 21, within a given media type.

The FMSETK routine provides the user with more control. One might wish to set the order to be native mode copy on disk, native mode copy on 3480 robotically mounted cartridge, exchange mode copy on disk, exchange mode copy on 3480 cartridge.

Declare location codes to FATMEN

CALL **FMSETL** (LOC,NLOC,IRC*)

LOC	Array of length NLOC containing a list of location codes to be used in FATMEN selection of an entry.
NLOC	The number of location codes declared, maximum 99
IRC	Integer variable in which the return code is returned.

Load location code definitions from a file

CALL **FMLCOD** (LUNLOC,CHFILE,CHOPT,IRC*)

LUN	Integer variable or constant specifying the Fortran logical unit which should be used to read the file CHFILE
CHFILE	Character variable specifying the file containing the location code definitions
CHOPT	Character variable specifying the required options
IRC	Integer variable in which the return code is passed

This routine loads the location code definitions from the specified file. By default, FATMEN will load a file FATMEN.LOCCODES from the server directory at initialisation time.

Example of a location code definition file

```
      1=Cern Vault  :   For CERNVM Shift Cray VXCERN etc
      2=Cern Vault  :
      9=Cern Vault  :   Hidden or Obsolete data
     11=VXOPON      :   OPAL Online Vax cluster
     12=Online      :   OPAL (apollo) online facilities
     21=VXOPOF      :   OPAL Offline cluster
     31=SHIFT       :   SHIFT disk and archive storage
  16040=U-Vic       :   'obsolete' cartridges
  16041=U-Vic       :   Active data accessible at U-Vic.
```

```
49201=Bonn        :   Data accessible at Bonn
49202=Bonn        :   Data in Transit
49203=Bonn        :   Data in CERN Vault
49401=DESY        :   Data accessible at DESY/Hamburg
49402=DESY        :   Data in Transit
49403=DESY        :   Data in CERN Vault
49601=Heidelburg  :   Data accessible at Heidelburg
49602=Heidelburg  :   Data in Transit
49603=Heidelburg  :   Data in CERN Vault
49701=Freiburg    :   Data accessible at Freiburg
49702=Freiburg    :   Data in Transit
49703=Freiburg    :   Data in CERN Vault
16042=U-Vic       :   Data in transit
16043=U-Vic       :   Data at CERN in PC-VAULT
16049=U-Vic       :   Hidden or obsolete data
33101=Saclay      :   Active cartridges
33901=Saclay      :   'obsolete' cartridges
44501=UKACRL      :   Active data accessible at RAL
44502=UKACRL      :   Data in transit
44503=UKACRL      :   Data at CERN in PC-VAULT
44509=UKACRL      :   Hidden or obsolete data
44901=UKACRL      :   Inaccessible / archived data
97281=Weizmann    :   Active cartridges
97282=Weizmann    :   Data in transit
97283=Weizmann    :   Data at CERN in PC-VAULT
97289=Weizmann    :   Hidden or obsolete data
```

Obtain location code corresponding to a node

CALL **FMNTOL** (CHNODE,LOCCOD*,CHOPT,IRC*)

CHNODE Character variable specifying the name of the node for which the corresponding location code should be return

LOCCOD Integer variable in which the location code corresponding to CHNODE is returned.

CHOPT Character variable specifying the options required.

- (default) - perform case insensitive match
- C respect case

IRC Integer variable in which the return code is returned.

- 0 Success
- 1 File containing location codes and node names not found
- 2 File containing location codes could not be opened
- 3 No location code found for CHNODE

Obtain list of node names corresponding to a location

```
CALL FMLTON (LOCCOD,MAXNOD,CHNODE*,NNODES*,CHOPT,IRC*)
```

`LOCCOD` Integer variable containing the location code for which the corresponding node names should be returned.

`MAXNOD` Integer constant specifying the maximum number of node names that the calling procedure can accept

`CHNODE` Character array in which the node names corresponding to LOCCOD are returned.

`NNODES` Integer variable in which the number of nodes corresponding to LOCCOD are returned.

`NNODES` Integer variable in which the number of nodes corresponding to LOCCOD is returned. Only MIN(MAXNOD,NNODES) elements of CHNODE contain useful data.

`IRC` Integer return code

Declare media types to FATMEN

```
CALL FMSETM (MTP,NMTP,IRC*)
```

`MTP` Array of length NMTP containing a list of media types to be used in FATMEN selection of an entry.

`NMTP` The number of media types declared, maximum 99

`IRC` Integer variable in which the return code is returned.

Declare copy levels to FATMEN

```
CALL FMSETC (CPL,NCPL,IRC*)
```

`CPL` Array of length NCPL containing a list of copy levels to be used in FATMEN selection of an entry.

`NCPL` The number of copy levels declared, maximum 99

`IRC` Integer variable in which the return code is returned.

Declare selection matrix and options to FATMEN

```
CALL FMSETK (KEYM,NK,CHOPT,IRC*)
```

`KEYM` Matrix of size (LKEYFA,NK) containing an ordered list of key vectors to be used in FATMEN selection of an entry.

`NK` The number of copy levels declared, maximum 99

`CHOPT` The options to be used when selecting a dataset, as defined by the routine `FMSELM`.

`IRC` Integer variable in which the return code is returned.

See the description of the `FMSELM` routine for more information on how the key matrix is used in dataset selection.

7.14 Utility routines

Search in names file

```
CALL FMNICK (LUN,CHFILE,CHNICK,CHNAME,CHDESC,CHOPT,IRC*)
```

LUN	Integer variable or constant giving the Fortran unit on which the names file should be read.
CHFILE	Character variable specifying the name of the names file. If blank, a default names file will be used. The default names file for a FATMEN group XYZ is FMXYZ.NAMES in the directory pointed to by the shell variable FMXYZ.
CHNICK	Character variable specifying the value of the :nick tag to be searched for in the names file
CHNAME	Character variable in which the value of the :GNAME tag corresponding to the above nick name is returned.
CHDESC	Character variable in which the value of the :DESC tag corresponding to the above nick name is returned.
CHOPT	Character variable, currently ignored.
IRC	Integer variable in which the return code is returned.

This routine searches in the specified or default names file for the generic name and description that matches the specified nick name. If the nick name contains a slash, then the characters after the slash are treated as a range and appended to the value of the :GNAME tag. Only characters upto the character before the slash are used to find the names file entry.

A range may be given in the standard FATMEN form (mm:nn) or as mm-nn.

An example of using the FMNICK routine

```
      character*80 chname,chdesc

      call fmnick(1,'fmdelphi.names','RAWD91',chname,chdesc,' ',irc)

      print *,chname(1:lenocc(chname))
      print *,chdesc(1:lenocc(chdesc))

      call fmnick(1,'fmdelphi.names','LEPT92/(10:20)',
     +            chname,chdesc,' ',irc)

      print *,chname(1:lenocc(chname))
      print *,chdesc(1:lenocc(chdesc))

      call fmnick(1,'fmdelphi.names','RAWD91/17-77',
     +            chname,chdesc,' ',irc)

      print *,chname(1:lenocc(chname))
      print *,chdesc(1:lenocc(chdesc))

      end
```

The names file used in the above example

```
:NICK.RAWD92
:GNAME.P01_ALLD/RAWD/NONE/Y92V00/*/R:DESC.RAW data of ALL events; 1992 data
*---------------------------------------------------*
:NICK.RAWD91
:GNAME.P01_ALLD/RAWD/NONE/Y91V00/*/R
:DESC.RAW data of ALL events; 1991 data
*---------------------------------------------------*
:NICK.RAWD90
:GNAME.P01_ALLD/RAWD/NONE/Y90V00/*/R
:DESC.RAW data of ALL events; 1990 data
*---------------------------------------------------*
:NICK.RAWD
:GNAME.P01_ALLD/RAWD/NONE/Y*/*/R
:DESC.RAW data of ALL events.
*---------------------------------------------------*
:NICK.LEPT92
:GNAME.P01_ALLD/DSTO/LEPT/Y92V>/*/R
:DESC.DST data of the LEPTONIC events; 1992 data, last proc.
*---------------------------------------------------*
:NICK.LEPT91
:GNAME.P01_ALLD/MDST/LEPT/Y91V>/*/R
:DESC.RTD data of the LEPTONIC events; 1991 data, last proc.
*---------------------------------------------------*
:NICK.LEPT91_D
:GNAME.P01_ALLD/MDST/LEPT/Y91V05/*/R
:DESC.RTD data of the LEPTONIC events; 1991 data, 5th proc. (DELANA_D)
:NICK.LEPT90
*---------------------------------------------------*
:GNAME.P01_ALLD/MDST/LEPT/Y90V>/*/R
:DESC.RTD data of the LEPTONIC events; 1990 data, last proc .
*---------------------------------------------------*
:NICK.LEPT
:GNAME.P01_ALLD/MDST/LEPT/Y[90:99]V>/*/R
:DESC.RTD data of the LEPTONIC events; last available processings.
*---------------------------------------------------*
:NICK.DSTLEP92
:GNAME.P01_ALLD/DSTO/LEPT/Y92V>/*/R
:DESC.DST-only data of the LEPTONIC events; 1992 data, last proc .
*---------------------------------------------------*
:NICK.DSTLEP91
:GNAME.P01_ALLD/DSTO/LEPT/Y91V>/*/R
:DESC.DST-only data of the LEPTONIC events; 1991 data, last proc .
*---------------------------------------------------*
:NICK.DSTLEP90
:GNAME.P01_ALLD/DSTO/LEPT/Y90V>/*/R
:DESC.DST-only data of the LEPTONIC events; 1990 data, last proc .
*---------------------------------------------------*
:NICK.DSTLEP
:GNAME.P01_ALLD/DSTO/LEPT/Y[90:99]V>/*/R
:DESC.DST-only data of the LEPTONIC events; last available processings.
*---------------------------------------------------*
:NICK.S2PR92
:GNAME.P01_ALLD/MDST/S2PR/Y92V>/*/R
:DESC.RTD data of 2-PRONG events for calib./align.; 1992 data, last proc
*---------------------------------------------------*
:NICK.S2PR91
```

```
:GNAME.P01_ALLD/MDST/S2PR/Y91V>/*/R
:DESC.RTD data of 2-PRONG events for calib./align.; 1991 data, last proc
*---------------------------------------------------*
:NICK.S2PR
:GNAME.P01_ALLD/MDST/S2PR/Y[90:99]V>/*/R
:DESC.RTD data of 2-PRONG events selected for calibration/alignment.
*---------------------------------------------------*
:NICK.DSTO92
:GNAME.P01_ALLD/DSTO/PHYS/Y92V>/*/R
:DESC.DST-only of the "OR" of the physics teams, 1992 data, last proc.
*---------------------------------------------------*
:NICK.DSTO91
:GNAME.P01_ALLD/DSTO/PHYS/Y91V>/*/R
:DESC.DST-only of the "OR" of the physics teams, 1991 data, last proc.
*---------------------------------------------------*
:NICK.DSTO91_D
:GNAME.P01_ALLD/DSTO/PHYS/Y91V05/*/R
:DESC.DST-only, "OR" of the physics teams, 91 data, 5th proc.(DELANA_D)
*---------------------------------------------------*
:NICK.DSTO90
:GNAME.P01_ALLD/DSTO/PHYS/Y90V>/*/R
:DESC.DST-only of the "OR" of the physics teams; 1990 data, last proc.
*---------------------------------------------------*
:NICK.DSTO
:GNAME.P01_ALLD/DSTO/PHYS/Y[90:99]V>/*/R
:DESC.DST-only of the "OR" of the physics teams; all available procs.
*---------------------------------------------------*
:NICK.PHYS90
:GNAME.P01_ALLD/MDST/PHYS/Y90V>/*/R
:DESC.RTD data of the "OR" of the physics teams; 1990 data, last proc .
*---------------------------------------------------*
:NICK.CRAW
:GNAME.P01_ALLD/CRAW/PHYS/Y*V>/*/R
:DESC.Raw data of events tagged by Delana; last proc .
*---------------------------------------------------*
:NICK.CRAW91
:GNAME.P01_ALLD/CRAW/PHYS/Y91V>/*/R
:DESC.Raw data of events tagged by Delana; 1991 data, last proc .
*---------------------------------------------------*
:NICK.PHYS
:GNAME.P01_ALLD/MDST/PHYS/Y90V>/*/R
:DESC.RTD data of the "OR" of the physics teams (only exist for 1990)
*---------------------------------------------------*
:NICK.HOTP90
:GNAME.P01_ALLD/MDST/HOTP/Y90V>/*/R
:DESC.RTD data of the "HOT PHYSICS" events; 1990 data, last proc .
*---------------------------------------------------*
:NICK.HOTP
:GNAME.P01_ALLD/MDST/HOTP/Y90V>/*/R
:DESC.RTD data of the "HOT PHYSICS" events (only exist for 1990).
*---------------------------------------------------*
:NICK.CRAWHAD90
:GNAME.P01_ALLD/CRAW/HADR/Y90V>/*/R
:DESC.Raw data of events tagged by TEAM 4; 1990 data,last proc .
*---------------------------------------------------*
:NICK.CRAWHAD
:GNAME.P01_ALLD/CRAW/HADR/Y90V>/*/R
```

```
:DESC.Raw data of events tagged by TEAM 4 (only exist for 1990).
*-----------------------------------------------------*
:NICK.CRAWHADR90
:GNAME.P01_ALLD/CRAW/HADR/Y90V>/*/R
:DESC.Raw data of events tagged by TEAM 4; 1990 data, last proc .
*-----------------------------------------------------*
:NICK.CRAWHADR
:GNAME.P01_ALLD/CRAW/HADR/Y90V>/*/R
:DESC.Raw data of events tagged by TEAM 4 (only exist for 1990).
*-----------------------------------------------------*
:NICK.HADR90
:GNAME.P01_ALLD/CRAW/HADR/Y90V>/*/R
:DESC.Raw data of events tagged by TEAM 4; 1990 data, last proc .
*-----------------------------------------------------*
:NICK.HADR
:GNAME.P01_ALLD/CRAW/HADR/Y90V>/*/R
:DESC.Raw data of events tagged by TEAM 4 (only exist for 1990).
*-----------------------------------------------------*
:NICK.REST90
:GNAME.P01_ALLD/MDST/REST/Y90V>/*/R
:DESC.RD data of events NOT TAGGED by any team; 1990 data, last proc .
*-----------------------------------------------------*
:NICK.REST
:GNAME.P01_ALLD/MDST/REST/Y90V>/*/R
:DESC.RD data of events NOT TAGGED by any team (only exist for 1990).
*-----------------------------------------------------*
:NICK.PHDO90
:GNAME.P01_ALLD/DSTO/PHYS/Y90V>/*/R
:DESC.DST-only of the "OR" of the physics teams; 1990 data, last proc .
*-----------------------------------------------------*
:NICK.PHDO
:GNAME.P01_ALLD/DSTO/PHYS/Y90V>/*/R
:DESC.DST-only of the "OR" of the physics teams; 1990 data, last proc .
*-----------------------------------------------------*
:NICK.XDST
:GNAME.P01_ALLD/XDST/DETD/Y*V>/*/R
:DESC.DST + RAW data of RICH & HPC
*-----------------------------------------------------*
:NICK.XDST92
:GNAME.P01_ALLD/XDST/DETD/Y92V>/*/R
:DESC.DST + RAW data of RICH & HPC  (1992)
*-----------------------------------------------------*
:NICK.ALLD92
:GNAME.P01_ALLD/*/Y92V*/*/R
:DESC.All available Delphi 1992 real data.
*-----------------------------------------------------*
:NICK.ALLD91
:GNAME.P01_ALLD/*/Y91V*/*/R
:DESC.All available Delphi 1991 real data.
*-----------------------------------------------------*
:NICK.ALLD90
:GNAME.P01_ALLD/*/Y90V*/*/R
:DESC.All available Delphi 1990 real data.
*-----------------------------------------------------*
:NICK.ALLD
:GNAME.P01_ALLD/*/*/R
:DESC.All available Delphi real data.
```

```
*--------------------------------------------------*
:NICK.*
:GNAME.P01_ALLD/*/*/R
:DESC.All available Delphi real data.
*--------------------------------------------------*
*----------   S I M U L A T I O N  ----------------*
*--------------------------------------------------*
:NICK.BABA91
:GNAME.P01_SIMD/DSTO/BABA/*JUL91*/*
:DESC.Bhabha events in barrel region, July 91 release (SIM35 ANA41).
*--------------------------------------------------*
:NICK.BAFO91
:GNAME.P01_SIMD/DSTO/BAFO/*JUL91*/*
:DESC.Bhabha events in barrel+forward region, July 91 SIM35 ANA41.
*--------------------------------------------------*
:NICK.MUMU91
:GNAME.P01_SIMD/DSTO/MUMU/*JUL91*/*
:DESC.Dimuon events, July 91 release (SIM35 ANA41).
*--------------------------------------------------*
:NICK.TAU291
:GNAME.P01_SIMD/DSTO/TAU2/*JUL91*/*
:DESC.Tau tau events, July 91 release (SIM35 ANA41).
*--------------------------------------------------*
:NICK.QQME91
:GNAME.P01_SIMD/DSTO/QQME/*JUL91*/*
:DESC.qqbar events Matrix Element, July 91 release (SIM35 ANA41).
*--------------------------------------------------*
:NICK.QQPS91
:GNAME.P01_SIMD/DSTO/QQPS/*JUL91*/*
:DESC.qqbar events Parton Shower, July 91 release (SIM35 ANA41).
*--------------------------------------------------*
:NICK.QQPN91
:GNAME.P01_SIMD/DSTO/QQPN/*JUL91*/*
:DESC.qqbar events Parton Shower, no B0 mixing. July 91 release.
*--------------------------------------------------*
```

Output of the above program

```
P01_ALLD/RAWD/NONE/Y91V00/*/R
RAW data of ALL events; 1991 data
P01_ALLD/DSTO/LEPT/Y92V>/*/R(10:20)*
DST data of the LEPTONIC events; 1992 data, last proc.
P01_ALLD/RAWD/NONE/Y91V00/*/R(17:77)*
RAW data of ALL events; 1991 data
```

Modify user words

```
CALL FMMODU (PATH,UFORM,UVECT,UCOMM,CHOPT,IRC*)
```

PATH Character variable of maximum length 255 to specify the path name of containing the files for which the user words are to be modified. The path name may contain wild-cards, as for FMLDIR or FMSCAN.

PATH Character variable of length 4 specifying the user file format.

PATH Vector of length 10 containing the user words.

PATH Character variable of maximum length 80 specifying the user comment.

CHOPT Character variable to specify the type of operation required.

```
CHOPT = C: modify comment field
        F: modify user file format
        V: modify user vector
```

IRC Integer variable in which the return code is returned.

This routine may be used to modify any or all of the user fields associated with all files in the specified path. The path name may contain wild-cards.

Example of using the FMMODU routine

```
      DIMENSION  IVECT(10)
*
*     Fill IVECT
...
*
*     Now update the user words for all files in the subtree
*     corresponding to the highest pass
*
      CALL FMMODU('//CERN/OPAL/PROD/PASS>/*',' ',IVECT,' ','V',IRC)
```

Declare logical units to FATMEN

```
CALL FMSETU (LUN,NLUN,IRC*)
```

LUN Array of length NLUN containing a list of logical units that may be used by FATMEN. This list must not include those declared in the call to FMSTRT or FMINIT.

NLUN The number of logical units, maximum 99

IRC Integer variable in which the return code is returned.

Get a free logical unit

```
CALL FMGLUN (LUN*,IRC*)
```

LUN Integer variable in which an unused logical unit is returned. A list of logical units must have first been declared using FMSETU (see on Page 107). On VAX/VMS systems, if no units have been declared or if the list has been exhausted, the routine LIB$GET_LUN is called to obtain a new unit.

IRC Integer variable in which the return code is returned.

Get a free logical unit

```
CALL FMFLUN (LUN,IRC*)
```

LUN Integer variable containing the number of the FORTRAN logical unit to be freed.
IRC Integer variable in which the return code is returned.

Verify bank contents

```
CALL FMVERI (GENAM,LBANK,KEYS,CHOPT,IRC*)
```

GENAM Character variable of maximum length 255 to specify the generic name.
LBANK Integer variable to input the address of the bank corresponding to the generic name specified.
KEYS Integer array of length 10 to contain the keys vector associated with the specified generic name.
CHOPT Character variable specifying the required options.

	check entire bank
A	check entire bank, except option Q
C	check comment string
F	check file attributes
K	check keys
L	check logical attributes
M	check media attributes
N	check dataset name on disk/tape of this file
O	check owner, node and job of creator etc.
P	check physical attributes, such as record format etc.
Q	check that volume is known to TMS
S	check security details of this file (protection)
T	check date and time of creation, last access etc.

IRC Integer variable specifying the return code.

This routine will check the contents of a FATMEN bank and the associated keys vector. FMVERI returns warnings via the **IQUEST** vector and error conditions via the return code **IRC**. Errors are generated when a value for a required field is not specified or if an invalid value is given.

Example of using the FMVERI routine

```
      CALL FMVERI(GENAM,LBANK,KEYS,'T',IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMVERI
```

The following character options may be specified to cause FMVERI to check various parts of the FATMEN bank. See the table below for details of the formats of the various fields.

Errors and warnings returned via the IQUEST vector are given on the following page.

N.B. an error in the fields denoted by the options F,K,L,M,N,O,P or T will result in a non-zero return code. Banks containing such errors will not be added to the FATMEN catalogue.

IQUEST(3) **1** Comment is blank

	2 Comment contains 'unseen' characters
IQUEST(6)	**1** One or more file attributes missing
IQUEST(11)	**1** File name in keys does not match generic name
	2 Copy level, Media type or location code missing
	3 Copy level, Media type or location code conflict (keys/bank)
IQUEST(12)	**1** Invalid FATMEN format
IQUEST(13)	**1** Host or O/S (disk files) or VSN,VID or FSEQ (tape files) missing
	2 File size greater than maximum for specified media type
IQUEST(14)	**1** Dataset name is missing
	2 Dataset name is in invalid format
IQUEST(15)	**1** Owner, node or job name missing
IQUEST(16)	**1** Physical attributes missing or invalid
IQUEST(19)	**1** (Security field - no check at present)
IQUEST(20)	**1** Time or date fields are invalid
IQUEST(21)	**1** All user words are zero
IQUEST(26)	**IQFOUL** Invalid bank status word. Most likely caused by an assignment statement containing an undefined variable or by specifying an invalid bank address.

Pack date and time.

```
CALL FMPKTM (IDATE,ITIME,IPACK*,IRC*)
```

IDATE	Integer variable with date in YYMMDD format.
ITIME	Integer variable with time in HHMM format.
IPACK*	Integer variable to store the date and time in packed format.
IRC*	Integer variable in which the return code is returned.

This routine allows a date and time to be stored in a 4 byte integer word. The CERNLIB routine DATIME, entry Z007, can be used to obtain IDATE and ITIME in the correct format. Should IDATE, ITIME be in invalid format, a non-zero return code will be returned and IPACK will be set to zero.

Unpack date and time.

```
CALL FMUPTM (IDATE*,ITIME*,IPACK,IRC*)
```

IDATE*	Integer variable to store the date in YYMMDD format.
ITIME*	Integer variable to store the time in HHMM format.
IPACK	Integer variable with date and time in packed format.
IRC*	Integer variable in which the return code is returned.

This routine unpacks the date and time from a 4 byte integer word. IPACK must be the result of a previous call to `FMPKTM`. Should IPACK be in invalid format, a non-zero return code will be returned and IDATE and ITIME will be set to zero.

Pack date and time for VAX format.

```
CALL FMPKVX (CHDATE,IDATE*,ITIME*,IPACK*,IRC*)
```

- `CHDATE` Character variable of length 23 containing the date and time in VAX format (dd-mon-yyyy hh:mm:ss.ff, e.g. 11-JUL-1991 17:14:41.37)
- `IDATE*` Integer variable with date in YYMMDD format.
- `ITIME*` Integer variable with time in HHMM format.
- `IPACK*` Integer variable to store the date and time in packed format.
- `IRC*` Integer variable in which the return code is returned.

This routines provides the functionality of the `FMPKTM` except that the input date is expected in VAX format.

Unpack date and time for VAX format.

```
CALL FMUPVX (CHDATE*,IDATE*,ITIME*,IPACK,IRC*)
```

- `CHDATE*` Character variable of length 23 in which the date and time are returned in VAX format (dd-mon-yyyy hh:mm:ss.ff, e.g. 11-JUL-1991 17:14:41.37)
- `IDATE*` Integer variable to store the date in YYMMDD format.
- `ITIME*` Integer variable to store the time in HHMM format.
- `IPACK` Integer variable with date and time in packed format.
- `IRC*` Integer variable in which the return code is returned.

This routines provides the functionality of `FMUPTM` for VAX date and time formats.

7.15 Obsolete routines

The following routines remain for backward compatibility only. New code should use the suggested replacement, which is normally both more powerful and more efficient.

Return file names in specified directory

```
CALL FMFNMS (PATH,FILES*,KEYS*,NKEYS*,MAXKEY,IRC*)
```

- `PATH` Character variable of maximum length 255 to specify the path name of interest.
- `FILES` Character array of length 20 characters and dimension MAXKEY to return the names of the files found.
- `KEYS` Integer matrix of size (10,MAXKEY) to return the keys vectors associated with the file names in FILES.
- `NKEYS` Integer variable to return the actual dimension of KEYS.
- `MAXKEY` Integer constant to specify the second dimension of KEYS.
- `IRC` Integer variable in which the return code is returned.

This routine returns the file names and keys vectors in the directory specified by PATH. NKEYS returns the actual number of files and keys returned. If more than MAXKEY files are found, IRC will be non-zero. Otherwise IRC will be 0.

N.B. This routine returns ALL file names in the specified directory in the order in which they were added to the catalogue. See the routine on Page 80 for details of how to sort the FILES array.

The suggested replacement for this routine is `FMLFIL`, which provides wild-card support.

Example of using the `FMFNMS` routine

```
      PARAMETER     (LKEYFA=10)
      PARAMETER     (MAXKEY=100)
      CHARACTER*20  FILES(MAXKEY)
      INTEGER       KEYS(LKEYFA,MAXKEY)
      CALL FMFNMS('//CERN/DELPHI'//
     +            '/ALLD/RAWD/CERN/V001/E091.00/P01R000314/NONE',
     +            FILES,KEYS,NKEYS,MAXKEY,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMFNMS'
```

Using the `FMLFIL` routine as a replacement for `FMFNMS`

```
      PARAMETER     (LKEYFA=10)
      PARAMETER     (MAXKEY=100)
      CHARACTER*255  FILES(MAXKEY)
      INTEGER        KEYS(LKEYFA,MAXKEY)
      ICONT = 0
*     On return, IRC = -1 if more than MAXKEY files found.
*     Call FMLFIL again with ICONT ^=0 to retrieve the next batch of MAXKEY files.
      CALL FMLFIL('//CERN/DELPHI'//
     +            '/ALLD/RAWD/CERN/V001/E091.00/P01R000314/NONE/*',
     +            FILES,KEYS,NKEYS,MAXKEY,ICONT,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMLFIL'
```

Return file names in directory structure

`CALL` **FMLIST** `(PATH,FILES*,KEYS*,NFOUND,MAXFIL,IRC*)`

`PATH`	Character variable of maximum length 255 to specify the path name of interest.
`FILES`	Character array of length 255 characters and dimension `MAXFIL` to return the names of the files found.
`KEYS`	Integer matrix of size `(10,MAXFIL)` to return the keys vectors associated with the file names in `FILES`.
`NFOUND`	Integer variable to return the actual dimension of `FILES` and then actual second dimension of `KEYS`.
`MAXFIL`	Integer constant to specify the second dimension of `KEYS`.
`IRC`	Integer variable in which the return code is returned.

The suggested replacement for this routine is `FMLFIL`, which allows an arbitrary number of file names to be returned.

This routine returns the file names and keys vectors in the directory tree specified by PATH. The directory tree may contain wild-cards (* or %) in any position, or numeric ranges specified as (mm:nn), such as (13:147). See on Page 81 for more details. NFOUND returns the actual number of files and keys returned. If more than MAXFIL files are found, IRC will be non-zero. Otherwise IRC will be 0. *N.B. This routine returns ALL file names in in the order in which they were added to the catalogue.* See the routine on Page 80 for details of how to sort the FILES array.

Example of using the FMLIST routine

```
      PARAMETER     (LKEYFA=10)
      PARAMETER     (MAXFIL=1000)
      CHARACTER*255 FILES(MAXFIL)
      INTEGER       KEYS(LKEYFA,MAXFIL)
      CALL FMLIST('//CERN/DELPHI'//
     +            '/ALLD/RAWD/*/E091.*/P01R000%%%/NONE',
     +            FILES,KEYS,NFOUND,MAXFIL,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMLIST'
```

Using the FMLFIL routine as a replacement for FMLIST

```
      PARAMETER     (LKEYFA=10)
      PARAMETER     (MAXKEY=100)
      CHARACTER*255  FILES(MAXKEY)
      INTEGER        KEYS(LKEYFA,MAXKEY)
      ICONT = 0
*
*     On return, IRC = -1 if more than MAXKEY files found.
*     Call FMLFIL again with ICONT ^=0 to retrieve the next
*     batch of MAXKEY files.
*
      CALL FMLFIL('//CERN/DELPHI'//
     +            '/ALLD/RAWD/CERN/V001/E091.00/P01R000314/NONE/*',
     +            FILES,KEYS,NKEYS,MAXKEY,ICONT,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMLFIL'
```

Obtain names of subdirectories in specified tree

CALL **FMTREE** (PATH,SUBDIR*,NLEVEL,NFOUND*,MAXDIR,IRC*)

PATH	Character variable of maximum length 255 to specify the path to be searched.
SUBDIR	Character array of maximum length 255 and dimension MAXDIR to return the directory names found.
NLEVEL	Integer variable to set the number of levels below PATH to be searched.
NFOUND	Integer variable to return the number of names returned in the array SUBDIR.
MAXDIR	Integer constant to specify the maximum second dimension of the array SUBDIR.
IRC	Integer variable in which the return code is returned.

The suggested replacement for this routine is FMLDIR, which provides wild-card support and allows an arbitrary number of file names to be returned.

This routine returns in the character array SUBDIR the names of all directories below and including the input directory specified in PATH down to the level NLEVEL. MAXDIR specifies the maximum dimension of the array SUBDIR. NFOUND returns the number of directories found. Should NFOUND be less than or equal to MAXDIR, SUBDIR(NFOUND) will be the name of the current directory. If NFOUND > MAXDIR, IRC will be non-zero. PATH and SUBDIR are of type CHARACTER and may be as long as 255 characters.

Example of using the FMTREE routine

```
*
*     Argument declarations
      PARAMETER     (MAXDIR=1000)
      CHARACTER*255 PATH,SUBDIR(MAXDIR)
*     Get list of subdirectories down to a level of 10
      CALL FMTREE('//CERN',SUBDIR,10,NFOUND,MAXDIR,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMTREE'
```

Using FMLDIR as a replacement for FMTREE

```
*
      PARAMETER     (MAXDIR=100)
      CHARACTER*255  CHDIRS(MAXDIR)
      ICONT = 0
*
*     On return, IRC = -1 if more than MAXDIR directories found.
*     Call FMLDIR again with ICONT ^=0 to retrieve the next
*     batch of MAXDIR directories
*
      CALL FMLDIR('//CERN/DELPHI'//
     +            '/ALLD/RAWD/CERN/V001/E091.00/P01R000314/NONE/*',
     +            CHDIRS,NDIRS,MAXKEY,ICONT,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMLDIR'
```

User routine to allocate new piece of media

CALL **FUALLO** (MEDIA,VSN*,VID*,IRC*)

MEDIA	Character variable of length 4 to specify the medium required.
VSN	Character variable of length 6 in which the VSN is returned.
VID	Character variable of length 6 in which the VID is returned.
IRC	Integer variable in which the return code is returned.

The suggested replacement for this routine is FMUALL, which is automatically called from FMALLO if FATMEN has been installed without the TMS option.

This routine returns a free VSN and VID of type MEDIA.

Example of using the routine FUALLO

```
*     Argument declarations
      CHARACTER*6 VSN,VID
      CHARACTER*4 MEDIA
      MEDIA = '3480'
      CALL FUALLO(MEDIA,VSN,VID,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FUALLO
```

Create a new FATMEN bank

```
CALL FMLIFT (GENAM,KEYS*,MEDIA,CHOPT,IRC*)
```

- `GENAM` Character variable of maximum length 255 to specify the generic name.
- `KEYS` Integer array of length 10 to return the keys vector associated with the specified generic name.
- `MEDIA` Character variable of length 4 to specify the medium required.
- `CHOPT` Character variable to specify the options desired.
- `IRC` Integer variable in which the return code is returned.

The suggested replacement for this routine and `FMLINK` is `FMBOOK`.

This routine will create a new ZEBRA bank for the specified generic name GENAM and fill in default values. The user may then modify these before committing the changes via `FMPUT`. If IRC is non-zero, the IQUEST vector will contain the error condition signaled by MZBOOK. See the description of MZBOOK in the ZEBRA manual. After calling the `FMLIFT` routine, the user must obtain the address of the associated bank using `FMLINK` and then fill in the appropriate media details using `FMALLO` or `FUALLO`. See on Page 114, on Page 87 and on Page 113 for further details.

Example of using the `FMLIFT` routine

```
      CALL FMLIFT('//CERN/CNDIV/GOOSSENS/DCF/M123',KEYS,'DISK',' ',IRC)
```

`FMLIFT` returns warning conditions using the IQUEST vector. The following conditions may be reported:

`IQUEST(11)`0 if this generic name does not exist, 1 otherwise.
`IQUEST(12)` 0 if the corresponding directories already exist, 1 otherwise.

Get the address of a FATMEN bank

```
CALL FMLINK (GENAM,LBANK*,CHOPT,IRC*)
```

- `GENAM` Character variable of maximum length 255 to specify the generic name.
- `LBANK` Integer variable to input the address of the bank corresponding to the generic name specified.
- `CHOPT` Character variable to specify the options desired.
- `IRC` Integer variable in which the return code is returned.

This routine will return the address LBANK of the ZEBRA bank corresponding to the specified generic name GENAM. It should be called after `FMLIFT` if the user wishes to modify the information contained in the bank before committing the changes via `FMPUT`, or at the end of run when more details, such as file size, first and last event number, are known. If no bank exists for the specified generic name GENAM, warning messages will be printed and a bank address of zero returned. These messages may be suppressed by specifying the character option 'Q' (quiet), or globally by using `FMLOGL` described below. If no bank corresponding to the specified generic name GENAM is found, IRC will be non-zero.

Example of using the `FMLINK` routine

```
*     Obtain address of bank previously lifted by FMLIFT
      CALL FMLINK('CERN/GOOSSENS/DCF/M123',LBANK,'Q',IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from FMLINK
```

Obtain volume characteristics

CALL **FMQTMS** (VID,LIB*,MODEL*,DENS*,MNTTYP*,LABTYP*,IRC*)

VID	Character variable of length 6 specifying the visual identifier of the volume on which information is required.
LIB	Character variable to return the name of the library in which the specified volume resides.
MODEL	Character variable to return the generic device type (e.g. CART, TAPE, SMCF) associated with VID, or the physical device type (e.g. 3480).
MNTTYP	Character variable to indicate whether VID will be robotically or manually mounted. MNTTYP returns 'R' or 'M' respectively.
LABTYP	Character variable to return the label type of the volume VID, e.g. SL, NL, BLP.
IRC	Integer variable in which the return code is returned.

This routine interfaces to the local Tape Management System and returns information on a given volume. Interfaces currently exist to the HEPVM TMS and VMTAPE.

If FATMEN has been installed without the TMS option, then default values will be returned. See the description of the FMEDIA routine for information on setting these default values. To allow this default information to be overridden on a volume by volume basis, FMQTMS calls a user exit FMUTMS which has exactly the same calling sequence. A dummy FMUTMS routine exists in PACKLIB.

Example of using the routine FMQTMS

```
* Definitions from FATMEN sequence TMSDEF
      CHARACTER*6  DENS
      CHARACTER*8  LIB
      CHARACTER*4  LABTYP
      CHARACTER*1  MNTTYP
      CHARACTER*8  MODEL
      CHARACTER*7  ROBMAN(2)
      DATA         ROBMAN(1)/'-Robot '/,ROBMAN(2)/'-Manual'/
*
*     Obtain characteristics of volume I28901
*
      CALL FMQTMS('I28901',LIB,MODEL,DENS,MNTTYP,LABTYP,IRC)
      IF(IRC.EQ.100) PRINT *,'Volume unknown to TMS'
            IF(IC.EQ.0) THEN
              ITYPE = 1
              IF(MNTTYP.EQ.'M') ITYPE = 2
              PRINT *,'Library = ',LIB,' model = ',MODEL//ROBMAN(ITYPE)
     +               ,' density = ',DENS,' label type = ',LABTYP
              ENDIF
```

Example of a user coded FMUTMS routine

```
      SUBROUTINE FMUTMS(VID,LIB,MODEL,DENS,MNTTYP,LABTYP,IRC)
      CHARACTER*(*) VID
+CDE,FATTYP.
+CDE,TMSDEF.
*
*     Return codes (HEPVM TMS convention)
*                  0   ok
```

```
*                    8   Syntax error
*                    12  Access denied
*                    100 Volume does not exist
*                    312 Volume unavailable
*
*     The following test is CERN specific!!!
*
      IF((VID(1:1).EQ.'I').AND.(ICNUM(VID,2,6).EQ.7)) THEN
         LIB = 'SMCF_1'
         MODEL = 'SMCF'
         MNTTYP= 'R'
      ENDIF

      END
```

7.16 A sample FORTRAN program

A sample FORTRAN program is contained in the PATCH FATUSER on the FATMEN PAM. A CRADLE to produce this FORTRAN is currently stored in FATUSER CRADLE on FAT3's 191 disk on CERNVM.

An additional example is given below: this example loads information from DELPHI's Production Summary File (PSF) and adds it to the FATMEN file catalogue.

Other examples programs, including generation of a FORTRAN program from scratch, are to be found in the tutorial section of this manual.

Example of a PSF file

```
COMM ********************************************************************************
COMM ***                                   Institute and computer identifier
COMM ********************************************************************************
INST CERN CC           IBM3090
COMM ********************************************************************************
COMM ***                   List of Raw Data cass. from august 1989 Pilot Run
COMM ********************************************************************************
TDAS EP0001                                      X ALLD/RAWD/E091.0
       0/P01R000314/NONE/F001/CERN/V001/#000003/#000384*
UDAS No S-O-R.
TDAS EP0002                                          X ALLD/RAWD/E091.00
     /P01R000315/NONE/F001/CERN/V001/#000002/#000140
TDAS EP0003 C                                        X ALLD/RAWD/E091.00
     /P01R000315/NONE/F002/CERN/V001/#100002/#100203
UDAS 100000 added to event numbers because of duplicate run/event numbering in
              DAS system
TDAS EP0004 C01            X ALLD/RAWD/E091.00/P01R000315/NONE/F003/CERN/V001/#
200002/#203148
UDAS 200000 added to event numbers because of duplicate run/event numbering in
       DAS system
TDAS EP0005   C 01                                     X ALLD/RAWD/E091.00
    /P01R000316/NONE/F001/CERN/V001/#000002/#000193
TDAS EP0006   C 01                                     X ALLD/RAWD/E091.00
    /P01R000317/NONE/F001/CERN/V001/#000002/#000235
TDAS EP0007   C 01                                     X ALLD/RAWD/E091.00
/P01R000317/NONE/F002/CERN/V001/#000002/#000222
TDAS EP0008   C 01                                     X ALLD/RAWD/E091.00/
P01R000318/NONE/F001/CERN/V001/#000003/#000786
UDAS No S-O-R.
```

Using the FATMEN database from FORTRAN

```
      PROGRAM FATDEL
*----------------------------------------------------------------------*
*                                                                      *
* Example FATMEN program, which reads DELPHI Production Summary File   *
* and adds information to FATMEN database.                             *
* In this example, only TDAS records are processed.                    *
*                                                                      *
*----------------------------------------------------------------------*
*
*     Stuff for ZEBRA
*
      PARAMETER (LURCOR=200000)
      COMMON/CRZT/IXSTOR,IXDIV,IFENCE(2),LEV,LEVIN,BLVECT(LURCOR)
      DIMENSION    LQ(999),IQ(999),Q(999)
      EQUIVALENCE (IQ(1),Q(1),LQ(9)),(LQ(1),LEV)
      COMMON /USRLNK/LUSRK1,LUSRBK,LUSRLS
      COMMON /QUEST/IQUEST(100)
      PARAMETER      (LKEYFA=10)
      DIMENSION KEY(LKEYFA)
      CHARACTER*1 CHLUN
*
*     Initialise ZEBRA
*
      CALL MZEBRA(-3)
      CALL MZSTOR(IXSTOR,'/CRZT/','Q',IFENCE,LEV,BLVECT(1),BLVECT(1),
     +            BLVECT(5000),BLVECT(LURCOR))
      CALL MZLOGL(IXSTOR,-3)
*
* *** Define user division and link area like:
*
      CALL MZDIV  (IXSTOR, IXDIV, 'USERS', 50000, LURCOR, 'L')
      CALL MZLINK (IXSTOR, '/USRLNK/', LUSRK1, LUSRLS, LUSRK1)
*
*     Units for RZ database, FZ update files and PSF
*
      LUNRZ  = 1
      LUNFZ  = 2
      LUNPSF = 3
*
*     Issue FILEDEF for PSF
*
      WRITE(CHLUN,'(I1)') LUNPSF
      CALL VMCMS('FILEDEF '//CHLUN//
     +' DISK CERN PSF * (LRECL 132 RECFM F)',IRC)
*
*     Initialise FATMEN for DELPHI
*
      CALL FMINIT(IXSTOR,LUNRZ,LUNFZ,'//CERN/DELPHI',IRC)
*
*     Process information in PSF
*
      CALL ADDPSF(LUNPSF)
      END
      SUBROUTINE ADDPSF(LUNPSF)
* Start sequence FATPARA
*
```

```
*         Keys
      PARAMETER ( MKSRFA= 1, MKFNFA= 2, MKCLFA=7, MKMTFA=8
     1           ,MKLCFA= 9, MKNBFA=10, NKDSFA=10 )
*
** ***    Bank offsets
*
      PARAMETER ( MFQNFA=  1, MHSNFA= 65, MCPLFA= 67, MMTPFA= 68
     1           ,MLOCFA= 69, MHSTFA= 70, MHOSFA= 74
     2           ,MVSNFA= 77, MVIDFA= 79, MVIPFA= 81, MDENFA= 82
     3           ,MVSQFA= 83, MFSQFA= 84, MSRDFA= 85, MERDFA= 86
     4           ,MSBLFA= 87, MEBLFA= 88, MRFMFA= 89, MRLNFA= 90
     5           ,MBLNFA= 91, MFLFFA= 92, MFUTFA= 93, MCRTFA= 94
     6           ,MCTTFA= 95, MLATFA= 96, MCURFA= 97, MCIDFA= 99
     7           ,MCNIFA=101, MCJIFA=103, MFPRFA=105, MSYWFA=106
     8           ,MUSWFA=116, MUCMFA=126, NWDSFA=145
     9           ,MFSZFA=MSYWFA)

* End   sequence FATPARA
      PARAMETER (LURCOR=200000)
      COMMON/CRZT/IXSTOR,IXDIV,IFENCE(2),LEV,LEVIN,BLVECT(LURCOR)
      DIMENSION    LQ(999),IQ(999),Q(999)
      EQUIVALENCE (IQ(1),Q(1),LQ(9)),(LQ(1),LEV)
*
      COMMON /USRLNK/LUSRK1,LUSRBK,LUSRLS
*
      COMMON /QUEST/IQUEST(100)
      PARAMETER (LKEYFA=10)
      DIMENSION KEYS(LKEYFA)

      CHARACTER*132 CARD
      CHARACTER*240 GENEN
      CHARACTER     UFORM(4)*1,FFORM(4)*2
      DATA          UFORM/'N','X','A','G'/
      DATA          FFORM/'FZ','FX','FA','AS'/
      DATA NCARDS/0/,NPROC/0/

 1    CONTINUE
*
*     Process all records in file
*
      READ(LUNPSF,'(A132)',END=99) CARD
      NCARDS = NCARDS + 1
*
*     Got a PSF line, now process
*
      IF (CARD(1:4) .EQ. 'TDAS') THEN
         PRINT *,'Processing ',CARD
         NPROC = NPROC + 1
*
*     Format of TDAS card is:
*
*23456789_123456789_123456789_123456789_123456789_123456789_123456789_
*KEY VID----- M FS                                   F Generic name
*
*     where M  = media type (C=cart, T=tape)
*           FS = File sequence number
*           F  = file format
```

```
*
         CALL CFILL(' ',GENEN,1,240)
         GENEN = CARD(59:119)
         LGEN = LENOCC(GENEN)
*
*     Create bank for this generic name
*
         CALL FMLIFT('//CERN/DELPHI/'//GENEN(1:LGEN),
     +   KEYS,'3480','U',IRC)
*
*     Get address of the bank
*
         CALL FMLINK('//CERN/DELPHI/'//GENEN(1:LGEN),LFAT,' ',IRC)
*
*     Dataset name is always DELPHI
*
         CALL UCTOH('DELPHI',IQ(LFAT+MFQNFA),4,6)
*
*     Set values according to information found in PSF
*
         READ(CARD(17:18),'(I2)') IFILE
         IQ(LFAT+MFSQFA) = IFILE
         CALL UCTOH(CARD(6:13),IQ(LFAT+MVIDFA),4,6)
         CALL UCTOH(CARD(6:13),IQ(LFAT+MVSNFA),4,6)
         IMATCH = ICNTH(CARD(57:57),UFORM,4)
         CALL UCTOH(FFORM(IMATCH),IQ(LFAT+MFLFFA),4,2)
*
*     Write bank to RZ file (and ORACLE...)
*
         CALL FMPUT('//CERN/DELPHI/'//GENEN(1:LGEN),LFAT,IRC)
         ELSE
*        PRINT *,'Unrecognised card ',CARD(1:4)
         ENDIF

      GOTO 1
99    CONTINUE
      PRINT *,'EOF on PSF file found, LUN=',LUNPSF
      PRINT *,NCARDS,' records found, of which ',NPROC,' processed'
      END
```

Chapter 8: The FATMEN interactive interface

8.1 Summary of commands

The command line interface consists of a set of Unix-like commands and is based on the KUIP package[3]. The command line interface is activated by typing FM at the command level ($ prompt on VMS, VM READ on VM etc). For instance, to obtain a directory listing of `//cern/electra/rawdata/p1b90/run1` enter the command:

```
FM
FM> ls //cern/electra/rawdata/p1b90/run1
```

ALLOCATE `POOL LIBRARY GNAME DSN FFORM CPLEV HOSTN [COMM] [RECFM] [LRECL] [BLOCK] [FSIZE] [MEDIA]`

POOL	Name of the TMS Pool from which the volume is to be allocated
LIBRARY	Name of the TMS Library containing the specified pool
GNAME	Generic file name for which the volume is to be allocated
DSN	Dataset name on the tape volume
FFORM	File format (FZ, FA, FX, RZ, EP, UN, AS)
CPLEV	Copy level or data representation
HOSTN	Host name (e.g. CERNVM)
COMM	Comment string
RECFM	record format (e.g. FB, VBS)
LRECL	record length (in words)
BLOCK	block length (in words)
FSIZE	file size (in megabytes)
MEDIA	media type (2=3480,3=3420,4=8200,...)

Use the `ALLOCATE` comand to add a new tape file to the FATMEN catalogue. A tape is allocated from the specified pool. The file sequence number is always set to 1.

ADD/DISK `GNAME DSN FFORM CPLEV HOSTN [COMM] [RECFM] [LRECL] [BLOCK] [FSIZE] [LOCCOD] [USERWORDS]`

GNAME	Generic file name for which the volume is to be allocated
DSN	Dataset name on the tape volume
FFORM	File format (FZ, FA, FX, RZ, EP, UN, AS)
CPLEV	Copy level or data representation
HOSTN	Host name (e.g. CERNVM)
COMM	Comment string
RECFM	record format (e.g. FB, VBS)
LRECL	record length (in words)
BLOCK	block length (in words)
FSIZE	file size (in megabytes)
LOCCOD	Location code

`USER1`	User word 1
`USER2`	User word 2
`USER3`	User word 3
`USER4`	User word 4
`USER5`	User word 5
`USER6`	User word 6
`USER7`	User word 7
`USER8`	User word 8
`USER9`	User word 9
`USER10`	User word 10

Use the ADD/DISK command to add a new disk file to the FATMEN catalogue

ADD/TAPE `VSN VID FSEQ GNAME DSN FFORM CPLEV HOSTN [COMM] [RECFM] [LRECL] [BLOCK] [FSIZE] [MEDIA]`

`VSN`	volume serial number
`VID`	visual identifier
`FSEQ`	file sequence number
`GNAME`	Generic file name for which the volume is to be allocated
`DSN`	Dataset name on the tape volume
`FFORM`	File format (FZ, FA, FX, RZ, EP, UN, AS)
`CPLEV`	Copy level or data representation
`HOSTN`	Host name (e.g. CERNVM)
`COMM`	Comment string
`RECFM`	record format (e.g. FB, VBS)
`LRECL`	record length (in words)
`BLOCK`	block length (in words)
`FSIZE`	file size (in megabytes)
`MEDIA`	media type (2=3480,3=3420,4=8200,...)
`LOCCOD`	Location code
`USER1`	User word 1
`USER2`	User word 2
`USER3`	User word 3
`USER4`	User word 4
`USER5`	User word 5
`USER6`	User word 6
`USER7`	User word 7
`USER8`	User word 8
`USER9`	User word 9
`USER10`	User word 10

Use the ADD/TAPE command to add a new tape file to the FATMEN catalogue

CD [PATH] [CHOPT]

PATH Name of directory

CHOPT options

- A all of below
- Q show quota for new directory
- S show number of subdirectories
- T show creation and modification times
- U show usage information

Use the CD command to change the current default directory. Some examples of the CD command are given below:

```
[ zfatal]  > fm
 FATMEN.KUMAC not found
 Type INIT to initialise FATMEN> init cndiv

 FMINIT.  Initialisation of FATMEN package
 FATMEN   1.38  910218 15:00  CERN PROGRAM LIBRARY FATMEN=Q123
          This version created on      910218  at        1514
 Current Working Directory = //CERN/CNDIV
 FM> cd jamie/dsts
 Current Working Directory = //CERN/CNDIV/JAMIE/DSTS
 FM> cd ..
 Current Working Directory = //CERN/CNDIV/JAMIE
 FM> cd $HOME
 Current Working Directory = //CERN/CNDIV
 FM> cd jamie/dsts
 Current Working Directory = //CERN/CNDIV/JAMIE/DSTS
 FM> cd \
 Current Working Directory = //CERN/CNDIV/JAMIE
 FM>
```

CLR

Use the CLR command to clear the screen

CP FROM TO [KSN] [LOCCOD DATREP MEDTYP VSN VID FSEQ DSN HOST

FROM Original file

TO Target file

KSN Key serial number to identify a particular source file if there is more than one entry for the specified generic name

LOCCOD The location code for the output entry. If not specified, the location code of the input dataset will be used.

DATREP The data representation of the output entry. If not specified, the data representation of the input dataset will be used.

MEDTYP The media type of the output entry. If not specified, the media type of the input dataset will be used.

`VSN`	The VSN of the output entry.
`VID`	The VID of the output entry.
`FSEQ`	The file sequence number of the output entry.
`DSN`	The dataset name of the output entry.
`HOST`	The hostname of the output entry.

Use the CP command to copy a file. Note that this command makes a copy of the catalogue entry. It does not initiate a physical file copy.

COPY `GNAME [KS1] [KS2] [POOL] [LIBRARY] [VID] [FSEQ] [NODE] [FILE] [CHOPT]`

`GNAME` Generic name of the file to be copied

`KS1` Key serial number of the input file. If not specified, the normal FATMEN selection mechanism will be used.

`KS2` Key serial number of the output file. If specified the information on the output file will be taken from the corresponding catalogue entry.

`POOL` Name of the TMS pool from which a volume is to be allocated for the copy. If neither the POOL nor VID are specified, the output file is assumed to be on disk.

`LIBRARY` TMS Library containing the named pool from which a volume is to be allocated.

`VID` Visual Identifier of the output tape, if automatic allocation from a named pool is not to be used.

`FSEQ` File sequence number on the output tape volume.

`NODE` Node name on which the file resides. The default is the current node. If a remote node is specified, the file will be copied using the CSPACK [5] routines.

`FILE` File name for the output file

`TRANSPORT` Transport mechanism in case of a remote copy

- `TCPIP` This is the default transport for making remote copies
- `DECnet` Between VAX/VMS systems only
- `CHEOPS` Queue for transfer via Olympus satellite. This option is only valid on nodes at sites participating in the CHEOPS project. At other sites such requests will automatically be rejected. The source and destination sites must previously have been set using the `SET/SOURCE` and `SET/DESTINATION` commands.

`LOCCOD` The location code for the copied file

`DATREP` The data representation for the copied file

`MEDTYP` The medium type of the medium on which the copy resides

`CHOPT` Options

- `C` perform copy using STAGE CHANGE
- `L` Lock output tape volume using TMS LOCK command
- `P` Perform a physical copy. This uses the VMS copy command on VAX/VMS systems, the cp command on Unix systems, and VMIO subroutine calls on VM/CMS systems.
- `S` STAGE IN the input file

Use the COPY command to copy the data referenced by a generic name. A dataset may be copied by specifying existing catalogue entries (KS1 and KS2), a target VID, a TMS pool and library from which the output tape should be allocated, or a remote node and file name.

An example of a copy request where the copy should be performed using CHEOPS is shown below.

Requesting a copy via CHEOPS

```
set/source cern
set/dest helsinki
copy gname=//CERN/CNDIV/JAMIE/TEST node=HEL1 transport=CHEOPS vid=QQ1234
```

DIR [PATH] [OUTPUT] [CHOPT]

PATH	path-name
OUTPUT	output file name. If not specified the output will be directed to the terminal
CHOPT	options T list also subdirectory tree

Use the DIR command to issue a call to RZLDIR for the specified path. This command is normally used for debug purposes only.

DUMP GENAM [KSN] [BYTES] [BLOCKS] [FILES] [CODE]

GENAM	generic-name
KSN	Key serial number. If not specified, the default FATMEN selection will be used with the exception that disk entries will be ignored.
BYTES	Number of bytes per block to dump. The default is 320.
BLOCKS	Number of blocks per file to dump. The default is 1.
FILES	Number of files to dump. The default is 1.
CODE	Character code (EBCDIC or ASCII). If the volume is labelled, the character code used for the VOL1 label will be assumed. If unlabelled, this option can be used to override the default character code which is EBCDIC.

The DUMP command requests a TAPEDUMP of the VID corresponding to the specified generic name.

END

The END command closes the current FATMEN catalogue. After this command has been issued, the INIT command may be re-issued to look at a different FATMEN catalogue.

```
[zfatal] (543) fm
 FATMEN.KUMAC not found
 Type INIT to initialise FATMEN> init l3
 FMINIT.  Initialisation of FATMEN package
 FATMEN   1.51/07 910924 11:00 CERN PROGRAM LIBRARY FATMEN=Q123
          This version created on      910924  at        1114
 Current Working Directory = //CERN/L3
 FM> fc */*
 Total of  10926 matches ( 10926 files) in     43 directories
 FM> end
 FMEND. Terminating FATMEN package
 FM> init opal
 FMINIT.  Initialisation of FATMEN package
 FATMEN   1.51/07 910924 11:00 CERN PROGRAM LIBRARY FATMEN=Q123
          This version created on      910924  at        1114
 Current Working Directory = //CERN/OPAL
```

EXIT

Use the EXIT command to leave the FATMEN shell. Any outstanding catalogue updates will be sent to the server. See also the description of the `QUIT` command.

EXTRACT `GNAME OUTPUT [CHOPT]`

`GNAME`	Generic name describing the entries to be extracted. The name may contain wild-cards.
`OUTPUT`	Output filename in which the information is to be written.
`CHOPT`	Options

Use the `EXTRACT` command to extract all or part of a FATMEN catalogue. The select subset will be stored in the file specified in a format suitable for processing by the standard FATMEN catalogue server. (i.e. FZ exchange format, alpha mapping). Only entries that match the current key selection, as defined by the commands `SETLOCATION`, `SETMEDIATYPE` and `SETCOPYLEVEL` will be processed.

FC `[GNAME] [OUTPUT] [CHOPT]`

`GNAME`	Generic name describing the entries to be counted. The name may contain wild-cards. The default is *, i.e. to count all files in the current directory.
`OUTPUT`	Output filename. If not specified, the output will be directed to the terminal.
`CHOPT`	Options

- `D` display number of subdirectories at each level
- `F` display number of files at each level
- `L` display lowest level only, i.e. directories with no subdirectories
- `Z` display only directories with no (zero) files

Use the `FC` comand to count the number of files in a directory. `FC` will also count the number of files which match the specified pattern

Examples of using the FC command

```
FM> init delphi

 FMINIT.  Initialisation of FATMEN package
 FATMEN   1.64/07 920520 16:45 CERN PROGRAM LIBRARY FATMEN=Q123
          This version created on      920520  at        1645
 Current Working Directory = //CERN/DELPHI
FM> fc */*
 Total of  95141 matches ( 95141 files) in    394 directories
FM> ...
FM> cd $HOME
Current Working Directory = //FNAL/D0
FM> fc clmc/*/*
Total of     41 matches (    41 files) in     22 directories
FM> fc */*
Total of   2333 matches (  2333 files) in     41 directories
FM>
```

FIND `GNAME LOGNAM`

`GNAME`	Generic name of the file to be accessed
`LOGNAM`	Logical name (DDNAME, link etc.) to be associated with the file. A numeric value is converted according to the FORTRAN conventions on the local machine, unless the file is to be accessed using a special package.

`VM/CMS`	nn is converted to FTnnF001. If the logical format is EP, then it is converted to IOFILEnn.
`MVS`	nn is converted to FTnnF001. If the logical format is EP, then it is converted to IOFILEnn.
`VAX/VMS`	nn is converted to FOR0nn.
`Unix`	nn is converted to fort.nn

Use the `FIND` command to FIND the specified file and associate it with the specified logical unit. If required, the file is first staged to disk (See on Page 67 for the description of the format of the logical unit parameter).

GIME

Use the `GIME` command to reaccess the disk of the service machine which maintains the FATMEN catalogue. This command has no parameters.

INIT `GROUP`

`GROUP`	Group or Throng name

Use the `INIT` command to initialise the FATMEN system for the specified group or throng, e.g. `ALEPH`
If the file system name is not `//CERN`, this should also be given.

Examples of file system names

```
INIT //FNAL/D0

INIT //DESY/H1

INIT //CERN/DELPHI or INIT DELPHI
```

FATMEN automatically looks for and executes the files `FATSYS.KUMAC, FATGRP.KUMAC, FATUSER.KUMAC` and `FATLOGON.KUMAC`
The search will be made as follows:

`VAX/VMS`	look in directories defined by the logical name `FATPATH`. If this logical name is not defined, a default search list of `SYS$DISK:[],SYS$LOGIN` is used. (i.e. current and home directories)
`Unix`	Look in directories defined in path variable FATPATH. If this variable is not defined, use current and home directories.
`VM/CMS`	check disks in global variable `FATPATH`. If not defined, only the A disk is searched.
`MVS`	The user prefix is prepended to the kumac name, e.g. `R01JDS.FATUSER.KUMAC`.

The following examples show how to define the search path on different systems.

Defining the search path on VAX/VMS systems

```
$!
$! Look for macros in current directory, default directory and
$! public fatmen directory
$!
$  define fatpath sys$disk:[],sys$login,disk$fatmen:[public]
```

Defining a search path on Unix systems

```
export FATPATH=.:~:/fatmen/public
echo $FATPATH
.:/u/cp/jamie:/fatmen/public
```

Defining a search order on VM/CMS systems

```
setenv FATPATH a,d,g,q,p,s
```

For a transition period, the following scheme will also be supported for backword compatibility purposes.

If a file `FATMEN.KUMAC` (`fatmen.kumac` on Unix systems) is found in the current directory, it will be automatically executed. On VM/CMS systems, this file must reside on the disk accessed at mode A or G, or an extension of one of these disks. (e.g. in HEPVM batch, your 191 disk is accessed as a read-only extension of the 191 disk of the batch worker in which the job is executing, i.e. B/A)

LD `[PATH] [OUTPUT] [NLEVEL] [CHOPT]`

`PATH`	Directory name to be listed. If not specified, the directories below the current working directory are displayed. The directory name may contain wild cards.
`OUTPUT`	Output file name. If not specified, the output is directed to the terminal.
`NLEVEL`	Number of levels below the current directory to be displayed when option R is specified. The default is 1 level without option R and 100 levels with option R.
`CHOPT`	List of options

- `H` write header line in output file
- `R` list subdirectories recursively
- `V` 'very wide' listing. As W, but 132 columns
- `W` 'wide'. Subdirectories are displayed in multi-column (80) format

Use the `LD` command to display the contents of a directory The output of the command may be redirected to a file, e.g.

```
ld * output=subdirs.lis
```

Example of LD command

```
[zfatal] (796) fm
 FATMEN.KUMAC not found
 Type INIT to initialise FATMEN> init delphi

 FMINIT.  Initialisation of FATMEN package
 FATMEN   1.51/04 910918 17:00 CERN PROGRAM LIBRARY FATMEN=Q123

          This version created on      910918  at        1649
 Current Working Directory = //CERN/DELPHI
 FM> ld -w
 List of subdirectories...

 Directory: //CERN/DELPHI

 P01_ALLD P01_UNKN P01_TECH P01_COSM MEOLA
 Total of          5 subdirectories of which         5 match
 FM>
```

LOCK GENAM [KSN] [CHOPT]

GENAM generic name of the entry to be processed.
KSN Key serial number of the entry to be processed. If not specified, the default FATMEN selection is used with the exception that disk files are ignored.
CHOPT Option string

Use the LOCK command to disable WRITE access to the tape on which the file specified resides. If a negative key serial number is given, all entries matching the specified generic name will be locked.

LOGLEVEL [LEVEL]

LEVEL Loglevel to be set
- -3 Suppress all log messages
- -2 Error messages
- -1 Terse logging
- 0 Normal (FMINIT, FMEND etc.)
- 1 Log calls to FATMEN routines (FORTRAN callable interface)
- 2 Log to monitor FATMEN internal decisions, such as selection of a dataset
- 3 Debug messages

Use the LOGLEVEL command to set the FATMEN loglevel.

LN CHSRCE CHTRGT [CHCOMM] [IW1] [IW2] [IW3] [IW4] [IW5] [IW6] [IW7] [IW8] [IW9] [IW10] [CHOPT]

CHSRCE Character variable of maximum length 255 to specify the generic name of the link to an existing object.
CHTRGT Character variable of maximum length 255 specifying an existing generic name.
CHCOMM Character variable of maximum length 80 specifying the comment to be associated to the link.

IW1	User word 1
IW2	User word 2
IW3	User word 3
IW4	User word 4
IW5	User word 5
IW6	User word 6
IW7	User word 7
IW8	User word 8
IW9	User word 9
IW10	User word 10
CHOPT	Character variable specifying the required options. C set the comment field to the string specified in CHCOMM U set the user words to the values in the vector IVECT

Use the LN command to make a link to an existing catalogue entry. If the existing entry is itself a link, the link will point to the target of that link.

LS `[GNAME] [KSN] [OUTPUT] [NAMES] [CHOPT]`

GNAME	Generic name to be listed, containing wild-cards if required. If not specified, a * is used, i.e. all files in the current directory will be displayed.
KSN	Key serial number. If specified only the entry matching the generic name and key serial number combination will be displayed. This parameter is ignored if the specified generic name contains wild cards.
OUTPUT	Output file name. If not specified the output is directed to the terminal.
NAMES	Name of the names file (if the specified generic names starts with a %).
CHOPT	Option string

- A list all attributes, except DZSHOW (option Z).
- B brief (80) column listing
- C display comment field associated with file
- D to be used to generate a KUMAC file to remove (delete) the entry
- E extended (132) column listing
- F list file attributes, such as start/end record and block
- G list the full generic name of each file
- H write header line]useful in the case of output redirection
- I sort generic names in Increasing order
- J Just show entries that are accessible
- K list keys associated with this file (copy level, media type, location)
- L list logical attributes, such as FATMEN file format (ZEBRA exchange etc.)
- M list media attributes, such as VSN, VID, file sequence number for tape files, host type and operating system for disk files.
- N lists dataset name on disk/tape of this file
- O list owner, node and job of creator etc.
- P list physical attributes, such as record format etc.

- `Q` obtain volume information from Tape Management System (TMS) if the entry corresponds to a tape file, and if the TMS option is installed.
- `R` display where the data Reside.
  - `Disk files` Displays if the file is accessible and the access method, if known. (e.g. local disk, NFS, AFS, DFS etc.)
  - `Tape files` Displays if the associated volume is in an active library, i.e. not archived, whether it is staged, and whether a device of the appropriate type exists on the local node or is served.
- `S` lists security details of this file (protection)
- `T` list date and time of creation, last access etc.
- `U` list user words.
- `X` display only one entry per generic name (implies I)
- `W` list generic names across the page (default is one name per line)
- `Z` dump ZEBRA bank with DZSHOW.

Use the `LS` command to display the contents of a directory or display information on a given file within the current or specified directory.

If the generic name begins with a $ then it is assumed to be an environmental variable and is translated using the `CERNLIB` routine `GETENVF`.

If the generic name begins with a % then it is assumed to be a nick name defined in a names file. If no names file is specified then a default names file is used. This names file resides in the same directory or on the same minidisk as the current FATMEN catalogue and is named FMgroup.NAMES. e.g. for the group `L3` the name would be `fml3.names`.

Note that the filename specified may include wildcards, such as `ls t*`, to list all files starting with the letter `t` or `ls %%%` to list all files with three character filenames. The syntax (`mm:nn`) may be used in the filename to specify ranges, such as `P(3:45)`. See on page 81 for more details.

If the filename begins with a %, then the file name is first expanded using the `FMNICK` routine.

If the filename begins with a $, then it is treated as an environmental variable and translated using `GETENVF`.

Examples of using the LS command

```
FM> ls -w

Directory ://FNAL/CDF/FATMEN/8MM

R19570AB R19570AC R19573AB R19573AC R19573AD R19573AE R19573AF R19573AG
R19573AH R19573AI R19575AB R19575AC R19575AD R19575AE R19575AF R19575AG
R19575AH R19575AI R19576AB R19576AC R19576AD R19579AA R19579AB R19579AC
R19579AD R19579AE R19580AA R19580AB R19581AA R19581AC R19581AD R19581AE
R19581AF R19581AG R19581AH R19581AI R19581AK R19581AM R19581AO R19581AP
R19583AA R19583AB R19583AC R19587AB R19587AC R19599AA R19599AB R19599AC
R19599AD R19602AA R19602AB R19603AA R19604AA R19604AB R19604AC R19604AD
R19604AE R19604AF R19604AG R19604AH R19604AI R19604AJ R19604AK R19604AL
R19604AM R19604AN R19604AO R19604AP R19604AQ R19604AR R19604AU R19604AZ
R19614AD R19570AA
Total of     74 files in      1 directories
```

```
FM> ls r19(570:575)* -iw

Directory ://FNAL/CDF/FATMEN/8MM

R19570AA R19570AB R19570AC R19573AB R19573AC R19573AD R19573AE R19573AF
R19573AG R19573AH R19573AI R19575AB R19575AC R19575AD R19575AE R19575AF
R19575AG R19575AH R19575AI
Total of     19 files in      1 directories

FM> ls r19(570:575)ae -kmop

Directory: //FNAL/CDF/FATMEN/8MM


Generic filename: R19573AE
Copy level:  0 Media type: 4 Location code:      1 File serial number:      8
VSN: CC2010 VID: CC2010 FSEQ:    7
Created by:  CDF_FATM ACCT: E741CD_C on node: FNALE    by job: CDF_FATM
RECFM:  F    LRECL:  2048 BLKSIZE: 43200 Filesize:  2000 Use count:     0

Generic filename: R19575AE
Copy level:  0 Media type: 4 Location code:      1 File serial number:     16
VSN: CC2010 VID: CC2010 FSEQ:   15
Created by:  CDF_FATM ACCT: E741CD_C on node: FNALE    by job: CDF_FATM
RECFM:  F    LRECL:  2048 BLKSIZE: 43200 Filesize:  2000 Use count:     0

Files:    2
Total of      2 files in      1 directories
FM>
```

MAKE GNAME LOGNAM

GNAME Generic name of the file to be accessed

LOGNAM Logical name (DDNAME, link etc.) to be associated with the file. A numeric value is converted according to the FORTRAN conventions on the local machine, unless the file is to be accessed using a special package.

- VM/CMS nn is converted to FTnnF001. If the logical format is EP, then it is converted to IOFILEnn.
- MVS nn is converted to FTnnF001. If the logical format is EP, then it is converted to IOFILEnn.
- VAX/VMS nn is converted to FOR0nn.
- Unix nn is converted to fort.nn

The MAKE command is similar to the FIND command, except that it is for write access.

MEDIA `MEDIA [TYPE] [MODEL] [SIZE] [DENS] [MNTP] [LABEL]`

`MEDIA`	FATMEN media type. This is an integer as used in the FATMEN keys (MKMTFA) and bank (MMTPFA).
`TYPE`	device type. This is the physical device type, e.g. 3480, 3420 etc.
`MODEL`	generic device type. This may be the same as the physical device type. However, on many systems it differs. For example, at CERN the generic device type for `3480` is `CT1`. This is the resource that is required to access a volume of the physical type `TYPE`.
`SIZE`	capacity in MB
`DENS`	density
`MNTP`	mount-type (M/R)
`LABEL`	label-type (SL/AL/NL)

Use the `MEDIA` command to set or list attributes of a given type of media.

Example of using the `MEDIA` command

```
MEDIA 2 3480 CT1 200 38K M SL

would set the attributes of FATMEN media type 2 to

model:               3480
generic device type: CT1 (used on STAGE/SETUP requests)
capacity:            200 MB (maximum size on STAGE command)
density:             38K
mount type:          M
label  :             SL

MEDIA with no arguments lists the current settings

MEDIA 3 lists the settings for media type 3
```

MKDIR `PATH`

`PATH`	Path name of the directory to be created.

Use the `MKDIR` command to create a directory or directory tree.

MODIFY `GNAME [KSN] [LOCCOD] [DATREP] [MEDTYP] [FFORM] [RECFM] [RECL] [BLOCK] [FSIZE] [COMM]`

`FROM`	Original file
`TO`	Target file
`KSN`	Key serial number to identify a particular source file if there is more than one entry for the specified generic name
`LOCCOD`	The location code for the output entry.
`DATREP`	The data representation of the output entry.
`MEDTYP`	The media type of the output entry.

`FFORM`	File format (FZ, FA, FX, RZ, EP, UN, AS)
`RECFM`	record format (e.g. FB, VBS)
`LRECL`	record length (in words)
`BLOCK`	block length (in words)
`FSIZE`	file size (in megabytes)
`COMM`	Comment string

MV `FROM TO [KSN] [LOCCOD DATREP MEDTYP VSN VID FSEQ DSN HOST`

`FROM`	Original file
`TO`	Target file
`KSN`	Key serial number to identify a particular source file if there is more than one entry for the specified generic name
`LOCCOD`	The location code for the output entry. If not specified, the location code of the input dataset will be used.
`DATREP`	The data representation of the output entry. If not specified, the data representation of the input dataset will be used.
`MEDTYP`	The media type of the output entry. If not specified, the media type of the input dataset will be used.
`VSN`	The VSN of the output entry.
`VID`	The VID of the output entry.
`FSEQ`	The file sequence number of the output entry.
`DSN`	The dataset name of the output entry.
`HOST`	The hostname of the output entry.

Use the `MV` command to "move" or rename a file.

NICK `NICKNAME CHFILE CHOPT`

`NICKNAME`
`CHFILE`
`CHOPT`

Use the NICK command to display the generic name and description corresponding to the specified nickname. See the description of the `FMNICK` routine for further information.

PWD

Use the PWD command to print the current (working) directory.

QUIT

Use the QUIT command to leave the FATMEN shell. Any outstanding catalogue updates will be purged. See also the description of the `EXIT` command.

RM GNAME [KSN] [DSN] [HOST] [VID] [USER] [POOL] [PROT] [CHOPT]

GNAME File to be removed

KSN Key serial number. Must be specified if more than one entry exists for the given generic name and the A option is not given.

DSN Fileid/DSN. Only entries with the specified fileid will be deleted.

HOST Hostname. Only entries with the specified hostname will be deleted.

VID VID. Only entries with the specified VID will be deleted.

USER Username. Only entries with the specified username will be deleted.

POOL Pool into which the tape volume corresponding to the specified entry is returned to, if option F is given. If not specified, a pool name gg_FAT1 will be used, e.g. WS_FAT1.

PROT Protection group to be applied to the freed volume. The protection group will be set as follows:

- Unchanged
- G Set to the value specified by PROT or *None if no value is specified.
- GP Set to the value specified by PROT or the value of POOL if no value of PROT is specified.

CHOPT Options

- A remove all occurances of this generic name
- E erase disk file
- I prompt before removing each matching entry
- F free tape associated with specified entry
- G Set protection group
- P when used with option F, allows privileged TMS user to free anyones tapes (within a group)
- U 'unlock' or write-enable tape
- D delete TMS tag
- B binary TMS tag
- T text TMS tag

Use the RM command to remove a file from the catalogue. When multiple entries of the same generic name exist, the key serial number, as displayed by LS using the K option, should be specified. Alternatively, one can select which entry should be removed by specifying the DSN, HOST, VID and/or USER. Any of these fields may contain wild-cards.

Example of using the RM command

```
rm run123 -a vid=i* | Remove all entries for run123 for which
                    | the VID begins with an I.
```

Example of freeing a tape volume

```
rm //cern/l3/prod/data/pdreqq/cc00ws46 90 ! ! ! ! xvprod ! fugp
```

The following example shows how an entry may be removed from the catalogue, and the corresponding tape freed at the TMS level.

In this example, the tape volume associated with the specified catalogue entry is write enabled (option U), freed (option F). Option G specifies that the protection group for the volume is to be set. By default, the protection group has the same name as the TMS pool into which the volume is returned. Option P allows suitably privileged users to free any tapes within their group.

The command is repeated twice: the first form is currently required if issued from a KUMAC file as named parameters are not currently supported in macros. The second form may be used if the command is entered interactively.

Removing a catalogue entry and freeing a tape volume

```
RM //CERN/L3/PROD/DATA/PDREQQ/CC00WS46 90 ! ! ! ! XVPROD ! FUGP


RM //CERN/L3/PROD/DATA/PDREQQ/CC00WS46 ksn=90 pool=XVPROD -FUGP
```

RMDIR `PATH`

`PATH` Name of the directory to be removed. The directory must contain no files and no subdirectories.

Use the `RMDIR` command to remove an **empty** directory from the catalogue.

RMTREE `PATH`

`PATH` Name of the directory tree to be removed. The tree must contain no files.

Use the `RMTREE` command to remove a complete directory tree from the catalogue. If any files are found in the directory tree, the command will be refused.

RMLN `CHLINK [LUN] [CHFILE] [CHOPT]`

`CHLINK` Character variable specifying the path to be searched for dangling links, containing wild-cards as necessary

`LUN` Unit number on which the file `CHFILE` is written, with option `F`

`CHFILE` Character variable specifying the name of the file to be written, with option `F`.

`CHOPT` Character variable specifying the required options.

- `P` Print the names of dangling links
- `D` Write the names of dangling links in the form `rm generic-name ksn`
- `R` Remote dangling links
- `F` Redirect output to the file `CHFILE` on the unit `LUN`

SEARCH `GNAME [DSN] [HOST] [VID] [USER] [OUTPUT] [NMATCH] [CREATED] [CATALOGED] [ACCESSED] [UFORM] [COMMENT] [OUTPUT] [CHOPT]`

`GNAME`	Generic name specifying the files to be searched
`DSN`	String containing wild cards, if required. If not specified, no match is made on this field.
`HOST`	String containing wild cards, if required. If not specified, no match is made on this field.
`VID`	String containing wild cards, if required. If not specified, no match is made on this field.
`USER`	String containing wild cards, if required. If not specified, no match is made on this field.
`NMATCH`	Instruct search to stop after `NMATCH` matches. If not specified, the entire catalogue will be searched.
`CREATED`	date range
`CATALOGED`	date range
`ACCESSED`	date range
`UFORM`	String containing wild cards, if required. If not specified, no match is made on this field.
`COMMENT`	String containing wild cards, if required. If not specified, no match is made on this field.
`OUTPUT`	Output filename. If not specified the output will be sent to the terminal.
`CHOPT`	Options

- `A` list all attributes, except DZSHOW (option Z).
- `B` brief (80) column listing
- `C` display comment field associated with file
- `D` to be used to generate a KUMAC file to remove (delete) the entry
- `E` extended (132) column listing
- `F` list file attributes, such as start/end record and block
- `G` list the full generic name of each file
- `H` write header line]useful in the case of output redirection
- `I` sort generic names in Increasing order
- `J` Just show entries that are accessible
- `K` list keys associated with this file (copy level, media type, location)
- `L` list logical attributes, such as FATMEN file format (ZEBRA exchange etc.)
- `M` list media attributes, such as VSN, VID, file sequence number for tape files, host type and operating system for disk files.
- `N` lists dataset name on disk/tape of this file
- `O` list owner, node and job of creator etc.
- `P` list physical attributes, such as record format etc.
- `Q` obtain volume information from Tape Management System (TMS) if the entry corresponds to a tape file, and if the TMS option is installed.
- `R` display where the data Reside.
  - `Disk files` Displays if the file is accessible and the access method, if known. (e.g. local disk, NFS, AFS, DFS etc.)
  - `Tape files` Displays if the associated volume is in an active library, i.e. not archived, whether it is staged, and whether a device of the appropriate type exists on the local node or is served.
- `S` lists security details of this file (protection)
- `T` list date and time of creation, last access etc.

- `U` list user words.
- `X` display only one entry per generic name (implies I)
- `W` list generic names across the page (default is one name per line)
- `Z` dump ZEBRA bank with DZSHOW.

Use the search command to print the generic names of files which match the specified criteria. Character fields may include the `*` or `%` wild cards. e.g. `SEARCH * VID=I* #` search current working directory for entries # with VID's beginning with I.

Options are passed to `FMSHOW`.

If `NMATCH` is non-zero, `SEARCH` will stop after `NMATCH` matches have been found.

Set the loglevel to `<0` to stop the printing of the names of files and directories searched.

Date and time ranges may be given, as in the following example: which searches for files in the current directory that have been accessed between 31st January, 1991 and midday on 30th June 1991

To make selections on the user words, use the `SETUSERWORDS` command. This allows you to set values or ranges for each of the 10 user words. Once these values have been set, `SEARCH` will only display entries where the userwords match these values or ranges. Currently, only 0 or positive integers are accepted as values.

```
SEARCH * accessed=910131-910630.1200
```

The format of the date and time fields is `from_date_and_time-to_date_and_time`.

Dates and times are specified as `YYMMDD.HHMM`.

Examples of valid date and time ranges

```
910125.1230-910227.1630   # Explicit range

910125-910227             # Time fields default to 0000

910125-                   # To date and time default to NOW

-910125                   # From date and time default to 000000.0000


ACCESSED=-910125          # gives files accessed prior to 910125

ACCESSED=910125           # gives files accessed on or after 910125

ACCESSED=910125-910127    # gives files accessed on 910125 or 910126.
```

An example of a search command is given below. This example looks file files KS09, KS10, KS11 in all subdirectories matching `88%/MIN*` for files where the host name is CERNVM, the VID starts with KR and the username is HAG. The output is sent to the screen and the LS options M (media details), G (generic name) and O (owner details) are used.

Example of SEARCH command

```
FM > sea 88%/min*/ks(09:11) host=cernvm vid=kr* user=hag -mgo output=tty
Searching directory //CERN/NA31/883/MIN8

//CERN/NA31/883/MIN8/KS09
VSN: KR5822 VID: KR5822 FSEQ:    1
Created by:  HAG      ACCT: HAG$VH   on node: CERNVM   by job: HAG

//CERN/NA31/883/MIN8/KS10
VSN: KR5823 VID: KR5823 FSEQ:    1
Created by:  HAG      ACCT: HAG$VH   on node: CERNVM   by job: HAG

//CERN/NA31/883/MIN8/KS11
VSN: KR5832 VID: KR5832 FSEQ:    1
Created by:  HAG      ACCT: HAG$VH   on node: CERNVM   by job: HAG
Searching directory //CERN/NA31/885/MIN8
Searching directory //CERN/NA31/886/MIN8

//CERN/NA31/886/MIN8/KS09
VSN: KR5847 VID: KR5847 FSEQ:    1
Created by:  HAG      ACCT: HAG$VH   on node: CERNVM   by job: HAG

//CERN/NA31/886/MIN8/KS10
VSN: KR5849 VID: KR5849 FSEQ:    1
Created by:  HAG      ACCT: HAG$VH   on node: CERNVM   by job: HAG

//CERN/NA31/886/MIN8/KS11
VSN: KR5851 VID: KR5851 FSEQ:    1
Created by:  HAG      ACCT: HAG$VH   on node: CERNVM   by job: HAG

Matches:    6
FM>
```

SCAN GNAME [NLEVELS] [DSN] [HOST] [VID] [USER] [OUTPUT] [NMATCH] [CREATED] [CATALOGED] [ACCESSED] [UFORM] [COMMENT] [OUTPUT] [STRING] [CHOPT]

GNAME	Generic name specifying the files to be searched
NLEVELS	Number of levels down to be scanned
DSN	String containing wild cards, if required. If not specified, no match is made on this field.
HOST	String containing wild cards, if required. If not specified, no match is made on this field.
VID	String containing wild cards, if required. If not specified, no match is made on this field.
USER	String containing wild cards, if required. If not specified, no match is made on this field.
NMATCH	Instruct search to stop after NMATCH matches. If not specified, the entire catalogue will be searched.
CREATED	date range
CATALOGED	date range
ACCESSED	date range
UFORM	String containing wild cards, if required. If not specified, no match is made on this field.
COMMENT	String containing wild cards, if required. If not specified, no match is made on this field.

`OUTPUT` Output filename. If not specified the output will be sent to the terminal.

`STRING` String to be appended to the `RM` command, in case of option D.

`CHOPT` Options

- `A` list all attributes, except DZSHOW (option Z).
- `B` brief (80) column listing
- `C` display comment field associated with file
- `D` to be used to generate a KUMAC file to remove (delete) the entry In this case files within a directory will be scanned backwards.
- `E` extended (132) column listing
- `F` list file attributes, such as start/end record and block
- `G` list the full generic name of each file
- `H` write header line]useful in the case of output redirection
- `I` sort generic names in Increasing order
- `J` Just show entries that are accessible
- `K` list keys associated with this file (copy level, media type, location)
- `L` list logical attributes, such as FATMEN file format (ZEBRA exchange etc.)
- `M` list media attributes, such as VSN, VID, file sequence number for tape files, host type and operating system for disk files.
- `N` lists dataset name on disk/tape of this file
- `O` list owner, node and job of creator etc.
- `P` list physical attributes, such as record format etc.
- `Q` obtain volume information from Tape Management System (TMS) if the entry corresponds to a tape file, and if the TMS option is installed.
- `R` display where the data Reside.
  - `Disk files` Displays if the file is accessible and the access method, if known. (e.g. local disk, NFS, AFS, DFS etc.)
  - `Tape files` Displays if the associated volume is in an active library, i.e. not archived, whether it is staged, and whether a device of the appropriate type exists on the local node or is served.
- `S` lists security details of this file (protection)
- `T` list date and time of creation, last access etc.
- `U` list user words.
- `X` display only one entry per generic name (implies I)
- `W` list generic names across the page (default is one name per line)
- `Z` dump ZEBRA bank with DZSHOW.

Use the scan command to print the generic names of files which match the specified criteria. Character fields may include the `*` or `%` wild cards. e.g. `SEARCH * VID=I* #` search current working directory for entries `#` with VID's beginning with I.

Options are passed to `FMSHOW`.

If `NMATCH` is non-zero, `SCAN` will stop after `NMATCH` matches have been found.

Set the loglevel to `<0` to stop the printing of the names of files and directories searched.

Date and time ranges may be given, as in the following example: which searches for files in the current directory that have been accessed between 31st January, 1991 and midday on 30th June 1991

To make selections on the user words, use the `SETUSERWORDS` command. This allows you to set values or ranges for each of the 10 user words. Once these values have been set, `SEARCH` will only display entries where the userwords match these values or ranges. Currently, only 0 or positive integers are accepted as values.

```
SCAN * accessed=910131-910630.1200
```

The format of the date and time fields is `from_date_and_time-to_date_and_time`.
Dates and times are specified as `YYMMDD.HHMM`.

Examples of valid date and time ranges

```
910125.1230-910227.1630   # Explicit range

910125-910227             # Time fields default to 0000

910125-                   # To date and time default to NOW

-910125                   # From date and time default to 000000.0000


ACCESSED=-910125          # gives files accessed prior to 910125

ACCESSED=910125           # gives files accessed on or after 910125

ACCESSED=910125-910127    # gives files accessed on 910125 or 910126.
```

SET/COPYLEVEL `[RANGE]`

`RANGE` range of copy levels codes. If a range or list is not specified, a value of -1 is used, indicating that no check on the copy level should be made.

Use the `SETCOPYLEVEL` command to set the copylevel, or range of copylevels that will be applied to subsequent catalogue searches.

Example of valid copy level ranges

```
FM>set/copy 1,3,5-7 | Set copy levels 1,3 and 5 to 7 inclusive

FM>set/copy -1      | No check on copy level will be made
```

SET/DATAREP `[RANGE]`

`RANGE` range of data representation codes. If a range or list is not specified, a value of -1 is used, indicating that no check on the data representation code should be made.

Use the `SETDATAREP` command to set the data representation, or range of data representation values that will be applied to subsequent catalogue searches.

SET/DESTINATION `DESTINATION-SITE`

`DESTINATION` Destination site for `CHEOPS` transfer

Use the `SETDESTINATION` command to set the name of the destination site for `CHEOPS` transfer. The current list of destination sites is

```
CERN
HELSINKI
LISBON
SARDINIA
ATHENS
```

If a name that is not in this list is specified, then a warning message will be issued.

SET/LOCATION `[RANGE]`

`RANGE` range of location codes. If a range or list is not specified, a value of `-1` is used, indicating that no check on the location code should be made.

Use the `SETLOCATION` command to set the location code, or range of location codes that will be applied to subsequent catalogue searches.

Example of valid location code ranges

```
FM>set/loc 1,3,5-7 | Set location codes 1,3 and 5 to 7 inclusive

FM>set/loc -1      | No check on location code will be made
```

SET/LOCCODES `[FILE]`

`FILE` Name of the file containing the location code definitions. See the description of the `FMLCOD` routine for an example of the file format.

SET/MEDIATYPE `[RANGE]`

`RANGE` range of media types. If a range or list is not specified, a value of -1 is used, indicating that no check on the media type should be made.

Use the `SETMEDIATYPE` command to set the media type, or range of media types that will be applied to subsequent catalogue searches.

Example of valid media type ranges

```
FM>set/media 1,3,5-7 | Set media types 1,3 and 5 to 7 inclusive

FM>set/media -1      | No check on media type will be made
```

SET/SOURCE SOURCE-SITE

SOURCE Source site for CHEOPS transfer

Use the SETSOURCE command to set the name of the source site for CHEOPS transfer. The current list of source sites is

- CERN
- HELSINKI
- LISBON
- SARDINIA
- ATHENS

If a name that is not in this list is specified, then a warning message will be issued.

SET/USERWORDS [UWORD1] [UWORD2] [UWORD3] [UWORD4] [UWORD5] [UWORD6] [UWORD7] [UWORD8] [UWORD9] [UWORD10]

UWORD1 range or value for user word 1
UWORD2 range or value for user word 2
UWORD3 range or value for user word 3
UWORD4 range or value for user word 4
UWORD5 range or value for user word 5
UWORD6 range or value for user word 6
UWORD7 range or value for user word 7
UWORD8 range or value for user word 8
UWORD9 range or value for user word 9
UWORD10 range or value for user word 10

Use the SETUSERWORDS command to set the values or ranges for the 10 user words that will be used in subsequent search commands. Values are 0 or positive integers, ranges are delimited by a minus sign. To indicate that no check is to be made on a given user word, specify -1.

Example of setting userwords

```
FM>set/userwords 0 3-5 77 -1 99
```

With the above settings, entries must have userword 1 be 0, userword 2 must be in the range 3 to 5 inclusive, userword 3 must be 77, no check is made on userword 4 and userword 5 must be 99. No change is made to the acceptable values for userwords 6 to 10, which are set at initialisation time to -1.

SHOW/COPYLEVEL `RANGE`

Use the `SHOWCOPYLEVEL` command to display the list of `COPYLEVEL` codes currently in effect (see `SETCOPYLEVEL` command).

SHOW/DATAREP `RANGE`

Use the `SHOWDATAREP` command to display the list of `DATAREP` codes currently in effect (see `SETDATAREP` command).

SHOW/DESTINATION

Use the `SHOWDESTINATION` command to display the current destination site for `CHEOPS` transfers.

SHOW/LOCATION `RANGE`

Use the `SHOW LOCATION` command to display the list of `LOCATION` codes currently in effect (see `SETLOCATION` command).

SHOW/LOCCODES

Use the `SHOWLOCCODES` command to display the current location code definitions, as defined in the `fatmen.loccodes` file.

SHOW/MEDIATYPE `RANGE`

Use the `SHOWMEDIATYPE` command to display the list of `MEDIATYPE` codes currently in effect (see `SETMEDIATYPE` command).

SHOW/SOURCE

Use the `SHOWSOURCE` command to display the current source site for `CHEOPS` transfers.

SHOW/USERWORDS

Use the `SHOWUSERWORDS` command to display the list of `USERWORD` values or ranges currently in effect (see `SETUSERWORDS` command).

TAG `GNAME [KSN] [TAG] [CHOPT]`

`GNAME`	Generic name of the entry to be tagged
`KSN`	Key serial number. If not specified, the default FATMEN selection will be used.
`TAG`	Tag
`CHOPT`	Options

- `D` delete the tag
- `G` get and display the tag
- `S` set the tag
- `B` to select the BINARY tag (stored as CHARACTER*255)
- `T` to select the TEXT tag (default)
- `V` to select the VOLINFO tag

Use the `TAG` command to get, set or delete the TMS TAG associated with the a tape volume that corresponds to the specified generic name.

TOUCH `GNAME [KSN] [CHOPT]`

`GNAME`	Generic name of the entry to be touched
`KSN`	Key serial number. If not specified, the default FATMEN selection will be used.
`CHOPT`	Options

- `O` reset owner, node and job of creator etc.
- `T` update date and time of last access
- `U` zero use count
- `A` set account field
- `C` clear comment field

Use the `TOUCH` command to reinsert an existing entry in the catalogue. If a negative key serial number is given, all matching entries will be updated.

TREE `[PATH] [NLEVEL] [OUTPUT]`

`PATH`	Path name
`NLEVEL`	number of levels to be displayed
`OUTPUT`	Output filename. If not specified, the output will be displayed on the terminal.

Use the `TREE` command to draw a directory tree starting at the specified directory down `NLEVEL` levels.

Example of using the `TREE` command

```
FM>pwd
Current working directory = //CERN/L3
FM>tree
 FMTREK. directory tree structure below //CERN/L3 down          99  levels
          /DRE
          /JUNK
          /CDREMM
          /DREMM
          /PROD
               /DATA
                    /LDRE
                    /SDSUJK
                    /SDREJK
                    /SDSUCR
                    /SDSUBG
                    /SDRECR
                    /SDREBG
                    /SDREQQ
                    /SDSUQQ
                    /SDRESG
                    /SDSUTK
                    /SDSUSG
                    /SDRETK
                    /SDRETT
                    /SDSUTT
                    /SDREMM
                    /SDSUMM
                    /SDSUMX
                    /SDREMX
```

```
                    /SDREEM
                    /SDSUEM
                    /SDSUEE
                    /SDRENP
                    /SDREEE
                    /SDSUNP
                    /MDSU
                    /PDRE
                    /HESMRY
                    /MDRE
                    /SDREHG
                    /SDSUBH
                    /SDREGG
                    /SDREBH
                    /SDSUHG
                    /SDSUGG
          /LEBRUN
          /TEST4

          44  subdirectories found
FM>
```

UNLOCK GENAM [KSN] [CHOPT]

GENAM generic name

KSN Key serial number. If not specified, the default FATMEN selection will be used.

CHOPT Options

Use the UNLOCK command to enable WRITE access to the tape on which the file specified resides. If a negative key serial number is given, all entries for the specified generic name will be unlocked.

UPDATE MAX NGROUP IFLAG

MAX — Maximum number of updates. If not specified, a default value of 999 is used.

NGROUP — Number of updates to be grouped together. The default is to send each update individually. This value is ignored except on VM/CMS systems

IFLAG — Flag controlling the option required

- -1 Reset to defaults
- 0 Send any outstanding updates
- 1 Purge any outstanding updates

Use the UPDATE command to control the updating mode of the FATMEN package. (See on Page 57 for details).

N.B. if updates are grouped together, care must be taken to leave the FATMEN shell with the exit command.

VERSION

Use the `VERSION` command to display the version of the FATMEN package. This information is also displayed at initialisation time.

VIEW `GNAME [KSN]`

`GNAME`	Generic name of the file to be editted.
`KSN`	Key serial number. If not specified, the default FATMEN selection will be used.

Use the `VIEW` command to edit a local disk file. This command will be extended to support tape and remote files.

ZOOM `PATH`

`PATH`	Path name to be followed

Use the `ZOOM` command to move to the first directory matching the specified path name that contains no subdirectories and one or more files.

Example of the ZOOM command

```
Current Working Directory = //CERN/DELPHI
FM> zoom
Current Working Directory = //CERN/DELPHI/P01_ALLD/CDST/PHYS/Y90V03/E093.3/L0312
FMZOOM. files:          30

...

Current Working Directory = //CERN/OPAL
FM> ld
List of subdirectories...
//CERN/OPAL/PLAY
//CERN/OPAL/PROD
//CERN/OPAL/EVKI
//CERN/OPAL/SIMD
//CERN/OPAL/MDST
//CERN/OPAL/STEST
//CERN/OPAL/RAWD
//CERN/OPAL/DDST
//CERN/OPAL/JAMIE
Total of          9 subdirectories of which          9 match
FM> zoom prod/*/p1*
Current Working Directory = //CERN/OPAL/PROD/PASS3/FILT/P1R1033L039
FMZOOM. files:           4
```

8.2 Using the command line interface

The command line interface is started by typing FM at the command level. If a KUIP macro named FATMEN KUMAC is found, this will be executed. It is normally used used to initialise FATMEN, and set the current default directory, if required. For example, the following macro could be used by a member of ALEPH:

```
INIT ALEPH
```

An example of starting a session using a FATMEN KUMAC for the throng CNDIV is given below. Once the session has been started, commands such as CD and MKDIR can be issued, as shown in the example.

Starting up FATMEN on VM/CMS and issuing a few commands

```
TYPE FATMEN KUMAC
INIT CNDIV     | Name of DD throng
fm                  | Enter FATMEN
FMINIT.  Initialisation of FATMEN package
         This version created on      891109  at        1049
Linked to FMCNDIV            mode W
FAOPEN : for FARZ on Unit    1 opened File CERN FATRZ
Current Working Directory = //CERN/CNDIV
CD GOOSSENS         | Go to user's top directory
Current Working Directory = //CERN/CNDIV/GOOSSENS
MKDIR DCF           | Make the DCF subdirectory
FM>
```

At this point a message is sent to the FATMEN service machine and since the actual creation of the subdirectory occurs asynchronously, one should wait a certain time before entering further commands which would use the sub-directory just created.

Although entries may be made by specifying the full pathname, it is good practice to first create the desired directory structure to avoid warning messages from internal ZEBRA RZ routines. In fact new experiments are strongly recommended to adopt a convention for file and directory names, and to create their directory tree before cataloging files in the FATMEN system.

8.3 Using KUIP macros with FATMEN CLI

KUIP macros may be used to execute a whole sequence of commands, e.g. entering data into **pre-existing** directories (created as shown above).

The KUIP macro FATEX1 KUMAC containing FATMEN commands

```
cd GOOSSENS/DCF/R3            | Go to target directory (DCF Release 3)
*   Enter information about tape MG2325 with its four files
add/tape MG2325 MG2325 1 Prodid file1 VMF 0 CERNVM 'DCF Product idendification'
add/tape MG2325 MG2325 2 MEMO   file2 VMF 0 CERNVM 'Memo to users'
add/tape MG2325 MG2325 3 EXECS  file3 VMF 0 CERNVM 'Installation execs'
add/tape MG2325 MG2325 4 Binary file4 VMF 0 CERNVM 'TEXT files and verification'
cd \ \ SMFF/R1         | Reset target directory (SMFF Release 1)
*   Enter information about tapes MG2201 and MG2309 with each one file
add/tape MG2201 MG2201 1 PTF_M5 file1 VMF 0 CERNVM 'SMFF PTF Mod-level 5'
add/tape MG2309 MG2309 1 PTF_M6 file1 VMF 0 CERNVM 'SMFF PTF Mod-level 6'
cd \ R3                  | Next SMFF subdirectory
*   Enter information about tape MG2401 with one file   (SMFF Release 3)
add/tape MG2401 MG2401 1 PTF_M2 file1 VMF 0 CERNVM 'SMFF PTF Mod-level 2'
```

The effect of running the KUIP macro file shown above is shown below:

Running a KUIP macro containing FATMEN commands

```
FM> TRACE ON           | Set trace to see what is happening
FM> EXE FATEX1         | Execute KUIP macro FATEX1
>>> trace on
>>> cd GOOSSENS/DCF/R3
Current Working Directory = //CERN/CNDIV/GOOSSENS/DCF/R3
>>> add/tape IC2325 IC2325 1 Prodid file1 VMF 0 CERNVM 'DCF Product
idendification'
//CERN/CNDIV/GOOSSENS/DCF/R3/PRODID
PUN FILE 0033 SENT TO   FMCNDIV RDR AS  0050 RECS 0009 CPY  001 A NOHOLD NOKEEP
FMFZO - Your update has been sent to the server
>>> add/tape IC2325 IC2325 2 MEMO   file2 VMF 0 CERNVM 'Memo to users'
//CERN/CNDIV/GOOSSENS/DCF/R3/MEMO
PUN FILE 0034 SENT TO   FMCNDIV RDR AS  0051 RECS 0008 CPY  001 A NOHOLD NOKEEP
FMFZO - Your update has been sent to the server
>>> add/tape IC2325 IC2325 3 EXECS  file3 VMF 0 CERNVM 'Installation execs'
//CERN/CNDIV/GOOSSENS/DCF/R3/EXECS
PUN FILE 0035 SENT TO   FMCNDIV RDR AS  0052 RECS 0008 CPY  001 A NOHOLD NOKEEP
FMFZO - Your update has been sent to the server
>>> add/tape IC2325 IC2325 4 Binary file4 VMF 0 CERNVM 'TEXT files and
verification'
//CERN/CNDIV/GOOSSENS/DCF/R3/BINARY
PUN FILE 0036 SENT TO   FMCNDIV RDR AS  0053 RECS 0009 CPY  001 A NOHOLD NOKEEP
FMFZO - Your update has been sent to the server
>>> cd
SMFF/R1
Current Working Directory = //CERN/CNDIV/GOOSSENS/SMFF/R1
>>> add/tape IC2201 IC2201 1 PTF_M5 file1 VMF 0 CERNVM 'SMFF PTF Mod-level 5'
//CERN/CNDIV/GOOSSENS/SMFF/R1/PTF_M5
PUN FILE 0037 SENT TO   FMCNDIV RDR AS  0054 RECS 0009 CPY  001 A NOHOLD NOKEEP
FMFZO - Your update has been sent to the server
```

```
>>> add/tape IC2309 IC2309 1 PTF_M6 file1 VMF 0 CERNVM 'SMFF PTF Mod-level 6'
//CERN/CNDIV/GOOSSENS/SMFF/R1/PTF_M6
PUN FILE 0038 SENT TO   FMCNDIV RDR AS  0055 RECS 0009 CPY  001 A NOHOLD NOKEEP
FMFZO - Your update has been sent to the server
>>> cd \ R3
Current Working Directory = //CERN/CNDIV/GOOSSENS/SMFF/R3
>>> add/tape IC2401 IC2401 1 PTF_M2 file1 VMF 0 CERNVM 'SMFF PTF Mod-level 2'
//CERN/CNDIV/GOOSSENS/SMFF/R3/PTF_M2
PUN FILE 0039 SENT TO   FMCNDIV RDR AS  0056 RECS 0009 CPY  001 A NOHOLD NOKEEP
FMFZO - Your update has been sent to the server
FM>
```

Continuation lines within KUIP macros

If a command does not fit completely in one line it can be wrapped on the next line(s). If the last character of a line is an underscore that line will be logically joined to the next one (excluding the underscore obviously). The next line may have also an underscore. The resulting full command line is however limited to a maximum of 255 characters.

A FATMEN KUMAC using continuation lines

```
CD //CERN/CHARM2/TEST/DST1/ELEC/HO20/NOM/E02
add/tape NH0391 NH0391 1  NOSC FILE1  EP 0 CERNVM _
'(11513) 4001 4002 4003 4004'
CD //CERN/CHARM2/TEST/DST1/ELEC/HO20/NOM/E02.5
add/tape NH0391 NH0391 2  NOSC FILE2  EP 0 CERNVM _
'(8841)  4005 4006'
CD //CERN/CHARM2/TEST/DST1/ELEC/HO20/NOM/E03
add/tape NH0392 NH0392 1  NOSC FILE1  EP 0 CERNVM _
'(19978) 3989 3990 3991 3992 3993 3994'
CD //CERN/CHARM2/TEST/DST1/ELEC/HO20/NOM/E04
add/tape NH0392 NH0392 2  NOSC FILE2  EP 0 CERNVM _
'(12798) 3995 3996 3997'
```

8.4 Accessing the data in the FATMEN database

The data, which is entered in the FATMEN database can be accessed interactively running the FATMEN program. In the session we have a look at the data, which was entered running the KUIP macro `FATEX1 KUMAC`, discussed previously.

Accessing information in the FATMEN database

```
FM> pwd
Current Working Directory = //CERN/CNDIV/GOOSSENS/SMFF/R3
FM> cd \\                 | Go up two levels in the directory
Current Working Directory = //CERN/CNDIV/GOOSSENS
FM> ld
List of subdirectories...
DCF
SMFF
Total of   2 subdirectories
FM> cd dcf
```

```
Current Working Directory = //CERN/CNDIV/GOOSSENS/DCF
FM> ld
List of subdirectories...
R3
Total of   1 subdirectories
FM> cd r3
Current Working Directory = //CERN/CNDIV/GOOSSENS/DCF/R3
FM> ls
BINARY
PRODID
MEMO
EXECS
Files:    4
FM> cd \\                | Up to user's top directory again
Current Working Directory = //CERN/CNDIV/GOOSSENS
FM> cd smff/r1
Current Working Directory = //CERN/CNDIV/GOOSSENS/SMFF/R1
FM> ls
PTF_M5
PTF_M6
Files:    2
FM> fc                       | Count files in current working directory
Files:    2
FM> * List files using wildcard construct and display comment field (c option)
FM> *                                          time and date fields (t option)
FM> ls ptf_m% -ct
Generic filename: PTF_M5
Comment: SMFF PTF MOD-LEVEL 5
Date and time of creation:     891114 1544
Date and time catalogued:      891114 1544
Date and time last accessed:        0    0
Generic filename: PTF_M6
Comment: SMFF PTF MOD-LEVEL 6
Date and time of creation:     891114 1544
Date and time catalogued:      891114 1544
Date and time last accessed:        0    0
Files:    2
FM> cd \ \ dcf/r3
Current Working Directory = //CERN/CNDIV/GOOSSENS/DCF/R3
FM> ls binary -a             | Display all information about entry
Generic filename: BINARY
Copy level:  0 Media type: 2 Location code:   1 File serial number:   4
Comment: INSTALLATION EXECS
Start record:     0 End record:     0 Start block:     0 End block:     0
File format: VMF  user format:
VSN: IC2325 VID: IC2325 FSEQ:    3
Fileid:      FILE3
Created by:  FAT3     ACCT: JDS$CT   on node: CERNVM   by job: FAT3
RECFM:       LRECL:     0 BLKSIZE:     0 FILESIZE:     0
File protection mask:          00000000
Date and time of creation:     891114 1544
Date and time catalogued:      891114 1544
Date and time last accessed:        0    0
User words:  00000000 00000000 00000000 00000000 00000000
             00000000 00000000 00000000 00000000 00000000
Files:    1
FM> end
```

```
DISK CNDIV (Your 19C Mode W ) released/detached.
FMEND. Terminating FATMEN package
```

Example of data staging with the FATMEN database

```
FM > cd chris                  | Go to relevant working directory
Current Working Directory = //CERN/CNDIV/CHRIS
FM > ls tape8 -nmc
Generic filename: TAPE8
Comment: ANOTHER UA2 TAPE
VSN: UW0060 VID: UW0060 FSEQ:    1
Fileid:      UA2DST
Files:    1
FM> find tape8 iofile13
EXEC STAGE IN IOFILE13  UW0060.1.SL.UW0060 (WAIT SIZE 200 DEN 38K
STAGE: Request sent to VMSTAGE - do NOT type at all until it replies. DO NOT
IPL CMS.
VMSTAGE:14Nov89-15:55:13 Staging-in job submitted to read tape
                         UW0060.1.SL.UW0060
VMSTAGE:14Nov89-15:55:13 If you choose not to wait, use HX or RETURN twice to
                         break out. Do NOT IPL CMS
STAGE: 16:00 waited 5/180 mins for staging request.
STAGE: 16:05 waited 10/180 mins for staging request.
STAGE: 16:10 waited 15/180 mins for staging request.
STAGE: 16:15 waited 20/180 mins for staging request.
STAGE: 16:20 waited 25/180 mins for staging request.
VMSTAGE:14Nov89-16:25:00 Staging-in retcode 0 tape info: UW0060.1.SL.UW0060
                         dcb= U 12600 12600 nreads=64 nMbytes=0.769042969
STAGE: 16:25 waited 30/180 mins for staging request.
STG517 ( 019B  V ) RR
FM > shell q filedef
Executing ... Q FILEDEF
FT06F001 TERMINAL
FT02F001 PUN
FT05F001 TERMINAL
FT07F001 PUN
DF@00002 DISK     CERN     FATRZ
IOFILE13 DISK     TUW0060  FSEQ1
```

8.5 Access to the Tape Management System

The CERN Tape Management System is fully integrated into the FATMEN package in an automatic and transparent manner. In addition, direct access to the TMS is possible via the SYSREQ interface, which provides both a FORTRAN callable and a command line interface.

Jobs running on remote systems will interface to the local or CERN TMS as defined by the local implementation.

The CERN Tape Management System is described in DD/TMS/UG.

Accessing existing tape data

When an attempt is made to access existing data which resides on tape, the file catalogue routines will check access rights to and availability of the tape in question. It will then issue the appropriate STAGE command to copy the data onto disk. No knowledge of the tape details, such as file number, dataset name or location, are required by the user.

Creating new tape data

When creating new datasets on tape, the FATMEN system will use the next free tape that is allocated to the user or group, according to the access rules defined in the Tape Management System for the group in question. The default medium is 3480. Alternatively, the user may override the default options, specifying a different media type, an explicit tape volume etc. This allows for tape allocation to be performed at run time, job submission time, or at any time prior to job submission time.

Part IV

FATMEN – Installation and Management Guide

Chapter 9: General hints

9.1 Availability of PAM files, libraries and FATMEN shell

The FATMEN package is installed as part of the CERN program library. If you have a version of the CERN program library corresponding to CERN Computer Newsletter 197 or later, you should have FATMEN on your system.

The standard installation (CNL 201 and after) creates the FATMEN callable interface (part of PACKLIB), the FATMEN shell and the FATMEN server.

9.2 Using ZEBRA, HBOOK etc. with FATMEN

The size of the users' store

The FATMEN package creates a division of type 'P' (package) in the store that the user declares in the call to FMINIT.(See on Page 56) This division is declared with NW=10000 and NWMAX=100000. (See the ZEBRA users' guide[4] for details of the routine MZDIV). Thus, the users store should be sufficiently large to accomodate this division.

Using HBOOK and FATMEN

If the user also wishes to use HBOOK, ZEBRA should be initialised first followed by a call to HLIMIT with a negative argument. This will indicate to HBOOK that ZEBRA has already been initialised.

Note that the FATMEN FORTRAN routines do not save and restore the current directory within an RZ file.

Calling MZWIPE

It is normally safe to call MZWIPE to clean divisions in the store which contains the FATMEN division with the following exception:

- Do not use MZWIPE to delete all package divisions (`IXWIPE=IXSTOR+23`). This will delete all banks created with FMLIFT (see Page 114) and not saved using FMPUT (see Page 60).

9.3 Using FATMEN without a Tape Management System

The only Tape Management Systems currently supported by FATMEN are the HEPVM TMS, and VMTAPE. These are selected by the PATCHY statements

```
+USE,TMS.

and

+USE,VMTAPE.

respectively.
```

The Tape Management System is responsible for maintaining information on tape volumes, such as availability, label type etc. When installed without the TMS flag, FATMEN takes the default values for each media type as defined at installation time, or by a call to the routine FMEDIA (or the MEDIA command in the shell).

To see the current defaults, type the command MEDIA in the shell.

As well as tailoring the default values for each media type, a user exit, FMUTMS, can be provided to override the default values for a given volume. See the description of the FMUTMS routine in the user interface section of this manual for more details.

Chapter 10: Installing FATMEN

As described above, the installation of the FATMEN software as part of the standard CERN program library installation. FATMEN relies heavily on the program library, and so this is a pre-requisite for its usage.

The standard program library installation procedure will generate the FATMEN FORTRAN interface (in PACKLIB), and three modules. These are the shell (FM), the server (FATSRV) and a program to create a new, empty RZ file for use with FATMEN (MKFATNEW).

`mkfatnew` is just a simple script that calls the `FATNEW` program. For example, on VAX/VMS systems the following maybe used.

Example MKFATNEW command file

```
$! fatsys:==CERN ! For example
$! fatgrp:==LHC  ! For example
$
$  type/nopage sys$input

Please give the name of the FATMEN system. This name forms
the top-level of the FATMEN catalogue, e.g. //CERN

$eod
$  inquire/nopunc ans "FATSYS? "
$  if ans.eqs."" then ans = "CERN"
$  fatsys==ans - "//"
$  type/nopage sys$input

Please give the name of the FATMEN group.

$eod
$  inquire/nopunc ans "FATGRP? "
$  if ans.eqs."" then exit
$  fatgrp==ans - "FM"
$  write sys$output ""
$  inquire/nopunc fatdir "Directory where FATMEN catalogue should reside? "
$  olddir = f$environment("DEFAULT")
$  set default 'fatdir'
$  create/directory [.todo]
$  create/directory [.tovm]          ! Only at CERN !!!
$  create/directory [.done]
$  set file/protection=w:rw todo.dir
$  fatman="FM''fatgrp'"
$  set file/acl=(id='fatman',access=read+write,options=default) todo.dir
$  set file/protection=w:rw tovm.dir ! Only at CERN !!!
$  set file/protection=w:rw done.dir
$  run cern:[pro.exe]fatnew
$  set default 'olddir'
```

10.1 Installing FATMEN on a new machine

The first step is the installation of the CERN program libraries. Once this has been achieved, FATMEN should be configured as below.

Access to data

Access to data through FATMEN is currently supported on:

1 VM/CMS systems running the HEPVM software, in particular SETUP and STAGE.

2 VM/CMS systems without the HEPVM software are supported provided that the CERN REXX local function package as well as the GIME and DROP execs are installed.

3 VM/CMS systems running VMBATCH and/or VMTAPE, provided that the REXX local function package as well as the GIME and DROP execs are installed.

4 VAX/VMS systems. The CERN Program Library Package VAXTAP is required for access to tapes.

5 Unix systems. Tape support is currently only possible on the CERN Cray system, the CERN SHIFT project and the L3 Apollo network at CERN.
CERN is developing generalised Unix tape software (staging and tape mounting) which will be used to provide tape support on all other Unix systems. This software is based upon the existing Cray and SHIFT syntax and so the FATMEN interface is already written.

6 MVS systems. Only FORTRAN I/O is supported in the current version. The IBM FORTRAN routine FILEINF is used whereever possible, together with the CERN Program Library Package FTPACK, written by R. Matthews.

On all other systems some modifications to the code are required for data access. These modifications are restricted to the routines FMOPEN, FMCLOS and FMCOPY.

Using the FATMEN catalogue

The FATMEN catalogue is simply a FORTRAN direct access file, processed using the CERN ZEBRA package. Users access the file in read-only mode, with updates being sent to and applied by a server (one per experiment). Updates are sent as ZEBRA FZ files (exchange mode, ASCII mapping).

Configuring FATMEN

VM/CMS systems

Machines that do not run the HEPVM software may use FATMEN provided that they install the GIME and DROP execs, and the REXX local function package. To install the REXX local function package, simply copy the files RXLOCFN SEGMENT and RXLOCFN MODULE to your local machine. They will be automatically loaded when needed. Some functions in this package will return a null-string or zero unless certain HEPVM CP modifications are installed. However, all of the functions required for FATMEN will work without any problem.

If you require any of the above software and do not have access to CERNVM (or any other machine running the HEPVM software), simply sent a mail message to JAMIE@CERNVM.

Upon initialisation the FATMEN software issues an 'EXEC GIME FMgroup', e.g. 'EXEC GIME FML3'. Thus, if the FATMEN catalogue for L3 is NOT kept on the 191 disk of FML3, an appropriate entry in the GIMEUSER, GIMEGRP or GIMESYS NAMES file is required. An additional entry is also required in the normal NAMES file so that the updates can be correctly sent to the appropriate service machine.

Systems running VMBATCH should activate the necessary code by putting

```
+USE,VMBATCH.
```

into the installation cradle for FATMEN. (Different code is required to obtain the username, jobname and account field in a VMBATCH job to that used in HEPVM batch).

Systems running VMTAPE should activate the necessary code by putting

```
+USE,VMTAPE.
```

This will provide access to tape files. If user tapes are catalogued in the VMTAPE catalogue (TMC), select also

```
+USE,VMTMC.
```

Systems running both VMTAPE and VMBATCH may simply add

```
+USE,VMCENTER.
```

VAX/VMS systems

Upon initialisation, the client looks for a file in the directory specified by the symbol FMgroup, e.g. FML3. If the FATMEN catalogue is kept in DISK$FAT:[FATMEN.L3], one should type

```
FML3:==DISK$FAT:[ FATMEN.L3]
```

This directory must contain the subdirectories [.TODO], [.DONE]. The [.TODO] directory must be writeable by all members of the L3 collaboration (in this example).

If the catalogue is on a disk that has disk quotas enabled, then an ACL must be established on the directory [.TODO] as in the example below. In this example, the top level directory is [FMCDF] belonging to user FMCDF. The identifier CDF_EXPERIMENT is held by all members of the CDF experiment and is an identifier with the RESOURCE attribute. Setting up the directory in this manner allows all CDF users to send updates to the FATMEN server for their experiment and avoids the need for entries in the quota file for every CDF user for the disk in question. Only entries for FMCDF and CDF_EXPERIMENT are required.

If disk quotas are not enabled on the volume in question, then allowing group write access to the [.TODO] directory is sufficient.

```
(IDENTIFIER=FMCDF,ACCESS=READ+WRITE+EXECUTE+DELETE+CONTROL)
(IDENTIFIER=CDF_EXPERIMENT,ACCESS=READ+WRITE+EXECUTE)
(IDENTIFIER=FMCDF,OPTIONS=DEFAULT,ACCESS=READ+WRITE+EXECUTE+DELETE+CONTROL)
(IDENTIFIER=CDF_EXPERIMENT,OPTIONS=DEFAULT,ACCESS=READ+WRITE+EXECUTE)
```

The configuration for the server is performed by the sample command file shown below.

For the server only, a variable FATSYS must be set to the appropriate name, should the catalogue name not be //CERN

```
$!
$! Example FATSERV.COM
$!
$ dd = f$cvtime(,,"WEEKDAY")
$ tt = f$time()
$ hh = f$trnlnm("SYS$NODE")
$ write sys$output ""
$ write sys$output "FATSERV starting at ''dd' ''tt' on ''hh'"
```

```
$ write sys$output ""
$ !
$ ! Set FATMEN system
$ !
$ FATSYS:==CERN
$ !
$ ! Set FATMEN group
$ !
$ FATGRP:==FMCNDIV
$ !
$ ! Set FATMEN wakeup interval in seconds
$ !
$ ! Define FMCNDIV if not defined in system login
$ !
$ FMCNDIV:==DISK$CERN:[ FMCNDIV]
$ FMWAKEUP:==30
$ !
$ ! Set FATMEN log level
$ !
$ FMLOGL:==3
$ write sys$output -
"FATMEN group set to ''FATGRP', wakeup interval is ''FMWAKEUP' seconds"
$ run cern:[ pro.exe] fatsrv
```

See the `VAXTAP` [13] long writeup for details on configuring the VAX tape handling software, in particular for remote and multi-file staging.

Unix systems

Upon initialisation, the client looks for a file in the directory specified by the variable FMexperiment, e.g. FML3. Thus, if the FATMEN catalogue is kept in /users/fatmen/l3, the variable FML3 should be set to this path:

```
FML3=/users/fatmen/l3; export FML3
```

This directory must contain the subdirectories todo and done. The todo directory must be writeable by all members of the L3 collaboration (in this example).

For the server only, a variable FATSYS must be set to the appropriate name, should the catalogue name not be //CERN

```
FATSYS=DESY;export FATSYS
```

The configuration for the server is performed by the sample script shown below.

```
#!/bin/sh
#
# Example FATSERV script.
#
t="date"
h="hostname"
echo
echo FATSERV starting at $t on $h
echo
FATGRP=FML3   ;export FATGRP
```

```
FATSYS=CERN   ;export FATSYS
FMWAKEUP=10;   export FMWAKEUP
FML3=/fatmen/fml3; export FML3
echo FATMEN group set to $FATGRP , wakeup interval is $FMWAKEUP seconds
echo
/cern/pro/bin/fatsrv
```

Chapter 11: Remote access to the FATMEN catalogue

Remote access to the FATMEN catalogue is currently supported over DECnet (between VAX/VMS systems only) NFS, AFS and CSPACK. CSPACK access has not yet been tested on VM/CMS or MVS systems.

11.1 DECnet access to FATMEN catalogues

To access a remote catalogue over DECnet, simply include the node specification in the symbol definition. e.g.

```
FMCDF:==FNALF::USR$ROOT37:[FMCDF]
```

11.2 NFS access to FATMEN catalogues

To access a remote catalogue over NFS, simply define the required environmental variable as in the example below.

```
FMOPAL=/fatmen/fmopal; export FMOPAL

df /fatmen
Filesystem    Total KB    free %used   iused %iused Mounted on
fatcat:/fatmen  409600  138704   66%   -       -    /fatmen
```

11.3 AFS access to FATMEN catalogues

To access a remote catalogue over AFS, one simply has to define the appropriate environment variable to point to the directory where the catalogue resides, e.g.

```
export FML3=/afs/cern.ch/fatmen/fml3
```

11.4 CSPACK access to FATMEN catalogues

If FATMEN has been installed with the `CSPACK` option enabled (`+USE,CSPACK` in the PATCHY step of the library build), then remote catalogues may also be accessed and updated using `CSPACK`. The standard `ZSERV` must be installed on the remote system on which the catalogue resides, as described in the `CSPACK` [5] manual. One then defines the symbol (VAX/VMS) or environmental variable (UNIX) that points to the directory where the catalogue resides with the syntax `node:path`.
Thus, to enable `CSPACK` access to the `L3` catalogue residing on the node `fatcat`, the appropriate `C shell` definition would be

```
setenv FML3 fatcat:/fatmen/fml3
```

11.5 FATCAT - the dedicated FATMEN server at CERN

At CERN, the FATMEN catalogues are stored on the node FATCAT. This is currently an RS6000 which is only used for FATMEN catalogue management. The recommended way of accessing a FATMEN catalogue at CERN is the following:

1 NFS mount the `/fatmen` file system

2 Define environmental variables to point to the directory of interest

An example of how this may be done is shown below.

NFS mounting the /fatmen file system on a Unix platform

```
mount fatcat:/fatmen /fatmen
```

NFS mounting the /fatmen file system on a VMS platform

```
$NFSMOUNT fatcat:"/fatmen" fatmen
```

Defining the environmental variables on a Unix system

```
#/bin/ksh
for i in /fatmen/fm*
   do
     typeset -u fatgrp
     fatpath=$i
     fatgrp=`basename $i`
     echo Setting $fatgrp to $fatpath ...
     eval $fatgrp=$fatpath;export $fatgrp
   done
```

Defining the environmental variables on a VAX/VMS system

```
$ loop:
$ fatman = f$search("FATMEN:[000000]FM*.DIR")
$ if fatman .eqs. "" then exit
$ fatman = f$parse(fatman,,,"NAME")
$ fatdir = "FATMEN:[''fatman']"
$ write sys$output "Setting ''fatman' to ''fatdir'..."
$ 'fatman' :== 'fatdir'
$ goto loop
```

Chapter 12: Distribution of catalogue updates

It is unfortunately at least unrealisitic and more likely impossible to access a single catalogue world-wide and so one typically has multiple copies of the catalogue. These catalogues are automatically keep up to date by the FATMEN servers. The updates may be sent over Bitnet, DECnet or TCP/IP. However, the servers must be configured so that they know to which nodes updates should be sent. In addition, it may be desirable to maintain subsets of the catalogue at remote sites. For example, information on raw data may remain at the laboratory where the data is taken whereas all DST information is sent to all collaborating institutes.

The configuration file that is used has the same format on all systems. This format is that of a VM/CMS names file. On non-VM systems, these files are processed using the FORTRAN callable routine, NAMEFD, available in the CERN program library KERNLIB.

12.1 Configuring servers on VM systems

This is performed by adding entries to the NAMES file of the servers in question. For each remote server one can define up to 16 generic name patterns, each of which may include wild-cards. For more details see the example below. Updates may be sent to remote IBM machines, VAXes connected via Interlink and Unix machines using the CSPACK software.

Note that the default for VM systems is to send the updates using the SENDFILE exec. This can be overridden for individual entries either by sending the updates to a gateway machine, as described later, or by specifying an alternate exec, as for the entry FMVAX.

```
:nick.FATSERVERS
              :list FMSACLAY FATCAT FMIN2P3 FMRAL FMVAX

:nick.FMVAX
              :userid.fmsmc
              :node.vxcrna
              :DIR1.//cern/smc/dst/*
              :exec.fat2vax

:nick.FMSACLAY
              :userid.fmsmc
              :node.frsac12
              :DIR1.//cern/smc/dst/*

:nick.FMIN2P3
              :userid.fmsmc
              :node.frcpn11
              :DIR1.//cern/smc/dst/*

:nick.FMRAL
              :userid.fmsmc
              :node.ukacrl
              :DIR1.//cern/smc/dst/*

:nick.FATCAT
              :userid.fmfatcat
              :node.cernvm
              :DIR1.//cern/smc/dst/*
```

In the preceeding example, all updates referring to generic names beginning //CERN/SMC are sent to the server defined by the name FATCAT, whereas only updates referring to names beginning //CERN/SMC/DST are sent to the servers at SACLAY, IN2P3 and RAL. Note the the FMFATCAT account is in fact a gateway into the Unix-world (using the program in PATCH FATCAT in the FATMEN PAM file and that FMVAX is an account of a VAX linked to the IBM via Interlink, but this is completely transparent to the server.

On systems other than CERNVM, updates that are to be sent to non-VM systems should pass via a gateway service machine.

```
:nick.FATDESY  :userid.R01H1
               :node.dhhdesy3
               :DIR1.//cern/h1/*
               :protocol.mvsjob
               :gateway.fmgate at cernvm

:nick.FATVAX   :userid.fmh1
               :node.vxdesy
               :dir1.//cern/h1/*
               :protocol.tcpip
               :gateway.fmgate at cernvm
               :receive.yes
```

In the above example, updates destined for FATDESY and FATVAX are not send directly, but pass via an intermediate service machine. wakes up at predefined intervals and transfers any files to the specified remote nodes. In the case of the entry FATVAX, files are also retrieved from the remote system.

Transferring updates to VAX/VMS systems via Interlink

This method should only be used at CERN. For other systems, a gateway service machine should be used. See the description below for setting up a gateway service machine.

Files sent from IBM VM/CMS systems via the Interlink VM-DECnet gateway will arrive in the default directory for the FAL DECnet object, typically

```
SYS$SPECIFIC:[ FAL$SERVER ]
```

By default, the files will be named FATMEN.RDRFILE. At CERN a small modification has been made to the Interlink software so that the files arrive as VAXUSER.FILENAME_FILETYPE, thus files sent to FMOPAL would be called

```
FMOPAL.FATMEN_RDRFILE
```

These files can be copied to the correct directory using the command file FATRL.COM, included in the PATCH DCL of the FATMEN pamfile.

12.2 Configuring servers on VMS, MVS and Unix systems

The names file configuration is exactly the same on VMS, MVS and Unix systems as on VM nodes. The processing of the names file is performed by a FORTRAN program, FATSEND, which uses the CERN program library routine NAMEFD and the updates are transferred using the CSPACK routines.

For each remote server, a subdirectory should be created. (If the subdirectories are note created, they will be created as required by the server). In the case of the example names file shown below, we would have the subdirectories [.TOD0SG01], [.TOD0SF01] and so on (VAX/VMS systems). The FATMEN server copies all updates into each of these subdirectories and the program FATSEND copies the updates to the specified remote nodes, using either TCP/IP or DECnet, and deletes those files that are successfully copied. DECnet transfers are only possible between VAX/VMS systems.

On MVS systems, the protocol BITNET is also valid for transmission of updates to remote VM systems. This is done using the TSO transmit command.

An example names file is shown below.

```
:nick.FATSERVERS
              :list FMSGI FMVAX

:nick.FMSGI   :userid.fmd0
              :node.d0sg01
              :protocl.tcpip
              :dir1//fnal/d0/*
              :receive.yes
              :queue:USR$ROOT37:[FMD0.TODO]

:nick.FMVAX   :userid.fmd0
              :node.d0sf01
              :protocl.tcpip
              :dir1//fnal/d0/*
```

In this example, servers connecting the the machine D0SG01 will transmit updates in both directions, rather than the default, which is to send updates only. This option is useful for the transmission of updates to a machine that does not accept incoming connections, as is the case with VAX systems running the UCX TCP/IP software, the Cray at CERN and MVS systems.

If the tag :queue is specified, the updates are placed in the specified directory. If not, they are placed in the subdirectory todo of the account that is used to performed the transfer.

If a subdirectory TOVM exists, the FATMEN server will attempt to send all updates via Interlink to CERNVM. This option is only intended for use at CERN.

12.3 Using a gateway service machine on VM systems

To send updates from a VM node into a non-VM system, use the :gateway tag to define a gateway service machine. The VM service machine will send all updates to this gateway machine, which in turn will transmit them by the specified protocol at regular intervals using the program FATSEND.

Valid protocols are TCPIP, BITNET and MVSJOB. TCPIP, which is the default, results in the files being transferred using the CSPACK routines, as described above. One should normally also set the :receive.yes option so that updates are also retrieved from these remote systems.

Specfiying BITNET together with a gateway machine permits updates to be sent to remote nodes at specified intervals, rather than immediately.

Although it is possible to use SENDFILE to send files to MVS nodes, these files are generally unreadable and hence the preferred method is to specify the protocol MVSJOB. This will result in an IEBGENER job being submitted to the remote node, which will copy the updates to a temporary file in the input queue of the specified server. The usual restrictions on remote job submission apply.

Chapter 13: The Program FATSEND

This program is generated using the command MAKEPACK FATSEND. Alternatively, one may use a job such as the following:

```
VAX/VMS systems

$if p1 .nes. "" then goto 'p1
$ y:
$ypatchy cern:[pro.pam]ZEBRA.pam fatsend.for :go
+use,qcde.
+use,vaxvms,*fatsend.
+exe.
+pam,11,r=qcde.
+pam,12,t=c,a. fatmen.cards
+quit
$ f:
$for fatsend
$ l:
$link fatsend,'lib$,sys$input/opt
sys$library:vaxcrtl/shareable
sys$system:sys.stb
multinet_socket_library/share
$ exit

Unix systems (Use f77 instead of xlf on non-AIX machines)

ypatchy /cern/pro/pam/ZEBRA.pam fatsend.f :go <<!
+use,qcde.
+use,ibmrt,*fatsend.
+exe.
+pam,11,r=qcde.
+pam,12,t=c,a. fatmen.cards
+quit
!
xlf -c -q extname -q charlen=32000 fatsend.f
xlf fatsend.o -L/cern/new/lib -lpacklib -lc -o fatsend
```

The program can be tailored by setting the variables FMLOGL and FMWAKEUP, e.g.

```
$fmlogl:==3
$fmwakeup:==3600 ! Once an hour
```

On Unix systems, the tailoring is done using environmental variables. The following example is for the C shell.

```
setenv fmlogl 3
setenv fmwakeup 3600
```

On VM/CMS systems, the SETENV command should be used, as in

```
SETENV FMWAKEUP 3600
```

On MVS systems no equivalent to environmental variables exists. Therefore names files entries are used, as shown below.

```
:nick.fatsrv  :wakeup.60   :logl.0

:nick.fatsend :wakeup.3600 :logl.3

:nick.FMSGI
              :mvsid.r01d0
              :userid.fmd0
              :node.d0sg01
              :protocl.tcpip
              :dir1//fnal/d0/*
```

Chapter 14: Installing VAXTAP for tape access on VAX/VMS systems

FATMEN interfaces to the CERN Program Library package VAXTAP to provide tape support on VAX/VMS systems. The long writeup should also be consulted. [13]

The package is installed by running a command file that can be generated by the following PATCHY run:

```
$YPATCHY CERN:[PRO.PAM]VAXTAP.PAM INSTALL.COM :GO
+USE,INSTALL,T=EXE.
+PAM.
+QUIT
```

Once this command file has been extracted, installation of the package proceeds by typing

```
@INSTALL
```

and answering the questions. If VAXTAP is to be installed on a system without access to the HEPVM Tape Management System, as is likely to be the case when installing it outside CERN, answer NO to the first question. Answering A or ALL to the second question will cause the installation to complete without any further dialogue.

When the command file completes, the file SETUP_STARTUP.COM should be edited and tailored.

```
$ !-----------------------------------------------------------------------*
$ !
$ !      Startup command file for SETUP/STAGE/LABELDUMP
$ !      Modify logical name definitions as required for your node.
$ !-----------------------------------------------------------------------*
$ !
$ !
$ !
$ !     Create lnm table for SETUP information ...
$       create/name_table/parent=lnm$system_table/prot=w:wred lnm$setup
$ !
$ !      Define directory for .EXE files
$      define/system setup_exe cern_root:[exe]
$ !
$ !      Allow usage of tapes interactively
$ !
$      define/system setup_enabled "INTERACTIVE"
$ !
$ !      Disallow specific users from using tapes (useful to stop troublemakers)
$ !
$ !      define/system setup_notapes "DECNET,CERNET"
$ !
$ !      Allow tapes in these batch queues
$ !
$!      define/system setup_queues "SYS$TAPES"
$!      define/system setup_queues "SYS$BATCH,SYS$TAPES"
$!      define/system setup_queues "*" ! all queues
$ !
$ !      Set up lists of available device types
$ !
$      define/system setup_tk50s "VSDD18$MKA700:"
$      define/system setup_8200s "UXDDB1$MUB0:"
$      define/system setup_exabytes setup_8200s ! Can also have aliases...
```

```
$ !
$ !      Allow tape staging
$ !
$      define/system stage_tapes "YES"
$ !      Must also ensure that DISK$STAGE exists...
```

See also the description of the FMEDIA routine and the MEDIA shell command for information on configuring generic device names in FATMEN. The generic device names used by VAXTAP must match those used by FATMEN. Thus, if the generic device type for a given medium is set to DAT, the logical name SETUP_DATS must point to a list of valid device names.

See also the installation instructions in PATCH DOC of the VAXTAP pam file.

Chapter 15: The VM FATMEN service machines

These machines should normally be running in disconnected mode and autologged at system startup time. These machines run a FORTRAN program which calls an EXEC to issue WAKEUP upon the arrival of new RDR files. These are then read in, processed, then WAKEUP is called again.

15.1 Setting up a new service machine

The service machine should have the name FM**throng**, e.g **throng** `FMALEPH`, `FMCHARM2`, `FMCPLEAR`, `FMDELPHI`, etc. [1] The account created requires a 191 disk of sufficient size to maintain the expected file catalogue information. Some 800 bytes are required per file catalogue entry, thus for a catalogue containing 15000 files, about 20 cylinders are required. A 193 disk is required for maintaining journal information. This disk should be at least 5 cylinders.

Table 15.1: Mini-disks required for FATMEN service machines

191	Disk for FATMEN RZ file, log file and `PROFILE EXEC`. The profile exec contains only one line - `'EXEC FATPROF'` 20 cylinders is normally sufficient for an initial allocation for this disk.
192	Link to the disk containing the server code and EXECs. At CERN, these are kept on the 191 disk of userid FATMEN.
193	Disk for keeping journal files. About 5 cylinders are normally required. These files are created automatically by the server following each update and are named `FATyyjjj FZhhmmss`, e.g. `FAT90001 FZ120000` for a file created at 12:00:00 on January 1st, 1990.

On CERNVM, the FATMEN service machines are monitored and controlled by the FATONE machine. Any new service machines must be registered by running the FATMEN exec on the 191 disk of this machine, as shown below.

Modifying the list of FATMEN servers

```
defaults list fatmen       /* list current list of FATMEN servers */
FMALEPH FMCNDIVR FMDELPHI FMCHARM2 FML3 FMOPAL FMCPLEAR CPDELPHI
defaults set fatmen fmep03 /* add new FATMEN server */
fatmen (edit               /* edit list of FATMEN servers */
```

15.2 Generating the FATMEN EXECs

The EXECs used by FATMEN can be generated using the following PATCHY cradle:

```
+EXE.
+ASM, 21,R=!./*BEGIN ! EXEC */
+USE,REXX.
+PAM,11.
+QUIT.
```

[1] The FATMEN servers may have any names, provided that suitable NAMES file entries are established.

The resultant file should be processed using the `SPLITFIL` exec as shown below:

```
splitfil fatrexx rexx
-----> Split FATREXX REXX A1 into pieces
Generate FM EXEC A
Generate FATSERV EXEC A
Generate FAT2CERN EXEC A
Generate FAT4WARD EXEC A
Generate FATJOURN EXEC A
Generate FATUSE EXEC A
Generate FATLOG EXEC A
Generate PURGE EXEC A
```

These EXECs should reside on the FATMEN 191 disk.

15.3 Monitoring the FATMEN servers

Privileged users may send the FATMEN servers management commands to monitor their progress, check that they are active etc. Examples of use are:

```
Jamie@Cernvm;
TELL FMDDDIVR QSPOOL
12:44:53  * MSG FROM FMDDDIVR: I HAVE 0 FILES IN MY RDR
Jamie@Cernvm;
TELL FMDDDIVR HELLO
12:44:58  * MSG FROM FMDDDIVR: HELLO AND HOW ARE YOU TODAY?
Jamie@Cernvm;
* See how much disk space CHARM2 service machine has used
TELL FMCHARM2 QDISK A
12:45:19 * MSG FROM FMCHARM2: LABEL  VDEV M STAT CYL TYPE BLKSIZE FILES  BLKS USED-(%) BLKS LEFT BLK TOTAL
12:45:19 * MSG FROM FMCHARM2: CHARM2 191  A R/W   20 3380    4096     6       35-01         2961      3000
```

Names file entries for the FATMEN Servers

Each FATMEN server should have at least two entries in their NAMES file. Those on CERNVM may also have a third, FATSERVERS, which points to the list of remote servers who should automatically receive updates from CERNVM.

The two entries that are always required are FATOWNERS and FATOPERATORS. At the present time, usernames in either of these lists are allowed to issue commands in the above table, and receive SMSGs when the servers stop.

Example of `NAMES` file for the FATMEN server

```
:nick.FATOPERATORS           :list.fatop1 fatop2
:nick.FATOP1                 :userid.console
                             :node.cernvm
:nick.FATOP2                 :userid.opsutil
                             :node.cernvm
:nick.FATOWN1                :userid.fatone
                             :node.cernvm
:nick.FATOWN2                :userid.wojcik
                             :node.frcpn11
:nick.FATOWNERS              :list.fatown1 fatown2 jamie
```

15.4 Generating the ORACLE tables

The ORACLE tables for the new group are created by editting the SQL statements below, replacing **throng** by the group name in question. Then, type `SQLPLUS user/password @file`, where `file` is the name of the file containing these statements. These SQL commands are in the patch `FATSQL` on the FATMEN pamfile.

Figure 15.1: Creation of the ORACLE table for a new throng

```
INSERT INTO Fatmen VALUES ('CERN', 'throng')
/
REM ***   SPECIFIC TABLES FOR throng
REM ***   Note: 1) CHAR(240) => Oracle V5.1
REM ***         2) TMS later substitutes Volumes_

CREATE TABLE GNames_throng ( GName CHAR(240) NOT NULL,
                             GN# NUMBER NOT NULL)
/
CREATE TABLE Files_throng ( File# NUMBER NOT NULL,
                            GN# NUMBER NOT NULL,
                            Copylevel NUMBER(2) NOT NULL,
                            Location NUMBER NOT NULL,
                            Hostname CHAR(8) NOT NULL,
                            Fullname CHAR(240) NOT NULL,
                            Hosttype CHAR(16),
                            Opersys CHAR(12),
                            Fileformat CHAR(4) NOT NULL,
                            Userformat CHAR(4),
                            Startrec# NUMBER,
                            Endrec# NUMBER,
                            Startblk# NUMBER,
                            Endblk# NUMBER,
                            Recformat CHAR(4),
                            Reclength NUMBER,
                            Blklength NUMBER,
                            Creation DATE,
                            Catalogation DATE,
                            Lastaccess DATE,
                            Active CHAR(1) NOT NULL,
                            Creatorname CHAR(8),
                            Creatoraccount CHAR(8),
                            Creatornode CHAR(8),
                            Creatorjob CHAR(8),
                            Protection NUMBER(2),
                            Userword0 NUMBER,
                            Userword1 NUMBER,
                            Userword2 NUMBER,
                            Userword3 NUMBER,
                            Userword4 NUMBER,
                            Userword5 NUMBER,
                            Userword6 NUMBER,
                            Userword7 NUMBER,
                            Userword8 NUMBER,
                            Userword9 NUMBER,
                            Sysword0 NUMBER,
                            Sysword1 NUMBER,
                            Sysword2 NUMBER,
                            Sysword3 NUMBER,
                            Sysword4 NUMBER,
                            Sysword5 NUMBER,
                            Sysword6 NUMBER,
                            Sysword7 NUMBER,
                            Sysword8 NUMBER,
                            Sysword9 NUMBER,
                            Comments CHAR(80),
                            Mediatype CHAR(1))
/
CREATE TABLE FXV_throng ( File# NUMBER NOT NULL,
                          Fileseq# NUMBER NOT NULL,
                          Vol# NUMBER NOT NULL,
                          Volseq# NUMBER)
/
CREATE TABLE Volumes_throng ( Vol# NUMBER NOT NULL,
                              VSN CHAR(6) NOT NULL,
                              VID CHAR(6) NOT NULL,
                              VIDprefix NUMBER,
                              Density NUMBER)
/
```

15.5 Generating the SQL/DS tables

The SQL/DS tables for the new group are created by editting the SQL statements below, replacing **throng** by the group name in question.

Figure 15.2: Creation of the SQL/DS tables for a new throng

```
INSERT INTO Fatmen VALUES ('CERN', 'throng')

CREATE TABLE GNames_throng ( GName CHAR(240) NOT NULL,   -
                             GN# INTEGER NOT NULL)
CREATE TABLE Files_throng ( File# INTEGER NOT NULL,      -
                            GN# INTEGER NOT NULL,        -
                            Copylevel INTEGER NOT NULL,  -
                            Location INTEGER NOT NULL,   -
                            Hostname CHAR(8) NOT NULL,   -
                            Fullname CHAR(240) NOT NULL, -
                            Hosttype CHAR(16),           -
                            Opersys CHAR(12),            -
                            Fileformat CHAR(4) NOT NULL, -
                            Userformat CHAR(4),          -
                            Startrec# INTEGER,           -
                            Endrec# INTEGER,             -
                            Startblk# INTEGER,           -
                            Endblk# INTEGER,             -
                            Recformat CHAR(4),           -
                            Reclength INTEGER,           -
                            Blklength INTEGER,           -
                            Creation DATE,               -
                            Catalogation DATE,           -
                            Lastaccess DATE,             -
                            Active CHAR(1) NOT NULL      -
                            Creatorname CHAR(8),         -
                            Creatoraccount CHAR(8),      -
                            Creatornode CHAR(8),         -
                            Creatorjob CHAR(8),          -
                            Protection INTEGER(2))
```

```
Alter table files_throng add (Userword0 INTEGER,     -
                              Userword1 INTEGER,     -
                              Userword2 INTEGER,     -
                              Userword3 INTEGER,     -
                              Userword4 INTEGER,     -
                              Userword5 INTEGER,     -
                              Userword6 INTEGER,     -
                              Userword7 INTEGER,     -
                              Userword8 INTEGER,     -
                              Userword9 INTEGER,     -
                              Sysword0 INTEGER,      -
                              Sysword1 INTEGER,      -
                              Sysword2 INTEGER,      -
                              Sysword3 INTEGER,      -
                              Sysword4 INTEGER,      -
                              Sysword5 INTEGER,      -
                              Sysword6 INTEGER,      -
                              Sysword7 INTEGER,      -
                              Sysword8 INTEGER,      -
                              Sysword9 INTEGER,      -
                              Comments CHAR(80),     -
                              Mediatype CHAR(1))
CREATE TABLE FXV_throng ( File# INTEGER NOT NULL,    -
                          Fileseq# INTEGER NOT NULL, -
                          Vol# INTEGER NOT NULL,     -
                          Volseq# INTEGER)
CREATE TABLE Volumes_throng ( Vol# INTEGER NOT NULL, -
                              VSN CHAR(6) NOT NULL,  -
                              VID CHAR(6) NOT NULL,  -
                              VIDprefix INTEGER,     -
                              Density INTEGER)
```

Table 15.2: List of commands currently supported by the FATMEN service machine

`HELLO`	Check whether server is active. If the server does not respond immediately, it means that it is either down, or processing an update. As multiple updates can be grouped together, it can happen that the server does not respond for a considerable period of time to a `HELLO` message.
`HELP`	Displays this list of commands.
`STOP`	Stop the server, but does not log it off.
`QDISK`	Issue a `QUERY DISK` command and send the output back to the originator of the message.
`QSPOOL`	Return the number of RDR files that the service machine currently has in its reader.
`DROP`	Cause the server to call the `DROP` exec to drop the specified link.
`GIME`	Cause an `EXEC GIME` to be issued.
`CLOSE`	Close the console log and spool it to the owner of the service machine.
`NEWLOG`	Cause the server to open a new log file.
`PURGE`	Purge (delete) journal files. May be sent automatically whenever a backup of the FATMEN RZ file has been taken.
`FATLOG*`	This command is issued automatically from the FATMEN user code to log FATMEN activity.
`LOGOFF`	Shutdown the service machines prior to system shutdown or other scheduled interruption, such as ORACLE backup.

Chapter 16: Restoring the RZ files from ORACLE or SQL/DS

Data can be restored from ORACLE or SQL/DS in one of two ways: either by directly recreating the FATMEN RZ file, or by sending each entry as an update in FZ format to the RDR of the virtual machine. The former is useful if the entire file is lost or corrupt, the latter for recovering individual entries or for sending the recovered data to a remote system.

16.1 Recreating the FATMEN RZ file directly

To restore the complete RZ file from ORACLE or SQL/DS, logon to the service machine of the group in question and type RESTORE. This exec currently extracts all active.[1] information from the ORACLE SQL/DS tables.

16.2 Extracting information from ORACLE or SQL/DS as FZ updates

If a small number of entries are to be restored, or the restored file is to be sent to a remote site, it may be convenient to send each entry as an FZ file to the server, with disturbing its normal operation. The program in PATCH FATO2F on the FATMEN PAM can be used for this purpose. The program reads a group name followed by a list of generic names from FORTRAN unit 5, i.e. the terminal or program stack unless overridden by a FILEDEF command. The generic names input may include a wild-card as shown below. The comments, delimited by an exclamation mark (!), should not be included in a real file and are only used here to help explain the various options. The routine FMUPDT (see on Page 57) may be used to group the updates together if required.

Example of a file for restoring from ORACLE into FZ

```
CHARM2                               ! Initialise for group CHARM2
//CERN/CHARM2/SPECIAL/R4248          ! Recover an individual entry
//CERN/CHARM2/TEST/*                 ! Recover a complete tree
```

[1] Active files are those not marked for delete. File entries marked for delete are be recovered by resetting the active flag to `Y`.

Chapter 17: The FATMEN code

17.1 Structure of the FATMEN PAM file

Cradles to generate the FATMEN FORTRAN from the FATMEN pamfile are maintained on the FATMEN pamfile. In general, it is sufficient to `+USE` the pilot patch plus the target machine, e.g. +USE,*FATLIB,IBM. In addition, the switch CZ must be selected with `+USE,CZ.` for all machines which will access a central file catalogue using the ZEBRA server.

Table 17.1: Description of the cradles in the FATMEN PAM file

*FATSQL	FORTSQL for the CERNVM service machines. The routines must be preprocessed using the PCC exec before compilation.
FATUSER	Sample program on the PAM file.
FATO2Z	Code to restore the RZ files from the ORACLE database.
FATSRV	Code for the VM service machines.
FMCDF	Command definition file (CDF) for the FATMEN shell. It must be processed with the KUIPC command before compilation.
FMKUIP	Code for the FATMEN shell. To generate the FATMEN shell, the output of FMINT and the CDF file, preprocessed using the KUIPC command, must also be included. The CDF file is in patch FMCDF on the FATMEN pam.
FATLIB	Code for the FORTRAN callable interface, maintained in FATLIB TXTLIB. It is also required by the FATMEN shell.

17.2 Installing the FATMEN software

Installation of the FATMEN software consists of two steps:

1 Installation of the FATMEN library (part of PACKLIB).

2 Creation of the FATMEN shell program.

Installation of FATMEN on CERNVM

Generating the FATLIB library

The extraction of the code is made with the following PATCHY cradle:

```
+EXE.
+USE,QCDE.
+USE,*FATLIB,IBM.
+PAM,11, R=QCDE, T=A.ZEBRA
+PAM,12.       , T=A.FATMEN
+QUIT.
```

followed by the standard procedure to generate the TXTLIB, namely VFORT asm, EDITLIB asm, TXT GEN FATLIB asm.

The FATMEN module (for the command line interface)

The extraction of the code is made with the following PATCHY cradle:

```
+EXE.
+ASM,31.
+USE,*FMKUIP,IBM.
+USE,FMCDF,T=EXE,DIV.
+PAM,11, T=A.FATMEN
+QUIT.
```

which generates two files: the standard ASM file with the FORTRAN code and the diverted ASM2 file containing input to the KUIP processor. Both the ASM code and the output from the KUIP processor must be compiled and linked to build the FATMEN module.

The following exec has been used at CERN to perform the complete operation:

```
/**/
 Address Command
 'EXEC ZPATCHY FATMEN ( CRADLE(FATMEN) ASM(FATMEN) ASM2(FMCDF CDF)'

 'KUIPC FMCDF'  /* To generate the FORTRAN equivalent of the CDF file */

 'EXEC VFORT FATMEN'
 'EXEC VFORT FMCDF'
 'EXEC VFORT ENDMODU'         /* Dummy routine to "squeeze" the module*/
 'EXEC CERNLIB FATLIB ( LINK'

 'LOAD FATMEN FMCDF ( CLEAR NOAUTO'
 'INCLUDE ENDMODU'
 'GENMOD FATMEN ( TO ENDMODU'
 Exit
/*BEGIN ENDMODU FORTRAN */
      BLOCK DATA ENDMODU
      END
```

Processing the ORACLE routines for the FATMEN server

The ORACLE FORTSQL routines can be generated using the following cradles:

```
+EXE.
+ASM, 21,R=!./*BEGIN ! FORTSQL */
+USE,ORACLE.
+USE,*FATSQL.
+PAM,11.
+QUIT.
```

The output file should then be processed by the SPLITFIL command, which will create a separate file for each routine. These can then be pre-processed by the PCC exec as follows:

```
splitfile fatsql fortsql
-----> Split FATSQL FORTSQL A1 into pieces
Generate BLANKDEK FORTSQL A
Generate FMLOGI FORTSQL A
Generate FODEL FORTSQL A
Generate FOGET FORTSQL A
```

```
Generate FOPUT FORTSQL A
FAT3@cernvm;
pcc iname=fmlogi host=fortran

ORACLE Precompiler: Version 1.2.13.10 - Production on Thu Feb 22 17:32:04 1990

Copyright (c) 1987, Oracle Corporation, California, USA.  All rights reserved.


Precompiling FMLOGI.FORTSQL
FAT3@cernvm;
vfort fmlogi
VFORT compilation options in effect are: CHARLEN(32767) FLAG(W) GOSTMT OPT(2)
MAP NOSDUMP.
File 'FMLOGI FORTRAN A1' will be processed.
VS FORTRAN VERSION 2 ENTERED.  17:32:07

**FMLOGI** END OF COMPILATION 1 ******

VS FORTRAN VERSION 2 EXITED.   17:32:10

FAT3@cernvm;
```

Generating the code for the FATMEN server

This code can be generated with the following cradle:

```
+USE,IBM,*SQL,*FATSRV.
+EXE.
+PAM,11, R=QCDE, T=A.  ZEBRA PAM *
+PAM,12,  T=A. FATMEN PAM *
+QUIT.
```

The server module can then be built as follows:

```
CERNLIB V5ORAFIX V5ORACLE FATLIB
LOAD FATSRV OSDDAT OSDU2OS (CLEAR NOMAP NODUP NOAUTO)
GENMOD FATSRV
```

Generating the FATMEN server for remote VM systems

This is performed in the same way as on CERNVM with the following exception:

– If neither ORACLE or SQL/DS are to be used, the

  ```
  +USE,*SQL
  ```

  statement should be omitted.

If ORACLE is to be used, the procedure for generating the FORTSQL routines is described on Page 177 should be followed. If SQL/DS is to be used, the procedure below should be followed.

Processing the SQLDS routines for the FATMEN server

The SQL/DS FORSQL routines can be generated using the following cradles:

```
+EXE.
+ASM, 21,R=!./*BEGIN ! FORSQL */
+USE,SQLDS.
+USE,*FATSQL.
+PAM,11.
+QUIT.
```

The output file should then be processed by the SPLITFILE command, which will create a separate file for each routine. These can then be pre-processed by the SQLIFY exec as follows:

```
splitfile fatsql forsql
-----> Split FATSQL FORSQL A1 into pieces
Generate BLANKDEK FORSQL A
Generate FMLOGI FORSQL A
Generate FODEL FORSQL A
Generate FOGET FORSQL A
Generate FOPUT FORSQL A
FAT3@Cernvm;
sqluser
SQL195 ( 01A8  R ) RR
FAT3@Cernvm;
sqlify fmlogi
ARI0717I START SQLPREP EXEC: 02/23/90 10:27:47 SET
ARI0320I THE DEFAULT DATABASE NAME IS SQLDBA.
ARI0663I FILEDEFS IN EFFECT ARE:
SYSIN    DISK     FMLOGI   FORSQL   A1
SYSPRINT DISK     FMLOGI   LISTPREP A1
SYSPUNCH DISK     FMLOGI   FORTRAN  A1
ARISQLLD DISK     ARISQLLD LOADLIB  R1
ARI0713I PREPROCESSOR ARIPRPF CALLED WITH THE FOLLOWING PARAMETERS:
........ PREP=FMLOGI, NOCHECK,ISOLATION(CS)
ARI0710E ERROR(S) OCCURRED DURING SQLPREP EXEC PROCESSING.
ARI0796I END SQLPREP EXEC: 02/23/90 10:27:48 SET
FAT3@Cernvm(00008);
FAT3@Cernvm;
vfort fmlogi
CRNVFT030I VFORT compilation options in effect are: CHARLEN(32767) FLAG(W)
GOSTMT OPT(2) MAP NOSDUMP NOPRINT.
CRNVFT000I File 'FMLOGI FORTRAN A1' will be processed.
VS FORTRAN VERSION 2 ENTERED.  10:27:55

**FMLOGI** END OF COMPILATION 1 ******

**SQLINT** END OF COMPILATION 2 ******

VS FORTRAN VERSION 2 EXITED.   10:27:55
```

```
/*******************************************************************/
/*                                                                 */
/*  Title  : Invoke SQL preprocessor                               */
/*  ======                                                         */
/*                                                                 */
/*  Format : SQLIFY   fn <ft>                                      */
/*  ======               $ASMSQLT                                  */
/*                                                                 */
/*  Author : J. Wood, Systems Group, CCD, RAL, 10/04/86            */
/*  ======                                                         */
```

```
/*                                                                       */
/*************************************************************************/
Address Command
Signal on HALT ; buf=0

Arg fn ft fm . '(' 'USERID' user pwd .

If user='' Then foruser=''
   Else Do
      user='STRIP'(user)
      foruser='USERID='user'/'pwd','
      End   /* Else Do */


ret=0

If fm = '' Then fm = 'A'

Select
   When fn='' Then Do
      Say 'Format is: SQLIFY file_name <file_type>'
      Exit 4
      End   /* Do When fn='' */
   When Abbrev('PLISQL',ft,1) Then ft='PLISQL'
   When Abbrev('ASMSQL',ft,1) Then ft='ASMSQL'
   When Abbrev('FORSQL',ft,1) Then ft='FORSQL'
   Otherwise Do
      ft='*'
      'MAKEBUF' ; buf=rc
      'LFILE' fn ft '* ( FIFO'
      n=Queued()
      reply=1
      j=0
      type.1='PLISQL'
      mode.1='A'
      typlist='/PLISQL/ASMSQL/FORSQL'
      Do i=1 To n
         Pull fn1 ft1 fm1 .
         loctyp='/'||Strip(ft1)
         If Pos(loctyp,typlist)>0 Then Do
            Do k=1 To j
               If ft1=type.k & fm1=mode.k Then Iterate i
               End   /* Do k=1 ... */
            j=j+1
            type.j=ft1
            mode.j=fm1
            End   /* Do If Pos ... */
         End   /* Do n */

      If j>1 Then Do k=1 While 1
         Say 'There is more than one file of filename' fn
         Say 'Select by number:'
         Do i=1 To j
            Say i':' fn type.i mode.i
            End   /* Do i= ... */
         Pull reply
         Select
```

```
            When ^Datatype(reply,'W') Then Iterate k
            When reply >j | reply < 1 Then Iterate k
            Otherwise Leave k
            End   /* Select */
         End   /* Do k=1 ... */

      ft=type.reply
      fm=mode.reply
      End   /* Do Otherwise */
   End   /* Select */

'STATE' fn ft fm
If rc^=0 Then Do
   Say fn ft fm 'does not exist'
   ret=rc
   Signal HALT
   End   /* Do If rc^=0 */
rc='CPUSH'('PRT')
'CP SPOOL 00E TO' Userid()

lang = left(ft,3)
if ft = 'FORSQL' then lang = 'FORTRAN'
'EXEC SQLPREP' lang 'PP(PREP='fn','foruser,
                 'NOCHECK,ISOLATION(CS))',
                 'SYSIN('fn ft fm')'
ret=rc
rc='CPOP'('PRT')
HALT:
If buf^=0 Then 'DROPBUF' buf
Exit ret
```

17.3 Tailoring the FATMEN shell

The FATMEN shell may be tailored by

1 The use of KUIP macros

2 Modifying the FATMEN CDF file, e.g. adding extra commands

KUIP macros

KUIP macros consist of files containing FATMEN or KUIP system commands. A trivial example might contain commands such as

Example macro CDOPAL KUMAC

```
MESS 'Initialise FATMEN for OPAL'
INIT OPAL
MESS 'Change directory to PROD/PASS3/FILT/P1R1039L044'
CD PROD/PASS3/FILT/P1R1039L044
MESS 'Count the number of files in this directory'
FC
```

The output of the above macro is given below:

Result of executing CDOPAL KUMAC

```
FAT3@cernvm;
fm
Type INIT to initialise FATMEN>
exec cdopal

FMINIT.  Initialisation of FATMEN package
FATMEN   1.05  900312 13.00  CERN PROGRAM LIBRARY FATMEN=Q123
         This version created on      900312  at        1600
Linked to FMOPAL               mode Z
FAOPEN : for FARZ on Unit    1 opened File CERN FATRZ
Current Working Directory = //CERN/OPAL
Current Working Directory = //CERN/OPAL/PROD/PASS3/FILT/P1R1039L044
Files:    5
FM>
```

Further details can be found in the KUIP Long writeup<BIBREF REFID=KUIP>.

Adding commands to the FATMEN shell

Extra commands maybe added to the FATMEN shell by

1. Creating a Command Definition File (CDF)
2. Merging this with the FATMEN CDF file (Patch FMCDF on the FATMEN PAM)
3. Processing the resultant CDF file using the command KUIPC
4. Compiling the FORTRAN file produced by the KUIPC command
5. Linking with the FATMEN library

For example, we could add the command XLS which would

1. Accept a generic name which could contain wild cards at any point
2. Find all matching files
3. Display the file entries

(In fact, the ls command has been modified to do just this!)

CDF file for XLS command

```
>COMMAND XLS
>GUIDENCE
Use the XLS command to perform an extended LS command.
Syntax: XLS path options
Options:
  A - list all attributes, except DZSHOW (option Z).
  C - display comment field associated with file
  F - list file attributes, such as start/end record and block
  G - list the full generic name of each file
  K - list keys associated with this file (copy level, media type, location)
  L - list logical attributes, such as FATMEN file format
      (ZEBRA exchange etc.)
  M - list media attributes, such as VSN, VID, file sequence number for tape
      files, host type and operating system for disk files.
```

```
  N - lists dataset name on disk/tape of this file
  O - list owner, node and job of creator etc.
  P - list physical attributes, such as record format etc.
  S - lists security details of this file (protection)
  T - list date and time of creation, last access etc.
  U - list user words.
  Z - dump ZEBRA bank with DZSHOW.
>ACTION FMXLSC
>PARAMETERS
+
FILE 'File or pathname' C D='CURRENT_DIRECTORY'
OPTN 'Options'          C D=' '
```

Our action routine, FMXLSC, might look like the following:

Example action routine, FMXLSC

```
      SUBROUTINE FMXLSC
      PARAMETER     (MAXFIL=10000)
      PARAMETER     (LKEYFA=10)
      DIMENSION     KEYS(LKEYFA,MAXFIL)
      CHARACTER*255 FILES(MAXFIL)
      CHARACTER*255 PATH
      CHARACTER*25  CHOPT
*
      CALL KUGETC(PATH,LPATH)
      CALL KUGETC(CHOPT,LCHOPT)
*
      CALL FMLIST(PATH(1:LPATH),FILES,KEYS,NFOUND,MAXFIL,IRC)
*
      DO 10 I=1,NFOUND
10    CALL FMSHOW(FILES(I),LBANK,CHOPT(1:LCHOPT),IRC)
*
      END
```

Chapter 18: Monitoring information

18.1 Introduction

FATMEN maintains three types of monitoring information:

1 Information recorded in the FATMEN catalogue
2 Information logged per file access
3 Information logged per session

Monitoring information in the FATMEN catalogue

The monitoring information that is stored in the FATMEN catalogued consists of the `file size`, the `date` and `time` of last access and the `number of file accesses`.

This information is stored per entry at the offsets MFSZFA, MLATFA and MUSCFA. See page 233 for a description of the bank offsets.

This information can be histogrammed with the example program FATPLOT, included in the FATMEN source file in patch EXAMPLE, deck FATPLOT and reproduced below.

Histogramming monitoring information

```
*=======================================================================
*
*  Example of using HBOOK to plot various FATMEN catalogue values.
*  This program histograms the file size, number of days since last
*  access, medium type etc.
*=======================================================================
      PARAMETER (LURCOR=200000)
      COMMON/CRZT/IXSTOR,IXDIV,IFENCE(2),LEV,LEVIN,BLVECT(LURCOR)
      DIMENSION    LQ(999),IQ(999),Q(999)
      EQUIVALENCE (IQ(1),Q(1),LQ(9)),(LQ(1),LEV)
*
      COMMON /USRLNK/LUSRK1,LUSRBK,LUSRLS
*
      COMMON /QUEST/IQUEST(100)
      CHARACTER*8   THRONG
+CDE,FATPARA.
+CDE,FATBUG.
      EXTERNAL      UROUT
*
*
*     Initialise ZEBRA
*
      CALL MZEBRA(-3)
      CALL MZSTOR(IXSTOR,'/CRZT/','Q',IFENCE,LEV,BLVECT(1),BLVECT(1),
     +            BLVECT(5000),BLVECT(LURCOR))
      CALL MZLOGL(IXSTOR,-3)

*
* *** Define user division and link area like:
*
      CALL MZDIV  (IXSTOR, IXDIV, 'USERS', 50000, LURCOR, 'L')
      CALL MZLINK (IXSTOR, '/USRLNK/', LUSRK1, LUSRLS, LUSRK1)
```

```
*
*     Units for FATMEN RZ/FZ files
*
      LUNRZ = 1
      LUNFZ = 2
      CALL GETENVF('THRONG',THRONG)
      LTH = LENOCC(THRONG)
*
*     Initialise FATMEN
*
      CALL FMINIT(IXSTOR,LUNRZ,LUNFZ,'//CERN/'//THRONG(1:LTH),IRC)
      CALL FMLOGL(0)
*
*     Initialise HBOOK
*
      CALL HLIMIT(-20000)
*
*     Book histograms
*
      CALL HBOOK1(1,'File Size (MB)',50,0.,200.,0.)
      CALL HBOOK1(2,'Number of accesses',50,0.,50.,0.)
      CALL HBOOK1(3,'Number days since last access',50,0.,300.,0.)
      CALL HBOOK1(4,'Number days since catalogued',50,0.,300.,0.)
      CALL HBOOK1(5,'Number days since created',50,0.,300.,0.)
      CALL HBOOK1(6,'Medium',5,0.,5.,0.)
      CALL HIDOPT(0,'BLAC')
*
*     Loop over all files
*
      CALL FMLOOP('//CERN/*/*',-1,UROUT,IRC)
*
*     Print and store the histograms
*
      CALL HPRINT(0)
      CALL HRPUT(0,'FATTUPLE.'//THRONG(1:LTH),'N')
*
*     Terminate cleanly
*
      CALL FMEND(IRC)
      END

      SUBROUTINE UROUT(PATH,KEYS,IRC)
+CDE,FATPARA.
      PARAMETER (LURCOR=200000)
      COMMON/CRZT/IXSTOR,IXDIV,IFENCE(2),LEV,LEVIN,BLVECT(LURCOR)
      DIMENSION    LQ(999),IQ(999),Q(999)
      EQUIVALENCE (IQ(1),Q(1),LQ(9)),(LQ(1),LEV)
      CHARACTER*(*) PATH
      PARAMETER     (LKEYFA=10)
      DIMENSION     KEYS(LKEYFA)
      DIMENSION     NDAYS(3)
      COMMON/QUEST/IQUEST(100)
      IRC   = 0
      LBANK = 0
      LP    = LENOCC(PATH)
      CALL FMGETK(PATH(1:LP),LBANK,KEYS,IRC)
*
```

```
*     Fill histograms
*
      IF(IQ(LBANK+MFSZFA).NE.0)
     +CALL HFILL(1,FLOAT(IQ(LBANK+MFSZFA)),0.,1.)
      IF(IQ(LBANK+MUSCFA).NE.0)
     +CALL HFILL(2,FLOAT(IQ(LBANK+MUSCFA)),0.,1.)
      CALL FMDAYS(PATH(1:LP),LBANK,KEYS,NDAYS,' ',IRC)
      CALL HFILL(3,FLOAT(NDAYS(3)),0.,1.)
      CALL HFILL(4,FLOAT(NDAYS(2)),0.,1.)
      CALL HFILL(5,FLOAT(NDAYS(1)),0.,1.)
      CALL HFILL(6,FLOAT(IQ(LBANK+MMTPFA)),0.,1.)
      CALL MZDROP(IXSTOR,LBANK,' ')
      END
```

18.2 Monitoring information logged per file access

For each call to `FMOPEN`, an update is sent to the `FATMEN` server to update the last access date and time, use count etc. as described above. This logging record also records additional information, which is stored in the logs of the servers.

This information consists of the following quantities:

`CHFNFA` The fully qualified name of the actual file that was accessed.

`IHOWFA` A bit pattern indicating how the file was accessed.

`JLOCFA=1`	Local disk (standard file system)
`JSFSFA=2`	VM shared file system
`JMSCFA=3`	MSCP (VAXcluster)
`JAFSFA=4`	Andrew file system
`JOSFFA=5`	OSF distributed file system
`JDFSFA=6`	DEC DFS
`JNFSFA=7`	Sun NFS
`JDECFA=8`	DECnet
`JCSPFA=9`	CSPACK server
`JFPKFA=10`	FPACK server
`JRFIFA=11`	RFIO
`JSTGFA=31`	Stage required
`JTPMFA=32`	TPMNT (=not staged)

`ITIMFA` The time in seconds spent in the routine `FMOPEN` (including the time for `STAGE` operations etc.)

In addition, the username and nodename from which the file was accessed is logged, together with the generic name and `FATMEN` keys (location code, media type and data representation). These permit further information to be extracted from the `FATMEN` catalogue if required.

18.3 Session logging

At the end of each `FATMEN` session, **provided** that a call to `FMEND` is made, a log record is written to the server. The server ignores this record (apart from counting the number that it receives, unless installed with the flag `+USE,FATLOG`.

If installed with this flag, the logging records are collected into a daily summary for subsequent processing.

The logging records consists of the following blocks:

1 Hollerith block

`KFMSYS`	FATMEN system (e.g. CERN)
`KFMGRP`	FATMEN group (e.g. DELPHI)
`KFMTIT`	Title of FATMEN source file
`KFMUSR`	User name
`KFMHST`	Host name
`KFMTYP`	Host type
`KFMOS`	Host operating system

2 MB counts

`KFMMBR`	Number of MB read
`KFMMBW`	Number of MB written
`KFZMBR`	Number of MB read with ZEBRA FZ
`KFZMBW`	Number of MB written with ZEBRA FZ
`KFMMBC`	Number of MB copied
`KFMMBN`	Number of MB copied over the network
`KFMMBQ`	Number of MB queued for copy (e.g. `CHEOPS`)

3 Dates and times (packed with `FMPKTM`)

`KFMIDQ`	Date and time of PATCHY installation job
`KFMIDS`	Date and time of start of FATMEN session
`KFMIDE`	Date and time of end of FATMEN session

4 Catalogue modifications

`KFMADD`	Number of disk entries added (`FMADDD`)
`KFMADL`	Number of links added (`FMADDL`)
`KFMADT`	Number of tape entries added (`FMADDT`)
`KFMMDR`	Number of `MKDIR` commands
`KFMRDR`	Number of `RMDIR` commands
`KFMRLN`	Number of `RMLN` commands
`KFMRTR`	Number of `RMTREE` commands
`KFMRMF`	Number of `RM` commands
`KFMCPF`	Number of `CP` commands
`KFMMVF`	Number of `MV` commands
`KFMMOD`	Number of `MODIFY` commands
`KFMTCH`	Number of `TOUCH` commands

5 File accesses

`KFMOPN`	Number of files opened, e.g. by `FMOPEN`
`KFMCLS`	Number of files closed, e.g. by `FMCLOS`
`KFMCPY`	Number of files copied, e.g. by `FMCOPY`
`KFMCPQ`	Number of files queued for copy, e.g. by `FMCOPQ`
`KFMCPN`	Number of files copied over the network, e.g. by `FMRCOP`

6 SYSREQ and TMS operations

KFMSRQ	Number of calls to SYSREQ (FMSREQ)
KFMQVL	Number of QVOL commands (FMQVOL)
KFMAVL	Number of volumes allocated (FMALLO)
KFMASP	Number of space allocations (FMGVOL or FMGVID)
KFMPOL	Number of TRANSFER commands (pool operations, e.g. FMPOOL
KFMLCK	Number of volumes locked, e.g. FMLOCK
KFMULK	Number of volumes unlocked, e.g. FMULOK
KFMDTG	Number of tags deleted, e.g. FMTAGS
KFMGTG	Number of tags obtained, e.g. FMTAGS
KFMSTG	Number of tags set, e.g. FMTAGS

7 Catalogue processing

KFMBNK	Number of banks read from the catalogue (FMRZIN)
KFMGET	Number of banks read with default selection (FMGET)
KFMGTK	Number of banks user selected (FMGETK)
KFMSHW	Number of calls to FMSHOW
KFMSCN	Number of calls to FMSCAN
KFMLOP	Number of calls to FMLOOP
KFMLDR	Number of calls to FMLDIR
KFMLFL	Number of calls to FMLFIL
KFMSRT	Number of calls to FMSORT
KFMRNK	Number of calls to FMRANK
KFMSLK	Number of calls to FMSELK
KFMMTC	Number of calls to FMATCH

Appendix A: The fatcat server

A.1 Overview of FATCAT files and directories

At CERN, a dedicated system is used to host FATMEN servers and the associated catalogues. This system is currently an IBM RS6000 with node name `fatcat`. No user accounts exist on the machine - the `/fatmen` file system is `exported` and `NFS-mounted` on all systems requiring access. At a later date, `OSF-DFS` will be used for catalogue access.

The `/fatmen` file system contains a subdirectory for each experiment, as shown below.

/fatmen directories

```
[fatcat] (209) ls -F /fatmen
fmaleph/   fmcplear/  fmna31/    fmatlas/   fmna44/    fmna52/    fmcadd/
fmdelphi/  fmnomad/   fmcharm2/  fmopal/    fmrd5/     fmchorus/  fmkeops/
fmrd6/     fml3/      fmsmc/     fmcndiv/
```

Each of these directories contains the following files or directories:

`cern.fatrz`	The FATMEN catalogue for the experiment in question.
`done`	A directory where the server places update files after processing them.
`fatserv`	The script that runs the server for this experiment.
`fatserv.log`	The log file from the server process.
`fatsrv`	The server program (in fact a link to the server program).
`fml3.names`	A names file `fm`experiment.names, e.g. fml3.names as here.
`todo`	A directory where updates are written by FATMEN clients (users) and which is scanned regularly by the server process.
`tovm`	Updates to be transferred to `CERNVM` are placed here. This directory is optional, and requires server software on the VM node.

In addition, one may also have the following configuration files:

`fatmen.loccodes`	A list of FATMEN location codes (integers) and associated text, e.g. 1=CERN Computer Centre.
`fatmen.acl`	An ACL file determining which users on which nodes may update the specified subdirectory structure.
`fatmen.updates`	A file controlling for which users and paths monitoring information is to be sent to the server.
`fatmen.accounts`	A list of account aliases.

Furthermore, one needs a subdirectory for each remote server. In the particular case of `fatcat` there are no remote servers, put the directories are called `to`node, e.g. `tovxcrna`. Provided that the names file is correctly configured, all missing subdirectories are automatically created by the server, with the exception of `tovm` and `todo`. `todo` must be given the permission `o+w` to allow all users the possibility to make catalogue updates. (With `AFS` one could use ACLs to provide better security).

A.2 Managing the servers

The FATMEN servers are started automatically at boot-time. This is done by adding the following line to the file /etc/inittab.

```
fatmen:2:wait:/etc/rc.fatmen > /dev/console 2>&1 # Start Fatmen
```

The file rc.fatmen is show below:

rc.fatmen

```
#!/bin/sh
#
#               Start FATMEN servers
#
#
if [ -x /u/jamie/bin/fatstart ]
then
        echo Start FATMEN servers ...
        su - jamie /u/jamie/bin/fatstart 2>&1
fi
```

Normally, no other operation is required. Should it be necessary to stop the servers, the following script may be used:

fatstop

```
#!/bin/ksh
#
#   Stop all FATMEN servers that are running and create a
#   file 'restart_fat' in the CWD that can be used to restart them.
#
stop=" "
run=" "
nolog=" "
noscr=" "
b="."
d=`date`
#
#   Ensure that variables are defined...
#
if [ -f restart_fat ]
   echo Remove old restart_fat file...
   then rm -i restart_fat
fi

for i in /fatmen/fm*
   do

typeset -u fatgrp
typeset -l fatman
fatpath=$i
fatgrp=`basename $i`
fatman=$fatgrp
eval $fatgrp=$fatpath;export $fatgrp
#
# and stop those servers that are running...
#
if [ -x $i/fatserv ]
   then
#
# does a log file exist?
#
   if [ -f /fatmen/$fatgrp.log ]
      then
```

```
      echo Log file exists for $fatgrp - looking for existing process
      log=$log$b$fatgrp
      pid=`cat /fatmen/$fatgrp.log | awk 'printf "%s
n",$13'`
      if (test $pid)
         then
         echo Looking for server process for $fatgrp
         if(ps -ae  | grep -s $pid )
            then
            echo FATSRV running PID = $pid
            run=$run$b$fatgrp
            echo rm /fatmen/$fatman/todo/signal.stop >> restart_fat
            echo Server stopped at $d > /fatmen/$fatman/todo/signal.stop
            else
            echo No existing server found for $fatgrp
            echo Removing old log file...
            rm   /fatmen/$fatgrp.log
            if [ -f $i/todo/signal.stop ]
               then echo signal.stop file found!
               rm $i/todo/signal.stop
               echo '(removed)'
            fi
         fi
      fi
   fi
fi

done

echo
echo Log files found for $log | tr '.' ' '
echo Servers already running for $run | tr '.' ' '
echo fatstart >> restart_fat
if [ -f restart_fat_fat ]
   then chmod +x restart_fat_fat
   echo restart reservers by typing restart_fat
fi
```

Should a server abend, it can be restarted (after curing the problem) by rerunning the `fatstart` script that is invoked at boottime.

A.3 Monitoring the servers

A simple script may be used to look for active servers and their CPU usage. The fact that each server runs its 'own' `fatsrv` module is used by this script, which reduces to three lines:

fatps

```
echo 'FATMEN server                                       Elapsed    CPU time   %CPU'
echo '============================================================================'
ps -aef -F "args,etime,time,pcpu" | grep "/fatsrv" | sort +2 -r
```

One may also inspect the log files, e.g. `tail -f fatserv.log`.

Appendix B: Catalogue recovery

On rare occasions a FATMEN catalogue may become corrupted. To recover, one can simply restore the most recent backup copy and reapply all journal files since the time of the backup. (Simply copy or move them from the `done` directory to the `todo` directory. The time stamp in the filenames will ensure that they are processed in the correct order).

Alternatively, one can attempt to 'repair' a catalogue. This may be necessary if the corruption is not noticed for some time. This can happen if a systematic check of the catalogue is not made before every backup as corruption typically affects only a few entries which may well not be accessed for some time.

Repairing a corrupted catalogue consists of the following steps:

1 Identifying which directories and/or entries are corrupted

2 Copying the catalogue skipping these directories and/or entries

3 Recovering the corrupted information from journal files or from a backup copy of the catalogue

B.1 Finding the corrupted entries

Normally, the corrupted entries are either reported by a user (e.g. I cannot access files in this directory), or are found by the server. Should the server abend, it will automatically send a mail message to the FATMEN manager. (Tailor the `fatabend` script as appropriate). The server log will then show which directory and/or entries were giving problems. Should this information be unavailable, one can find the corrupted entries by running the program `fatloop2`, as shown below (for a Unix system).

Running FATLOOP2

```
export FMLOGL=1
export FATSYS=CERN
export FATGRP=L3
fatloop2 > fatloop2.log
```

This program attempts to retrieve each catalogue entry in turn. Should a directory or catalogue entry be corrupted, then it will terminate abnormally (via ZFATAL). Thus it is good practice to run this program on a regular basis, e.g. before a periodic backup.

An example of the log is shown below.

Output of the FATLOOP2 program

```
 FATMEN system defaulted to CERN
 FATMEN group:  L3

 FMINIT.  Initialisation of FATMEN package
 FATMEN   1.81/07 930203 08:50 CERN PROGRAM LIBRARY FATMEN=Q123
          This version created on      930203  at         852
 FMLOGL. setting log level to          1
 Get: //CERN/L3/CDREMM/CC132563
 Get: //CERN/L3/CDREMM/CC132563
 Get: //CERN/L3/CDREMM/CC132563
```

```
Get: //CERN/L3/CDREMM/CC132564

...

Get: //CERN/L3/PROD/DATA/SDRETT/CC02H8G2
Get: //CERN/L3/PROD/DATA/SDRETT/CC02H8IU
Get: //CERN/L3/PROD/DATA/SDRETT/CC02HBJ6
Get: //CERN/L3/PROD/DATA/SDRETT/CC02HBGE
Get: //CERN/L3/PROD/DATA/SDRETT/CC02HBJ6
Get: //CERN/L3/PROD/DATA/SDRETT/CC02HBGE
Get: //CERN/L3/PROD/DATA/SDRETT/CC02HBLY

!!!!! ZFATAL called from MZGAR1
             called from FZIMTB

!!!!! ZFATAL reached from MZGAR1    for Case=  1

         IQUEST(11) = *********         DFE0035F   ^C_

         Current Store number =  0  (JQDIVI=19)
1ZEBRA SYSTEM Post-Mortem from ZPOSTM.

/QUEST/
            0            1         3835           47            0            1         1030           10
   1297762113   1378951200   -538967201            0           19    981191424        12309       930127
         1428          180            0         1019          644          645          646          647
          648          649          650          651          652          653          654          655
          656          657         3742         3836          203          204          205          206
     00000000     00000001     00000EFB     0000002F     00000000     00000001     00000406     0000000A
     4D5A4741     52312020     DFE0035F     00000000     00000013     3A7BCB00     00003015     000E314F
     00000594     000000B4     00000000     000003FB     00000284     00000285     00000286     00000287
     00000288     00000289     0000028A     0000028B     0000028C     0000028D     0000028E     0000028F
     00000290     00000291     00000E9E     00000EFC     000000CB     000000CC     000000CD     000000CE
```

B.2 Recovering from corrupted entries

Once the bad directories or entries have been identified, one can make a new catalogue using a special version of the RTOX/RFRX tools. Here we simply exclude directories and/or entries that have been identified as bad.

In the above example, we clearly see that there are problems in the directory //CERN/L3/PROD/DATA/SDRETT We can now use the FATMEN shell to localise the corrupted entries.

First, we use the ls command to list all files in this directory.

Using the ls command to locate corrupted entries

```
ls -w output=sdrett.log
```

The server log has told use that the first bad entry is CC02HBLY. As the catalogue entries take 163 words and the record length is 1024 words, we would expect 7 corrupted entries if an entire record was corrupt. That means that we should be able to retrieve the entry CC02HJOM correctly. This we can test using the ls command, e.g.

```
ls CC02HJ0M -a
```

However, if we attempt to retrieve the previous entry, we get the following:

```
FM> ls CC02HJ66 -a

 Directory: //CERN/L3/PROD/DATA/SDRETT

 FMRZIN. Error in RZIN for directory //CERN/L3/PROD/DATA/SDRETT
 Object:
 Key serial number =   1670 filename = CC02HJ66            data repr. =   0 media type =  2 location code =      1

 Generic filename: CC02HJ66
 Data repr.:  0 Media type: 2 Location code:      1 File serial number:   1670
 Device group: CT1
 Comment:
 Start record:          0 End record:          0
 Start block :          0 End block :          0
 File format:  user format:
 VSN:  VID:  FSEQ:    0
 Fileid:
 Created by:   ACCT:  on node:  by job:
 RECFM:   LRECL:     0 BLKSIZE:     0 Filesize:     0 Use count:     0
 File protection mask:          00000000
 Date and time of creation:          0    0
 Date and time catalogued:           0    0
 Date and time last accessed:        0    0
 User words:  00000000 00000000 00000000 00000000 00000000
              00000000 00000000 00000000 00000000 00000000
```

We can extract all of the good entries in this directory by using the `extract` or `touch` commands.

Extract good entries from a corrupted directory

```
FM>ls output=sdrett.log
FM>edit sdrett.log

(delete the lines corresponding to bad entries)
(add the command 'touch' to the beginning of every line)
(save as sdrett.kumac)

FM>update 9999 9999 0
FM>exec sdrett
```

B.3 Skipping bad directories

As described above, we can skip bad directories by modifying the program `RTOX`.

Extracting the source for RTOX

```
ypatchy /cern/new/src/car/zebra.car r2x.f :go <<!
+use,qcde.
+use,qmuix,ibmrt,*rtox.
+use,p=rz,d=rzcdes.
+use,p=rz,d=rztofz.
+use,p=rz,d=rztof1.
+use,p=rztofrfz,d=blankdek.
+exe.
+pam,11,t=c.
+quit
!
```

We then modify the routine RZTOFZ to exclude the directories that contain corrupted entries.

Modifying RTOX

```
CDECK  ID>, RZTOFZ.
      SUBROUTINE RZTOFZ(LUNFZ,CHOPT)
*
************************************************************************
*
*       Copy the CWD tree to a sequential FZ file
*       The FZ file must have been declared with FZOPEN
* Input:
*   LUNFZ   Logical unit number of the FZ sequential access file
*   CHOPT   default save only the highest cycle to LUNFZ
*           'C' save all cycles
*
* Called by <USER>
*
*  Author  : R.Brun DD/US/PD
*  Written : 14.05.86
*  Last mod: 26.06.92 JDS - protect against RZPAFF problems
*
************************************************************************
      PARAMETER      (IQDROP=25, IQMARK=26, IQCRIT=27, IQSYSX=28)
      COMMON /QUEST/ IQUEST(100)
      COMMON /ZVFAUT/IQVID(2),IQVSTA,IQVLOG,IQVTHR(2),IQVREM(2,6)
      COMMON /ZEBQ/  IQFENC(4), LQ(100)
                              DIMENSION    IQ(92),        Q(92)
                              EQUIVALENCE (IQ(1),LQ(9)), (Q(1),IQ(1))
      COMMON /MZCA/  NQSTOR,NQOFFT(16),NQOFFS(16),NQALLO(16), NQIAM
     +,              LQATAB,LQASTO,LQBTIS, LQWKTB,NQWKTB,LQWKFZ
     +,              MQKEYS(3),NQINIT,NQTSYS,NQM99,NQPERM,NQFATA,NQCASE
     +,              NQTRAC,MQTRAC(48)
                                       EQUIVALENCE (KQSP,NQOFFS(1))
      COMMON /MZCB/  JQSTOR,KQT,KQS,  JQDIVI,JQDIVR
     +,              JQKIND,JQMODE,JQDIVN,JQSHAR,JQSHR1,JQSHR2,NQRESV
     +,              LQSTOR,NQFEND,NQSTRU,NQREF,NQLINK,NQMINR,LQ2END
     +,              JQDVLL,JQDVSY,NQLOGL,NQSNAM(6)
                                       DIMENSION    IQCUR(16)
                                       EQUIVALENCE (IQCUR(1),LQSTOR)
      COMMON /MZCC/  LQPSTO,NQPFEN,NQPSTR,NQPREF,NQPLK,NQPMIN,LQP2E
     +,              JQPDVL,JQPDVS,NQPLOG,NQPNAM(6)
     +,              LQSYSS(10), LQSYSR(10), IQTDUM(22)
```

```
     +,              LQSTA(21), LQEND(20), NQDMAX(20),IQMODE(20)
     +,              IQKIND(20),IQRCU(20), IQRTO(20), IQRNO(20)
     +,              NQDINI(20),NQDWIP(20),NQDGAU(20),NQDGAF(20)
     +,              NQDPSH(20),NQDRED(20),NQDSIZ(20)
     +,              IQDN1(20), IQDN2(20),      KQFT, LQFSTA(21)
                                       DIMENSION    IQTABV(16)
                                       EQUIVALENCE (IQTABV(1),LQPSTO)
C
      COMMON /RZCL/  LTOP,LRZ0,LCDIR,LRIN,LROUT,LFREE,LUSED,LPURG
     +,              LTEMP,LCORD,LFROM
                   EQUIVALENCE (LQRS,LQSYSS(7))
C
      PARAMETER (NLPATM=100)
      COMMON /RZDIRN/NLCDIR,NLNDIR,NLPAT
      COMMON /RZDIRC/CHCDIR(NLPATM),CHNDIR(NLPATM),CHPAT(NLPATM)
      CHARACTER*16   CHNDIR,    CHCDIR,    CHPAT
C
      COMMON /RZCH/  CHWOLD,CHL
      CHARACTER*255  CHWOLD,CHL
C
      PARAMETER (KUP=5,KPW1=7,KNCH=9,KDATEC=10,KDATEM=11,KQUOTA=12,
     +          KRUSED=13,KWUSED=14,KMEGA=15,KIRIN=17,KIROUT=18,
     +          KRLOUT=19,KIP1=20,KNFREE=22,KNSD=23,KLD=24,KLB=25,
     +          KLS=26,KLK=27,KLF=28,KLC=29,KLE=30,KNKEYS=31,
     +          KNWKEY=32,KKDES=33,KNSIZE=253,KEX=6,KNMAX=100)
C
      CHARACTER*(*) CHOPT
      DIMENSION ISD(NLPATM),NSD(NLPATM),IHDIR(4)
*
*-----------------------------------------------------------------------
*
      IQUEST(1)=0
      IQ1=0
      IF(LQRS.EQ.0)GO TO 99
*
      CALL UOPTC(CHOPT,'C',IOPTC)
      NLPAT0=NLPAT
      DO 5 I=1,NLPAT0
         CHPAT(I)=CHCDIR(I)
   5  CONTINUE
      ITIME=0
      CALL RZCDIR(CHWOLD,'R')
*
*        Garbage collection in user short range divisions
*        in primary store
*
      CALL MZGARB(21,0)
*
*            Write general header
*
      IHDIR(1)=12345
      IHDIR(2)=NLPAT0
      CALL FZOUT(LUNFZ,JQPDVS,0,1,'Z',1,2,IHDIR)
      IF(IQUEST(1).NE.0)THEN
         IQ1=IQUEST(1)
         GO TO 90
      ENDIF
```

```
*
*           Set CWD to the current level
*
  10  CONTINUE
      IF(ITIME.NE.0)THEN
         CALL RZPAFF(CHPAT,NLPAT,CHL)
      IF(IQUEST(1).NE.0)THEN
         IQ1=IQUEST(1)
         NLPAT=NLPAT-1
         GO TO 20
      ENDIF
         CALL RZCDIR(CHL,' ')
      ENDIF
      ISD(NLPAT)=0
      NSD(NLPAT)=IQ(KQSP+LCDIR+KNSD)
*
*     Skip bad directories
*
      if(chl(1:lchl).eq.'//RZ/L3/PROD/DATA/SDRETT') goto 20
*
*           Write current directory
*
      CALL RZTOF1(LUNFZ,IOPTC)
      IF(IQUEST(1).NE.0)THEN
         IQ1=IQUEST(1)
         NLPAT=NLPAT-1
         GO TO 20
      ENDIF
*
*           Process possible down directories
*
  20  ISD(NLPAT)=ISD(NLPAT)+1
      IF(ISD(NLPAT).LE.NSD(NLPAT))THEN
         NLPAT=NLPAT+1
         LS=IQ(KQSP+LCDIR+KLS)
         IH=LS+7*(ISD(NLPAT-1)-1)
         CALL ZITOH(IQ(KQSP+LCDIR+IH),IHDIR,4)
         CALL UHTOC(IHDIR,4,CHPAT(NLPAT),16)
         ITIME=ITIME+1
         GO TO 10
      ELSE
         NLPAT=NLPAT-1
         IF(NLPAT.GE.NLPAT0)THEN
            LUP=LQ(KQSP+LCDIR+1)
            CALL MZDROP(JQPDVS,LCDIR,' ')
            LCDIR=LUP
            GO TO 20
         ENDIF
      ENDIF
*
*            Write final trailer
*
      NLPAT=NLPAT0
      CALL FZOUT(LUNFZ,JQPDVS,0,1,'Z',1,1,99)
      IF(IQUEST(1).NE.0)THEN
         IQ1=IQUEST(1)
         GO TO 90
```

```
      ENDIF
      LCORD=LQ(KQSP+LTOP-4)
      IF(LCORD.NE.0)THEN
         CALL MZDROP(JQPDVS,LCORD,'L')
         LCORD=0
      ENDIF
*
*           Reset CWD
*
   90 CONTINUE
      CALL RZCDIR(CHWOLD,' ')
      IF(IQ1.NE.0.AND.IQUEST(1).EQ.0)IQUEST(1)=1
*
   99 RETURN
      END
```

We now perform the following steps:

1 Convert the catalogue to sequential (FZ) format, skipping bad directories in the process
2 Convert the sequential file back to RZ format. If this is done on VM, the program `FATFROMX` should be used in preference to `RFRX`
3 Extract all good entries in the bad directories, as shown above
4 Move this file to the input directory (`/todo`) of the server

Now we have almost completed the catalogue recovery, with the exception of the corrupted entries.
When creating an RZ file on a VM system, it is important that

- The RZ file is mode 6 (update in place)
- Records are preformatted so that the file is not extended but updated in place

This is done using the program `FATFROMX`.
The `PARAMETER` NPRE should be set as follows:

Obtaining an appropriate value for `NPRE`

```
/* Calculate NPRE. This assumes a default blocksize of 4096
   bytes, both for the mini-disk and the FATMEN RZ file     */

npre = "QDISK"("191","BLKTOT") * .85
```

B.4 Restoring the corrupted entries

In the example above we are now searching for only 7 catalogue entries. Here we have the following possibilities:

1 Is there a copy of the catalogue on another node in which these entries exist?
2 Do we have a backup copy, e.g. on tape, in which these entries exist?
3 Can we find the journal files corresponding to these entries?

We can scan journal files for the entries we need as follows.

Printing the headers of the journal files

```
for i in $FML3/done
   do
      fathead $i >> l3.journal
   done
```

We then simply `grep` the file `l3.journal` for the entries in question and then moved the corresponding files to the `/todo` directory.

Appendix C: CHEOPS interface

If FATMEN has been installed with the `CHEOPS` option, users may request that files are copied between sites via the CHEOPS system. Requests will only be accepted at sites which also have the correct server configuration.

On VM/CMS systems, this requires a special service machine with username `FMCHEOPS`. This service machine receives copy requests from users on the local VM/CMS system and forwards them to the local CHEOPS server.

On Unix and VAX/VMS systems, an environmental variable/symbol `FMCHEOPS` must be defined. This points to a directory into which the copy requests will be automatically written by the FATMEN software.

The following steps are involved:

1 When a copy request is made, e.g. using the shell command `COPY` with the `TRANSPORT=CHEOPS` parameter or the `FMCOPY` subroutine with the `K` option, the FATMEN catalogue is updated with an entry corresponding to the destination file. The comment field contains the text

```
Copy request queue to CHEOPS on YYMMDD at HHMM
```

2 Once the request has been processed by the CHEOPS server, the comment is changed to

```
copy successfully queued to CHEOPS.
```

3 If there are errors processing the request, the comment becomes

```
ERROR: <cheops error message>
```

4 Finally, once the transfer has been performed, the original comment as specified by the user is restored.

5 If the actual transfer failed, the reason for the failure is instead written into the comment string of the corresponding FATMEN entry.

C.1 Building the FATMEN/CHEOPS interface on a Unix system

The following script may be used to build the FATMEN/CHEOPS interface on a Unix system.

Script to build the FATMEN/CHEOPS interface

```
ypatchy /cern/pro/src/car/zebra.car cheops2f.f :go <<!
+use,qcde.
+use,ibmrt,*cheops2f.
+exe.
+pam,11,r=qcde,t=c.
+pam,12,t=c,a. fatmen.cards
+quit
!
xlf -q extname cheops2f.f -L/cern/pro/lib -lpacklib -o cheops2f
```

This server may be run with the following script.

Running the server

```
#!/bin/sh
#
# script to run the FATMEN/CHEOPS interface.
#
t=`date`
h=`hostname`
echo
echo FATMEN/CHEOPS server starting at $t on $h
echo
FMWAKEUP=30;   export FMWAKEUP
FMLOGL=3;      export FMLOGL
FMCHEOPS=/fatmen/fmcheops; export FMCHEOPS
echo Wakeup interval is $FMWAKEUP seconds
echo
$HOME/fatmen/cheops2f
```

C.2 Building the FMCHEOPS server on a VM/CMS system

The FMCHEOPS server can be built using the following EXEC.

FMCHEOPS EXEC

```
/*       BUILD THE FMCHEOPS SERVER     */
'EXEC YPATCHY PAM1="ZEBRA CAR *" PAM2="FATMEN CARDS" ',
    'CRADLE="FATKEOPS CRADLE" ASM="FATKEOPS FORTRAN A1"'
'EXEC VFORT FATKEOPS'
'GIME TCPVMA'
'CERNLIB PACKLIB!NEW COMMTXT IBMLIB CMSLIB EDCBASE ( LINK'
'GLOBAL LOADLIB VSF2LOAD EDCLINK'
'LOAD FATKEOPS'
'GENMOD FATKEOPS'
```

The server should be set up as for other FATMEN servers, i.e. it should have a `NAMES` file defining who may issue privileged commands, as follows:

Example of a NAMES file for the FMCHEOPS server

```
:nick.FATOPERATORS
               :list.fatop1 fatop2

:nick.FATOP1
               :userid.console
               :node.cernvm

:nick.FATOP2
               :userid.opsutil
               :node.cernvm

:nick.FATOWN1
               :userid.fatone
               :node.cernvm
```

```
:nick.FATOWN2
               :userid.hrrcr
               :node.cernvm

:nick.FATOWN3
               :userid.jamie
               :node.cernvm

:nick.FATOWNERS
               :list.fatown1 fatown2 fatown3
```

The server may be run using the following EXEC.

Running the FMCHEOPS server

```
/*                      F A T _ S T A R T                    */
Address Command
   If QCONSOLE("DISCO") then nop
                        else do
                        Say "+++ Type #CP DISC to disconnect +++"
                        Say "+++ Type #CP DISC to disconnect +++"
                        Say "+++ Type #CP DISC to disconnect +++"
                        end
'IDENTIFY (STACK'
Parse pull . . locnode .

If locnode = 'CERNVM' then do

/* Start FATCAT FORTRAN Program, which calls FATSERV EXEC */
   'GLOBALV SELECT *EXEC GET PWD'
   'GLOBALV SELECT *EXEC GET USER'
   'FATKEOPS'
   end

   else do

   Say "FMCHEOPS can only be run on CERNVM..."
   end
```

Appendix D: CHEOPS interface to the FATMEN system

The CHEOPS software uses the information passed from FATMEN to stage in files pending transfer to the requested remote site. Similar, once the file has been transferred, a request is made through the FATMEN software to stage the file out to tape.

To simplify the interface between the CHEOPS software, written in C, and the FATMEN package, written primarily in FORTRAN, a special interface has been provided. This interface is intended only for use by the appropriate CHEOPS software and should not be called from user programs.

D.1 Stage in a file

```
CALL FMSTGI  (GENAME, CFQNFA,CHSNFA,ICPLFA,IMTPFA,ILOCFA,CHSTFA,CHOSFA,
              CVSNFA,CVIDFA,IVIPFA,IDENFA,IVSQFA,IFSQFA,ISRDFA,
              IERDFA,ISBLFA,IEBLFA,CRFMFA,IRLNFA,IBLNFA,CFLFFA,
              CFUTFA,ICRTFA,ICTTFA,ILATFA,CCURFA,CCIDFA,CCNIFA,
              CCJIFA,IFPRFA,ISYWFA,IUSWFA,CUCMFA, CHLINK,CHOPT,IRC)
```

GENAM	Character variable of maximum length 255 specifying the generic name of the file to be staged.
CFQNFA	Character variable of maximum length 256 specifying the fully qualified dataset name of the file to be staged.
CHSNFA	Character variable of maximum length 8 specifying the host on which the file resides (disk files).
ICPLFA	Integer variable specifying the copy level or data representation of the file to be staged.
IMTPFA	Integer variable specifying the media type code of the file to be staged.
ILOCFA	Integer variable specifying the location code of the file to be staged.
CHSTFA	Character variable of maximum length 16 specifying the host type of the system on which the file resides.
CHOSFA	Character variable of maximum length 12 specifying the operation system of the host on which the file resides.
CVSNFA	Character variable of maximum length 8 specifying the volume serial number of the tape on which the file resides.
CVIDFA	Character variable of maximum length 8 specifying the visual identifier of the volume on which the file resides.
IVIPFA	Integer VID prefix of the volume
IDENFA	Integer density
IVSQFA	Integer volume sequence number
IFSQFA	Integer file sequence number
ISRDFA	Integer start record number
IERDFA	Integer end record number
ISBLFA	Integer start block number
IEBLFA	Integer end block number
CRFMFA	Character variable of length 4 specifying the record format
IRLNFA	Integer record length in 32 bit words
IBLNFA	Integer block length in 32 bit words

CFLFFA	Character variable of length 4 specifying the FATMEN file format
CFUTFA	Character variable of length 4 specifying the user defined file type
ICRTFA	Integer containing the file creation date and time in packed format
ICTTFA	Integer containing the date and time that the file was catalogued in packed format
ILATFA	Integer containing the date and time of last access in packed format
CCURFA	Character variable of length 8 specifying the user name of the file creator
CCIDFA	Character variable of length 8 specifying the account of the creator
CCNIFA	Character variable of length 8 specifying the node on which the file was created
CCJIFA	Character variable of length 8 specifying the name of the job that created the file
IFPRFA	Integer file protection mask
ISYWFA	Integer vector of length 10 containing the system words
IUSWFA	Integer vector of length 10 containing the user words
CUCMFA	Character variable of length 80 containing the user comment
CHLINK	Character variable specifying the link should point to the staged file
CHOPT	Character options
IRC	Integer return code

Stage out a file

```
CALL FMSTGO  (GENAME, CFQNFA,CHSNFA,ICPLFA,IMTPFA,ILOCFA,CHSTFA,CHOSFA,
              CVSNFA,CVIDFA,IVIPFA,IDENFA,IVSQFA,IFSQFA,ISRDFA,
              IERDFA,ISBLFA,IEBLFA,CRFMFA,IRLNFA,IBLNFA,CFLFFA,
              CFUTFA,ICRTFA,ICTTFA,ILATFA,CCURFA,CCIDFA,CCNIFA,
              CCJIFA,IFPRFA,ISYWFA,IUSWFA,CUCMFA, CHLINK,CHOPT,IRC)
```

GENAM	Character variable of maximum length 255 specifying the generic name of the file to be staged.
CFQNFA	Character variable of maximum length 256 specifying the fully qualified dataset name of the file to be staged.
CHSNFA	Character variable of maximum length 8 specifying the host on which the file resides (disk files).
ICPLFA	Integer variable specifying the copy level or data representation of the file to be staged.
IMTPFA	Integer variable specifying the media type code of the file to be staged.
ILOCFA	Integer variable specifying the location code of the file to be staged.
CHSTFA	Character variable of maximum length 16 specifying the host type of the system on which the file resides.
CHOSFA	Character variable of maximum length 12 specifying the operation system of the host on which the file resides.
CVSNFA	Character variable of maximum length 8 specifying the volume serial number of the tape on which the file resides.
CVIDFA	Character variable of maximum length 8 specifying the visual identifier of the volume on which the file resides.
IVIPFA	Integer VID prefix of the volume
IDENFA	Integer density
IVSQFA	Integer volume sequence number

`IFSQFA`	Integer file sequence number
`ISRDFA`	Integer start record number
`IERDFA`	Integer end record number
`ISBLFA`	Integer start block number
`IEBLFA`	Integer end block number
`CRFMFA`	Character variable of length 4 specifying the record format
`IRLNFA`	Integer record length in 32 bit words
`IBLNFA`	Integer block length in 32 bit words
`CFLFFA`	Character variable of length 4 specifying the FATMEN file format
`CFUTFA`	Character variable of length 4 specifying the user defined file type
`ICRTFA`	Integer containing the file creation date and time in packed format
`ICTTFA`	Integer containing the date and time that the file was catalogued in packed format
`ILATFA`	Integer containing the date and time of last access in packed format
`CCURFA`	Character variable of length 8 specifying the user name of the file creator
`CCIDFA`	Character variable of length 8 specifying the account of the creator
`CCNIFA`	Character variable of length 8 specifying the node on which the file was created
`CCJIFA`	Character variable of length 8 specifying the name of the job that created the file
`IFPRFA`	Integer file protection mask
`ISYWFA`	Integer vector of length 10 containing the system words
`IUSWFA`	Integer vector of length 10 containing the user words
`CUCMFA`	Character variable of length 80 containing the user comment
`CHLINK`	Character variable specifying the link should point to the staged file
`CHOPT`	Character options
`IRC`	Integer return code

Appendix E: Security issues

Limited security precautions are provided in FATMEN. They are as follows:

1 There are no restrictions on access to catalogue information, other than those provided at the file system level. That is, any use may inspect any catalogue.
2 Directories may only be deleted if they are completely empty, i.e. contain no subdirectories and no files.
3 File entries may only be deleted by their creator. The creator is identified by the account field.
4 Protection may be established on directory trees, as explained below.

E.1 Restricting read access to catalogue information

If such restrictions are required, they must be performed at the file system level. For example, on VAX/VMS systems one may use access control to restrict read access to specific users or users with a given rights identifier.

E.2 Access to files catalogued in FATMEN

No restrictions on file access are provided by FATMEN itself. Volume access restrictions may be set up using the Tape Management System, if required. For disk files, the access restrictions of the file system again apply.

E.3 Access control lists

It is possible to restrict write access to the FATMEN catalogue using an ACL file. The ACL file consists of 3 fields.

1 User name
2 Node name
3 Path name

Any of the fields may be wild-carded. When an update operation is attempted, FATMEN checks the username, nodename and pathname against those defined in the file `FATMEN.ACL`. This file resides in the directory or on the mini-disk containing the catalogue for which the update is being attempted. Thus, in the case of the CDF experiment, the file will be in the directory pointed to by the symbol or variable `FMCDF`.

Lines beginning with an exclamation mark, an asterix, a hash or slash asterix treated as comments.

Example of an ACL file

```
/*              FATMEN.ACL file for CDF experiment at Fermilab         */

! ! !
! ! ! CDF FATMEN Superusers: can modify any directory
! ! !
LINGFENG * //FNAL/CDF
! ! !
```

```
! ! ! CDF FATMEN Test users: can modify the subtree //FNAL/CDF/FATMEN
! ! !
CDF_FATM * //FNAL/CDF/FATMEN
! ! !
! ! ! CDF FATMEN General users: can modify <user> tree.
! ! ! e.g. FRODO can modify //FNAL/CDF/USERS/FRODO
! ! !
<user> * //FNAL/CDF/USERS/<user>
! ! !
! ! ! CDF FATMEN General users: user ID does not match user name:
! ! !
GPYEH * //FNAL/CDF/USERS/YEH
! ! !
! ! ! CDF FATMEN TOP group Superusers:
! ! !
GPYEH * //FNAL/CDF/TOP
! ! !
! ! ! CDF FATMEN TOP DILEPTON subgroup Superusers:
! ! !
LUC * //FNAL/CDF/TOP/DILEPTON
CHIKA * //FNAL/CDF/TOP/DILEPTON
CENYI * //FNAL/CDF/TOP/DILEPTON
```

E.4 Account aliases

In addition to the above protection, catalogue entries may only be deleted by users whose account matches that of the catalogue entry. However, as shown in the following example, accounts are often not common across different systems.

```
FM> ls -o

 Directory: //CERN/CNDIV/TAPES

 Generic filename: XTRACE91
 Created by:  JAMIE    ACCT: JDS$CT   on node: CERNVM   by job: JAMIE

 Generic filename: XTRACE92
 Created by:  jamie    ACCT: JDSCT    on node: zfatal   by job: jamie-15
```

To cater for this situation, account aliasing is possible. An alias file, `FATMEN.ACCOUNTS`, consists of 2 fields.

- **1** Account name
- **2** List of aliases

Note that this scheme is not designed for large numbers of users with different accounts on each machine. It is intended for use by a small number of production managers.

Example of an account alias file

```
EMC$XV=CCA$XV,PRO$XV
CCA$XV=EMC$XV,PRO$XV
PRO$XV=CCA$XV,EMC$XV
```

E.5 Update control

By default, all file accesses that use the routine FMOPEN or the shell FIND or MAKE commands result in a logging record being sent to the servers. This record contains information which includes how the file was accessed, the time taken to access the file and updates the usage count and last access date and time.

Such updates can be controlled by a file named FATMEN.UPDATES which has the same format as the FATMEN.ACL file. If such a file exists, then the logging information will only be sent if there is an appropriate line in this file.

Example of an update control file

```
!
! Log monitoring information only for production users
!
DELPROD * //CERN/DELPHI
!
! but also for users accessing their private files
!
<user> * //CERN/DELPHI/USERS/<user>
```

Appendix F: TMS at CERN

The following information describes the TMS as installed at CERN. In general, the description is also valid for outside sites where the TMS is also installed, such as the Rutherford Appleton Laboratory in the UK, where the product originated, and IN2P3 and Saclay in France.

The Tape Management System (TMS) has been installed at CERN to manage the tapes and cartridges used at CERN. Information on some 450,000 volumes is held in a central database and may be queried and, in some cases, updated by users.

Note that FATMEN automatically issues TMS commands where necessary, and so a detailed knowledge of the TMS and the command syntax is not required. However, it may be useful to have a general overview of the system. **Note that all access to TMS information through FATMEN is via a generic name, and not a tape number.**

The TMS system runs in a special service machine and, as is the case for other service machines, does not accept commands directly but rather through a special interface. For other service machines the interface is SMSG, but TMS uses instead the SYSREQ interface (again from RAL) which handles automatic continuati of responses to queries and which allows (following modifications at CERN) queries from other CERN machines.

The SYSREQ interface is also available on remote machines, such as the central VAXcluster VXCERN, the Cray, the Siemens, SHIFT etc. This remote interface runs over TCP/IP and is contained in the CERN Program Library entry CSPACK.

A TMS query may be issued as follows:

```
SYSREQ TMS QUERY VID YZ0001
```

and similarly for other TMS commands. Note that descriptions of TMS commands in the various help files generally omit the "SYSREQ TMS", but this is always needed.

A list of all TMS commands is available via the command XFIND TMS on CERNVM. On remote systems, this information is available through the www browser. To access the XFIND information directly, use a script such as

```
# XFIND script
www "http://crnvmc/FIND" $@
```

and then use the command XFIND TMS as on CERNVM.

In either case you can then find information on a specific command of interest. To find information about a particular TMS command directly use the commands XFIND TMS command, e.g.

```
xfind tms query
```

F.1 Volume Organisation in TMS

The logical organisation of volumes within TMS follows closely the physical organisation and is therefore based around volume prefixes. For each different volume prefix a number of TMS LIBRARIES are defined, each corresponding to a possible location for volumes in that series. Most series will have the libraries/locations (where xx is the two lette VID prefix)

Examples of TMS libraries

```
xx_VAULT the location  for volumes physically in the tape  vault.  Only volumes
         in these libraries will be able to mounted on the central systems.

xx_REST  a general location for volumes not in the vault
```

In some cases there will be libraries such as xx_EXPT for volumes at the experiment - but this depends on the way an individual experiment chooses to manage its own volumes.

Certain FATMEN routines require a TMS library as an argument, such as `FMALLO`, which allocates a volume from a named pool (see below) from a given library.

F.2 Volume Ownership in TMS

As well as recording the location of a volume TMS also holds ownership details. Ownership of a volume is by ACCOUNT and OWNER. ACCOUNT is the usual CERN accounting group, gg - all Delphi volumes, for example, are owned by account XX.

The OWNER field in the TMS database is a little more special, however. All registered CERN computer users have an individual user code, or UUU, and if the owner field is a three character UUU then the volume is owned by the corresponding individual.

If the owner field is longer than 3 characters this then it cannot correspond to a UUU and so no real person owns the volume, instead the owner field is being used to group the volume into a POOL for a particular purpose. For example most experiments wish to maintain a pool of volumes which can be allocated to users on request. These volumes can be given the TMS 'owner' GG_USER and it is then easy to see which volumes remain in the pool and to choose one for allocation - perhaps with the GETPOOL command, or better, using the routine `FMALLO`. The FATMEN shell command **ALLOCATE** may also be used to allocate volumes. **Note that FATMEN only retains information on volumes by generic name. That is, there must be at least one file in the FATMEN catalogue on a given volume**. Another example would be to have the 'owner' for all DST volumes as GG_DST so users could easily list the available DSTs (and even for a given location using the QUERY CONTENTS command).

Volumes may be moved between pools using the routine `FMPOOL`.

F.3 Volume Access Rules in TMS

TMS always permits read access to all volumes by all users. **N.B. this may not be the case outside CERN!**

Write access is normally allowed to all members of the account group owning the volume, but access may be restricted by using PROTECTION GROUPS. A protection group allows access to be limited to a small number of individuals or extended to members of other groups. The QUERY VID command gives the name of the protection group governing write access to a volume and the QUERY PROTECT command can be used to see the users permitted access by a given protection group.

The owner of a volume (or an authorised user for 'pool' volumes) may globally disable write access using the LOCK command - a software equivalent of the "Do not write on this tape" sticker. This functionality is provided by the routine `FMLOCK`. Volumes may be unlocked using the routine `FMULOK`. LOCK and UNLOCK commands also exist in the FATMEN shell.

F.4 Creating TMS pools for use with FATMEN

The following TMS commands may be used to create TMS pools for use with the routine `FMALLO` or the shell `ALLOCATE` command. Note that one may also allocate tapes from these pools by issuing TMS commands directly without causing any conflict.

Note that these commands require privilege and may normally only be issued by the tape librarian.

If the tapes are already known to the TMS, the following commands should be used.

Creating a TMS pool

```
PROTECT poolname ACCOUNT gg CREATE

PROTECT poolname ACCOUNT gg GRANT AUTHORITY CONTROL ACCOUNT gg
                                                 or USERID  uuu

TRANSFER VID vid1 - vid2 FROMUSER owner TO USERID poolname
```

The `poolname` is the name of the pool, e.g. `XU_FREE`. gg and uuu have their usual meanings. The two `GRANT AUTHORITY` commands differ in that the first permits and member of the group gg to allocate tapes from this pool, where as the second enables only a specified user.

If the tape volumes are not yet known to the TMS, they must be entered using the `ENTER` command. At this time, one may also specify the `OWNER` and hence avoid the `TRANSFER` command described above. For example, if the tapes are to be in pool `XX_FREE`, then they should be entered with `USERID XX_FREE`. However, the `PROTECT` commands are still required.

F.5 TMS return codes

```
#ifndef STNDMESG_H_INCLUDED
/************************************************************/
/*                                                          */
/* Object : Definition of the  standard  messages used  by  */
/* =======  various standard parts of the TMS system.  For  */
/*          inclusion in local message repositories         */
/*                                                          */
/* Attrib : TMS system header file.                         */
/* =======                                                  */
/*                                                          */
/* Author : (c)  Jonathan  Wood,  Systems  Group, CCD, RAL, */
/* =======  5_Nov_1988                                      */
/*                                                          */
/* ANSI C Version : Martin Ellerker, IBM Support Section,   */
/* ===============  Software Group, CERN.                   */
/*                  28_April_1993                           */
/*                                                          */
/************************************************************/
/*  File:         "stndmesg.h"                              */
```

```
/*  Dependencies:  "message.h"                                */
/************************************************************/
#define STNDMESG_H_INCLUDED 1

#include "message.h"


#ifndef EXCLUDE_convert_msgs
    #define EXCLUDE_convert_msgs 1

    QMSG(1,"Input fileid is missing")
    QMSG(2,"Unable to open input file")
    QMSG(3,"\nSyntax{%d:%d}. Expected '%s' "\
"found instead '%.*s'")
    QMSG(4,"%*.*s%c")
    QMSG(5,"Unable to open output file")
    QMSG(6,"Processed %d clauses in %d records to "\
"output file '%s'")
    QMSG(7,"Processing input file %s")
    QMSG(8,"Processed %d clauses in %d records "\
"with %d errors")

    #endif

#ifndef EXCLUDE_parser_msgs
    #define EXCLUDE_parser_msgs 1

    QMSG(20,"Syntax Error")
    QMSG(21,"Found '%.*s' where expected")
    QMSG(22,"%c<%-8.8s>")
    QMSG(23,"%c'%-8.8s'")
    QMSG(24,"%c %-8.8s ")
    QMSG(25,"Parser Error: %s :OPR %-8.8s")
    QMSG(27,"Parser Error: %s :OPT %-8.8s")
    QMSG(28,"Syntax fallacy at token: %-8.8s")
    QMSG(29,"Unable to load DLCS syntax file '%s'")
    QMSG(30,"Unexpected clause type during load at block %d")

    #endif

#ifndef EXCLUDE_index_msgs
    #define EXCLUDE_index_msgs 1
    QMSG(35,"Rebuild attempted for indices on: %.*s code %d")
    QMSG(36,"Routine for index rebuild on: %.*s not provided")
    #endif
```

```
#ifndef EXCLUDE_util_msgs
    #define EXCLUDE_util_msgs 1

    QMSG(40,"Getcor failure for %d * %d bytes")
    QMSG(41,"Message buffer too small: Msg number %5.5d")
    QMSG(42,"Host command complete rc=%d")

    #endif


#ifndef EXCLUDE_comms_msgs
    #define EXCLUDE_comms_msgs 1

    QMSG(45,"**** Possible command retry: may have been "\
"executed")
    QMSG(50,"Schedule table error. Item number %d."\
"      SQLCODE %d")


    #endif

#ifndef EXCLUDE_sched_msgs
    #define EXCLUDE_sched_msgs 1

    QMSG(55,"Recurrence interval too large!")
    QMSG(56,"Permission Denied: Scheduled request %d")
    QMSG(57,"Command too long to schedule!")

    QMSG(58,"Reqnum   Userid   Command  Last_Date       "\
"Next_date       Recursion")
    QMSG(59,"-------- -------- -------- --------------- "\
"--------------- -----------------")
    QMSG(60,"%8.1d %-8.8s %-8.8s %15.6f %15.6f %8.8s %-8.8s")
    QMSG(61,"%.*s %1.8d")
    QMSG(62,"%.*s")

    #endif

#ifndef EXCLUDE_feature_msgs
    #define EXCLUDE_feature_msgs 1

    QMSG(65,"Invalid feature type: %.*s")
    QMSG(66,"Unsupported feature: %.*s")
    QMSG(67,"Feature change rejected: %.*s")
    QMSG(68,"%-8.8s")
    QMSG(69,"%.*s %-8.8s")
```

```
    #endif

#ifndef EXCLUDE_secure_msgs
    #define EXCLUDE_secure_msgs 1

    QMSG(70,"Permission denied: Command %.*s")
    QMSG(71,"Not you again?")
    QMSG(72,"You are becoming a nuisance")
    QMSG(73,"Your activity is being logged")
    QMSG(74,"Desist!  You have been warned")
    QMSG(75,"Your file has been passed to "\
"the Fraud Squad")

    QMSG(76,"Authorisation not found")
    QMSG(77,"Authorisation already exists")
    QMSG(78,"%-9.8s")
    QMSG(79,"%.*s %-8.8s")


    #endif

#ifndef EXCLUDE_dbase_msgs
    #define EXCLUDE_dbase_msgs 1

    QMSG(80,"Severe SQL error SQLCODE=%d Attempting recovery")
    QMSG(81,"Severe SQL error SQLCODE=%d Unable to recover")
    QMSG(82,"SQL Diagnostic error report:\n"\
" SQLCAID: %-8.8s SQLCABC: %d SQLCODE: %d SQLERRP: %-8.8s")
    QMSG(83," SQLERRM: %.*s")
    QMSG(84," SQLERRD: %d %d %d %d %d %d")
    QMSG(85," SQLEXT : %-5.5s")
    QMSG(86," SQLWARN: 0123456789A\n          %-11.11s")

    #endif

#ifndef EXCLUDE_main_msgs
    #define EXCLUDE_main_msgs 1

    QMSG(87,"Installation configuration complete")
    QMSG(88,"Installation configuration already complete")
    QMSG(89,"Initialisation already complete")
    QMSG(90,"**** %.*s: Sub-Command implementation deferred")
    QMSG(91,"**** %s stopped by %-8.8s on %-8.8s")
    QMSG(92,"**** %.*s: Command implementation deferred")
    QMSG(93,"**** %s initialisation complete")
```

```
    QMSG(94,"**** Command processor restart: %d")
    QMSG(95,"%s restarting at %.24s in '%s' mode")
    QMSG(96,"**** New command accepted at %.24s from "\
"%-8.8s (%-8.8s) on %-8.8s:\n%.*s")
    QMSG(97,"**** Completion code %d at %.24s")
    QMSG(98,"%s terminating at %.24s with code: %d")
    QMSG(99,"**** The %s is temporarily unavailable")

    #endif

#endif                             /* ifndef x_H_INCLUDED     */

/* UPDATES APPLIED AT 09:43:45 ON 12 AUG 1993
UUU$GG
*        STNDME$H UUU$GG   D1 REL191  08/12/93   09:40:06
*/

#ifndef APPLMESG_H_INCLUDED
/*********************************************************/
/*                                                       */
/* Object : Definition  of the  message repository for the */
/* =======  application server program.  Insert application */
/*          specific messages.                           */
/*                                                       */
/* Attrib : Application system Header file.              */
/* =======                                               */
/*                                                       */
/* Author : (c)  Jonathan  Wood,  Systems  Group, CCD, RAL, */
/* =======  5_Nov_1988                                   */
/*                                                       */
/*********************************************************/
/*  File:          "applmesg.h"                          */
/*  Included by:   "message.h"                           */
/*********************************************************/
#define APPLMESG_H_INCLUDED 1

QMSG(101,"%s")

/**********************************************************/
/*  Msgs for QTAPE command                                */
/**********************************************************/
QMSG(102,"%-8.8s %9.1ld %-8.8s %-8.8s %8.1ld %8.1ld")
QMSG(103,"Unknown volume : %-6.6s")

/**********************************************************/
/*  Msgs for QUERY VID command                            */
```

```
/***********************************************************/
QMSG(104,"Volume(s) not found")
QMSG(105," %-8.8s")
QMSG(106," %-6.6s")
QMSG(107," %-3.3s")
QMSG(108," %-1.1s")
QMSG(109," %-12.12s")
QMSG(110," %015.6f")
QMSG(111," %8.1ld")
QMSG(112," %5.1hd")
QMSG(113,"%.*s %.6s - %.6s %.*s")
QMSG(114,"%-6.6s")
QMSG(115," %8.8s")

/***********************************************************/
/*  Msgs for LIBRARY and QUERY LIBRARY commands            */
/***********************************************************/
QMSG(119,"Library not empty: %-.8s")
QMSG(120,"Duplicate library name: %-.8s")
QMSG(121,"Unknown library: %-.8s")
QMSG(122,"Library not empty: %-.8s")
QMSG(123,"Library size change invalid: %-8.8s")
QMSG(125,"Library  Czar     Group    R A L S M Target   "\
"Retain Racks  Slots    Spare")
QMSG(126,"-------- -------- -------- - - - - - -------- "\
"------ ------ -------- --------")
QMSG(127,"%-8.8s %-8.8s %-8.8s %c %c %c %c %c %-8.8s "\
"%6.1ld %6.1ld %8.1ld %8.1ld")
QMSG(128,"%.*s")
QMSG(129,"%.*s %-8.8s %-8.8s")

/***********************************************************/
/*  Msgs for RACK    command                               */
/***********************************************************/
QMSG(130,"Slot(s) in use: %-8.8s %1.1d %1.1d")
QMSG(131,"Unracked library: %-8.8s")
QMSG(132,"Unknown rack: %-8.8s %1.1d")
QMSG(133,"Invalid status change: %-8.8s %1.1d %1.1d")
QMSG(134,"Invalid split/join for rack: %-8.8s %1.1d %1.1d")
QMSG(135,"Rack too large: %-8.8s %1.1d %1.1d")
QMSG(136,"Slot was not in use: %-8.8s %8.1d")
QMSG(137,"Invalid device model: %-6.6s")

/***********************************************************/
/*  Msgs for Rule validation functions.                    */
/***********************************************************/
```

```
QMSG(139,"Permission Denied: Account %-8.8s")
QMSG(140,"Permission Denied: Volume %-6.6s")
QMSG(141,"Permission Denied: No volumes found")
QMSG(142,"Permission Denied: Library %-8.8s not found")
QMSG(143,"Permission Denied: Library %-8.8s")
QMSG(144,"Permission Denied: Group %-8.8s Account %-8.8s")

/**********************************************************/
/*  Msgs for Group Manipulation functions                 */
/**********************************************************/
QMSG(145,"Permission Denied: Volume account mismatch")
QMSG(146,"Permission Denied: Group exists")
QMSG(147,"Permission Denied: Group in use")

/**********************************************************/
/*  Msgs for Query Group(s) functions                     */
/**********************************************************/
QMSG(148,"*Account ")
QMSG(149,"*Userid  ")
QMSG(150,"*World   ")
QMSG(151,"%-9.8s")
QMSG(152,"%-7.6s")
QMSG(153,"%.*s %c %-8.8s")

/**********************************************************/
/*  Msgs for Query RACK     function                      */
/**********************************************************/
QMSG(154,"Library  Start    End      Slots    Spare    "\
"Model  Status")
QMSG(155,"-------- -------- -------- -------- -------- "\
"------ ------")
QMSG(156,"%-8.8s %8.1d %8.1d %8.1d %8.1d %-6.6s %c")
QMSG(157,"%.*s %8.1d ")

/**********************************************************/
/*  Msgs for MOVE           functions                     */
/**********************************************************/
QMSG(160,"Insufficient space: Library %-8.8s")
QMSG(161,"Move initiated for %1.1d volumes out of %1.1d tried")
QMSG(162,"Logical library: %-8.8s")
QMSG(163,"Insufficient space in library")

/**********************************************************/
/*  Msgs for the REQUEST queue manipulation functions     */
/**********************************************************/
QMSG(170,"Duplicate action request for vid: %-6.6s")
```

```
QMSG(171,"Request not found: vid %-6.6s")
QMSG(172,"Permission denied: Request on vid %-6.6s")

QMSG(173,"A C Target_1 Target_2 Requestr Jobname  Owner    "\
"Date            Jobid   ")
QMSG(174,"- - -------- -------- -------- -------- -------- "\
"--------------- --------")
QMSG(175,"%c %c %-8.8s %8.1d %-8.8s %-8.8s %-8.8s %15.6f"\
" %-8.8s")
QMSG(176,"%.*s %8.1d")

QMSG(177,"%1.1d requests of type '%.*s' found")
QMSG(178,"%-6.6s %-8.8s %8.1d %c %-8.8s %8.1d "\
"%-8.8s %15.6f %-8.8s")
QMSG(179,"%.*s %-8.8s %1.1d %-8.8s %1.1d "\
"%-6.6s %1.1d")
QMSG(180,"VID    Cur_lib  Cur_slot C Target_1 Target_2 "\
"Userid   Date            Target_3")
QMSG(181,"------ -------- -------- - -------- -------- "\
"-------- -------------- --------")

/************************************************************/
/*  Msgs for the MOUNT   command and subcommands             */
/************************************************************/
QMSG(182,"Volume locked read-only: %-6.6s")
QMSG(183,"Operating system unknown: %-8.8s")
QMSG(184,"Volume preallocated: %-6.6s")
QMSG(185,"Volume being deleted: %-6.6s")
QMSG(186,"Volume absent: %-6.6s")
QMSG(187,"Volume held: %-6.6s")
QMSG(188,"Volume unknown to TMS: %-6.6s")
QMSG(189,"Volume not in %.*s: %-6.6s")
QMSG(190,"Volume busy: %-6.6s")
QMSG(191,"Volume mount not (yet) active: %-6.6s")
QMSG(192,"Volume ring status changed: %-6.6s")
QMSG(193,"%-6.6s %-8.8s %8.1d %-6.6s %-3.3s %8.8s %-8.8s "\
"%8.1d %8.1d %1.1c")
QMSG(194,"Attempting auto-move to library: %-8.8s")
QMSG(195,"Volume is not of type '%.*s': %-6.6s")

/************************************************************/
/*  Msgs for the CLEAN   command                              */
/************************************************************/
QMSG(196,"Clean initiated for %1.1d volumes out of %1.1d "\
"tried")
```

```
/************************************************************/
/*  Msgs for the ENTER/ISSUE commands                       */
/************************************************************/
QMSG(200,"Rejected: Unknown device type: Model %-6.6s "\
"Max_den %-6.6s Length %d")
QMSG(201,"Rejected: Invalid density: %-6.6s")
QMSG(202,"Rejected: Invalid VSN range %-.6s-%-.6s")
QMSG(203,"Rejected: VID not the same as VSN")
QMSG(204,"Rejected: VID wraparound")
QMSG(205,"Rejected: Only %d slots available for %d volumes")
QMSG(206,"Rejected: %d VIDs in range already exist")
QMSG(207,"Rejected: Invalid Account(%-8.8s) or "\
"Userid(%-8.8s)")
QMSG(208,"Rejected: Congested VID range - pick another!")
QMSG(209,"Rejected: Ran out of slots with %d volumes to go")
QMSG(210,"Rejected: Unexpectedly entered no volumes!!")
QMSG(211,"Rejected: Insufficient supply of requested "\
"device type")
QMSG(212,"Rejected: Unknown allocation series: %-8.8s")
QMSG(213,"Rejected: Limit of %d volumes exceeded")
QMSG(214,"Rejected: Foreign batch not found: Model %.6s "\
"Max_den %-6.6s Length %d")

/************************************************************/
/*  Msgs for the TAG     command                            */
/************************************************************/
QMSG(220,"%d")
QMSG(221,"%.*s")

/************************************************************/
/*  Msgs for the REMOVE and RETURN commands                 */
/************************************************************/
QMSG(225,"Remove initiated for %1.1d volumes out of %1.1d "\
"tried")
QMSG(226,"Return initiated for %1.1d volumes out of %1.1d "\
"tried")
QMSG(227,"Volume is not absent: %6.6s")

/************************************************************/
/*  Msgs for the DEVTYPE and QUERY DEVTYPES commands        */
/************************************************************/
QMSG(230,"Unknown devtype: Model %-6.6s Max_den %d "\
"Length %d")
QMSG(231,"Duplicate devtype: Model %-6.6s Max_den %d "\
"Length %d")
QMSG(232,"Devtype in use: Model %-6.6s Max_den %d "\
```

```
"Length %d")

QMSG(233,"Model   Max_den   Length     Code Sort Group    "\
"P Max_strp Strp_siz")
QMSG(234,"------ -------- -------- -------- ---- -------- "\
"- -------- --------")
QMSG(235,"%6.6s %8.8s %8.1d %8.1d %4.4s %8.8s %c %8.1d %8.1d")
QMSG(236,"%.*s : %.6s %8.8s %d")

/**************************************************************/
/*  Msgs for the SUPPLY  and QUERY SUPPLY   commands          */
/**************************************************************/
QMSG(240,"Supply order requested not found")
QMSG(241,"Only %d volumes are still outstanding for this "\
"supply order")
QMSG(242,"Supply order already exists")

QMSG(243,"Reqnum   Batchnum Model    Maxden   Length S "\
"   Total     Left  Broken  Date")
QMSG(244,"-------- -------- ------ -------- -------- - "\
"-------- -------- -------- --------")
QMSG(245,"%-8.8s %8.1d %-6.6s %8.8s %8.1d %c %8.1d %8.1d "\
"%8.1d %8.8d")

QMSG(246,"%.*s %d")
QMSG(247,"%.*s %d %d")

QMSG(248,"Reqnum   Batchnum Model    Maxden   Length S "\
"   Total     Cost Userid   Manufact")
QMSG(249,"%-8.8s %8.1d %-6.6s %8.8s %8.1d %c %8.1d %8.1d "\
"%-8.8s %-8.*s")

QMSG(250,"Supply batch %d not found")
QMSG(251,"Batch %d is not in supply")
QMSG(252,"Batch %d is not HELD")
QMSG(253,"Supply batch only has %d volumes left")
QMSG(254,"Supply batch only has %d broken volumes")
QMSG(255,"Batch %d is in use")

QMSG(258,"Density value must not be greater than MaxDensity")


QMSG(260,"Volume %-6.6s not Free or Held")
QMSG(261," Volume %-6.6s")
QMSG(262,"Transfer from %-8.8s to %-8.8s and account "\
"unchanged")
```

```
QMSG(263,"Non existant new group: %-8.8s Account %-8.8s "\
"Volume %-6.6s")
QMSG(264,"No matching volumes found belonging to %-8.8s")
QMSG(265,"Permission Denied: Operand %.*s")

/**********************************************************/
/*  Msgs for the QUERY CONTENTS command.                  */
/**********************************************************/
QMSG(270,"Empty library: %-8.8s")
QMSG(271,"Library      Slot Type VID    Model  Owner    "\
"Account  S")
QMSG(272,"-------- -------- ---- ------ ------ -------- "\
"-------- -")
QMSG(273,"%-8.8s %8.1d Home %-6.6s %-6.6s %-8.8s %-8.8s %c")
QMSG(274,"%-8.8s %8.1d Move %-6.6s %-6.6s %-8.8s %-8.8s %c")
QMSG(275,"%.*s %-6.6s %8.1d")
QMSG(276,"No volumes found in given slot range")
QMSG(277,"Slots for library %-8.8s between %8.1d and %8.1d")

/**********************************************************/
/*  Msgs for the OPSYSTEM and QUERY OPSYSTEM commands     */
/**********************************************************/
QMSG(280,"Unknown Operating system: %-8.8s")
QMSG(281,"Duplicate Operating system: %-8.8s")
QMSG(282,"System   Generic  Userid   SysType  Enq Scr "\
"Location")
QMSG(283,"-------- -------- -------- -------- --- --- "\
"------------------------------")
QMSG(284,"%-8.8s %-8.8s %-8.8s %-8.8s  %c   %c  %.*s")
QMSG(285,"%.*s %.8s %.8s")
QMSG(286,"Requests outstanding from system %-8.8s")
QMSG(287,"Require location identifier")

/**********************************************************/
/*  Messages for the LOGICAL family of commands           */
/**********************************************************/
QMSG(290,"Library not logical: %-8.8s")
QMSG(291,"Duplicate cartridge: %-8.8s")
QMSG(292,"Unknown cartridge: %-8.8s")
QMSG(293,"Stripe range invalid: %1.1d-%1.1d")
QMSG(294,"Stripe(s) in use: %-8.8s %1.1d-%1.1d")

QMSG(295,"Cart     VID    Stripe_1 Stripe_2 Last_written    "\
"S Userid   Account  Library")
QMSG(296,"-------- ------ -------- -------- -------------- "\
"- -------- -------- --------")
```

```
QMSG(297,"%-8.8s %-6.6s %8.1d %8.1d %15.6f "\
"%c %-8.8s %-8.8s %-8.8s")
QMSG(298,"%.*s %-6.6s")

QMSG(299,"Cart     Model  Volumes  Userid   Account  "\
"Library      Slot Group")
QMSG(300,"-------- ------ -------- -------- -------- "\
"-------- -------- --------")
QMSG(301,"%-8.8s %-6.6s %8.1d %-8.8s %-8.8s "\
"%-8.8s %8.1d %-8.8s")
QMSG(302,"%.*s %-8.8s %-8.8s")

/*********************************************************/
/* For the QVOL command                                  */
/*********************************************************/
QMSG(310,"%-6.6s Volume Not Known. %25.1c")
QMSG(311,"%-6.6s %-8.8s %8.8ld %-6.6s %8.8s %1.1c %1.1c "\
"%1.1c %1.1c %1.1c %1.1c %8.8s %5.5ld %5.5ld %-3.3s")
QMSG(313,"Unknown Generic Operating System: %-8.8s")
QMSG(314,"Unknown Operating System: %-8.8s")
QMSG(315,"Volume unmountable")
QMSG(316,"Ignoring option: %-8.8s")

/*********************************************************/
/* For the ALLOCATE command                              */
/*********************************************************/
QMSG(320,"Can not allocate zero tapes.")
QMSG(321,"Allocated %d tape%sof device code %d")
QMSG(322,"Allocated %d tape%sfrom batch %d.")

/*********************************************************/
/* For the Transparent Tape Staging Interface.           */
/*********************************************************/
QMSG(330,"rc = %d  from TMSPURGE of vid: %6.6s")
QMSG(331,"Error flushing tape in Transparent Tape Stager. "\
"Probable stage out pending.")
QMSG(400,"Exchange initiated for vid: %6.6s")

/*********************************************************/
/*  Msgs for the CZAR/MANAGER command                    */
/*********************************************************/
QMSG(1076,"Czar/Account/Node not found")
QMSG(1077,"Czar/Account/Node already exists")
QMSG(1078,"%-9.8s")        /* Not used */
QMSG(1079,"%.*s %-8.8s %-8.8s %-8.8s")
QMSG(1080,"%.*s %-8.8s %-8.8s")
```

```
QMSG(1081,"%.*s %-8.8s")
QMSG(1121,"Unknown Czar   : %-.8s")
QMSG(1122,"Unknown Account: %-.8s")
QMSG(1125,"Manager  Account  Node     Set")
QMSG(1126,"-------- -------- -------- ---")
QMSG(1127,"%-8.8s %-8.8s %-8.8s  %c")
QMSG(1128,"%.*s")
QMSG(1129,"%.*s %-8.8s %-8.8s")
QMSG(1130,"TMS authorization granted to %-8.8s for "\
"account %-8.8s node %-8.8s")
QMSG(1131,"TMS authorization revoked from %-8.8s for "\
"account %-8.8s node %-8.8s")
QMSG(1140,"Permission Denied: Account %-8.8s Node %-8.8s")

/************************************************************/
/*  Msgs for the Frequently Used Commands algorithm         */
/************************************************************/
QMSG(1200,"**** Vector initialisation failure")
QMSG(1205,"Command    Location    Count MilliSec   "\
"    Next Previous   Syntax")
QMSG(1206,"--------   -------- -------- --------   "\
"-------- -------- --------")
QMSG(1207,"%-8.8s   %8x %8lu %8lu   %8x %8x %8x")
QMSG(1208,"%-8.8s %6lu")
QMSG(1210,"%.*s %-8.8s")

/************************************************************/
/*  Message for the QUERY FREE1ST SUBCOMMAND                */
/************************************************************/
QMSG(2001,"First free slot: %d")

/************************************************************/
/*  Msgs for the FLUSH SYSTEM command                       */
/************************************************************/
QMSG(2010,"All MOUNT requests flushed")
QMSG(2011,"No MOUNT requests found")

/************************************************************/
/*  Msgs for the extended AUTHUSER command                  */
/************************************************************/
QMSG(2076,"%-8.8s")
QMSG(2077,"%-8.8s by %-8.8s")
QMSG(2078,"%-8.8s AT %-8.8s")
QMSG(2079,"%.*s %-8.8s %-8.8s")
QMSG(2080,"%.*s FROM %-8.8s")
```

```
/***********************************************************/
/*  Msgs for the extended QUERY COMMAND command            */
/***********************************************************/
QMSG(2090,"General commands for %-8.8s at %-8.8s")
QMSG(2091,"All possible commands for %-8.8s at %-8.8s")
QMSG(2092,"Authorised commands for %-8.8s at %-8.8s")
QMSG(2095,"------- -------- --- -------- -- --------")
QMSG(2096,"--- -------- -------- --- -------- -- --------")
QMSG(2097,"---------- -------- --- -------- -- --------")
QMSG(2100,"General command %-8.8s not found")
QMSG(2101,"Command %-8.8s not found")
QMSG(2102,"Authorised command %-8.8s not found")

/***********************************************************/
/*  Msgs for the extended QUERY LIBRARY command            */
/***********************************************************/
QMSG(2125,"Library  Czar     Group    Slots    Spare    "\
"Location")
QMSG(2126,"-------- -------- -------- -------- -------- "\
"----------------------------")
QMSG(2127,"%-8.8s %-8.8s %-8.8s %8.1ld %8.1ld %.*s")
QMSG(2129,"%.*s %-8.8s %-8.8s ( LOCID")

/***********************************************************/
/*  Msgs for the extended unit address field in MOUNT      */
/***********************************************************/
QMSG(2175,"%c %c %-8.8s %8.8s %-8.8s %-8.8s %-8.8s "\
"%15.6f %-8.8s")
QMSG(2178,"%-6.6s %-8.8s %8.1d %c %-8.8s %8.8s "\
"%-8.8s %15.6f %-8.8s")

/***********************************************************/
/*  Msgs for the extended checking in the MOUNT command    */
/***********************************************************/
QMSG(2190," Volume in use on this system: %6.6s")
QMSG(2193," Request for volume %6.6s creates deadlock")

/***********************************************************/
/*  Msgs for the GENERIC and QUERY GENERIC commands        */
/***********************************************************/
QMSG(2200,"No Generic type for model %-6.6s library %-8.8s "\
"density %d system %-8.8s")
QMSG(2201,"Generic type for model %-6.6s library %-8.8s "\
"density %d system %-8.8s already exists")
QMSG(2202,"Model  Library  Density  SysType  Name  ")
QMSG(2203,"------ -------- -------- -------- ------")
```

```
QMSG(2204,"%-6.6s %-8.8s %8.8s %-8.8s %-6.6s")
QMSG(2205,"%.*s %-6.6s %-8.8s %8.8s %-8.8s")

/**********************************************************/
/*  Msgs for the FINDVIDS command                         */
/**********************************************************/
QMSG(2210,"FINDVIDS USERID %-8.8s VID %-6.6s TEXT   "\
"LIKE %.*s")
QMSG(2211,"FINDVIDS USERID %-8.8s VID %-6.6s BINARY "\
"LIKE %.*s")
QMSG(2212,"FINDVIDS USERID %-8.8s VID %-6.6s VOLINFO "\
"LIKE %.*s")
QMSG(2213,"FINDVIDS LIBRARY %-8.8s VID %-6.6s TEXT   "\
"LIKE %.*s")
QMSG(2214,"FINDVIDS LIBRARY %-8.8s VID %-6.6s BINARY "\
"LIKE %.*s")
QMSG(2215,"FINDVIDS LIBRARY %-8.8s VID %-6.6s VOLINFO "\
"LIKE %.*s")
QMSG(2216,"%-6.6s %.*s")

/**********************************************************/
/*  Msgs for the UPDATE command                           */
/**********************************************************/
QMSG(2400,"No new label information given")
QMSG(2401,"Label for VID %-6.6s: Type=%-3.3s;  VSN=%-6.6s;  "\
"Density=%8.8s")

/**********************************************************/
/*  Msgs for the DENSITY mapping command                  */
/**********************************************************/
QMSG(2500,"Model  Density  Code     P")
QMSG(2501,"------ -------- -------- -")
QMSG(2502,"%-6.6s %-8.8s %8.d %c")
QMSG(2503,"%.*s %-6.6s %-8.8s %8.d")
QMSG(2504,"Cannot update Primary Key Field")
QMSG(2505,"Mapping already exists")
QMSG(2506,"Mapping does not exist")
QMSG(2507,"Density %8.8s invalid for model %6.6s")
QMSG(2510,"Cannot delete Primary Entry")
QMSG(2511,"Cannot update code for Primary Entry")

/**********************************************************/
/*  Msgs for the LOCATION mapping command                 */
/*                                                        */
/*  Warning --> Location command relies on 2505 & 2506 above */
/**********************************************************/
```

```
QMSG(2520,"Code   Location Description")
QMSG(2521,"------ ------------------------------")
QMSG(2522,"%-6.d %.*s")
QMSG(2523,"%.*s %-6.d")

/**************************************************************/
/*  Msgs for the extended FEATURE QUERY ALERTEE command      */
/**************************************************************/
QMSG(9068,"%-8.8s at %-8.8s")
QMSG(9069,"%.*s %-8.8s AT %-8.8s")

/**************************************************************/
/*  Msgs for general Code debugging facilities               */
/**************************************************************/
QMSG(9900,"SQL Debug: %-8.8s  ->  %8.8x.%8.8x")
QMSG(9910,"PDate: %8.8d")
QMSG(9911,"PTime: %8.8d")
QMSG(9912,"DateTime: %f")
QMSG(9920,"Sqlcode: %d")
QMSG(9921,"__%s.%d")
QMSG(9922,"R%d code %d")

#endif

/* UPDATES APPLIED AT 09:36:29 ON 12 AUG 1993
UNITADDR * T. Cass      22 Mar 91 * Store MOUNT unit addr as char strin
*         APPLMESG UNITADDR D1 REL191  06/07/93   14:20:05
VOLINFO  * M. Ellerker   7 Jul 93 * Find Volinfo for user/lib
*         APPLMESG VOLINFO  D1 REL191  07/07/93   14:18:15
REORG10  * M. Ellerker  23 Jul 93 * Only allow GENERIC OR SYSTEM
*         APPLMESG REORG10  D1 REL191  08/06/93   14:30:20
DENMAP   * M. Ellerker   6 Aug 93 * Forgot this one !
*         APPLMESG DENMAP   D1 REL191  08/06/93   17:28:17
*/
```

Appendix G: Summary of the FATMEN system

Table G.1: FATMEN command line interface

Command	page	Command	page
ALLOCATE	120	ADDDISK	120
ADDTAPE	121	CD	122
CLR	122	CP	122
COPY	123	DIR	124
DUMP	124	END	124
EXIT	125	EXTRACT	125
FC	125	FIND	126
GIME	126	INIT	126
LD	127	LOCK	128
LOGLEVEL	128	LS	129
MAKE	131	MEDIA	132
MKDIR	132	MODIFY	132
		MV	133
PWD	133	QUIT	133
RM	134	RMDIR	135
RMTREE	135	SEARCH	136
TAG	143	TOUCH	144
TREE	144	UNLOCK	145
UPDATE	145	VERSION	146
VIEW	146	ZOOM	146
SET/COPYLEVEL	140	SET/MEDIATYPE	141
SET/LOCATION	141	SET/USERWORDS	142
SET/DATAREP	140	SET/SOURCE	142
SET/DESTINATION	141	SHOW/SOURCE	143
SHOW/DESTINATION	143	SHOW/DATAREP	143
SHOW/COPYLEVEL	143	SHOW/LOCATION	143
SHOW/MEDIATYPE	143	SHOW/USERWORDS	143

Table G.2: FATMEN Routine calling sequences

Function **Description**	**page**
Initialise FATMEN system `CALL FMSTRT(LUNRZ,LUNFZ,CHFAT,IRC*)`	52
Access a dataset `CALL FMFILE(LUN,GENAM,CHOPT,IRC*)`	52
Deaccess a dataset `CALL FMFEND(LUN,GENAM,CHOPT,IRC*)`	53
Add a tape file `CALL FMADDT(argument-list)`	53
Add a disk file `CALL FMADDD(argument-list)`	54
Return information on FATMEN entry `CALL FMPEEK(GENAM,IVECT,CHOPT,IRC*)`	55
Add entry to catalogue `CALL FMPOKE(GENAM,IVECT,CHOPT,IRC*)`	55
Initialise FATMEN system `CALL FMINIT(IXSTOR*,LUNRZ,LUNFZ,DBNAME,IRC*)`	56
Terminate FATMEN package `CALL FMEND(IRC*)`	56
Set logging level of FATMEN package `CALL FMLOGL(LEVEL)`	57
Control updating mode `CALL FMUPDT(MAX,NGROUP,IFLAG,IRC*)`	57
Purge old entries from catalogue `CALL FMPURG(PATH,KEYSEL,MAXSIZ,MINACC,MAXDAYS,MINCPS,LUNPUR,CHOPT,IRC*)`	58
Get information on named file `CALL FMGET(GENAM,LBANK*,KEYS*,IRC*)`	59
Get information on named file with key selection `CALL FMGETK(GENAM,LBANK*,*KEYS*,IRC*)`	59
Add entry to FATMEN catalogue `CALL FMPUT(GENAM,LBANK,IRC*)`	60
Modify existing entry `CALL FMMOD(GENAM,LBANK,IFLAG,IRC*)`	60
Create a new FATMEN bank `CALL FMBOOK(GENAM,KEYS*,LADDR*,*LSUP*,JBIAS,IRC*)`	61
Remove entry from FATMEN catalogue `CALL FMRM(GENAM,LBANK*,KEYS,IRC*)`	62
Make directory `CALL FMKDIR(CHDIR,IRC*)`	63
Set contents of FATMEN bank `CALL FMFILL(GENAM,*LBANK*,*KEYS*,CHOPT,IRC*)`	63
Insert character data into FATMEN bank `CALL FMPUTC(LBANK,STRING,ISTART,NCH,IRC*)`	64

Table G.2: FATMEN Routine calling sequences (cont.)

Function **Description**	**page**
Read character data from FATMEN bank `CALL FMGETC(LBANK,STRING*,ISTART,NCH,IRC*)`	65
Insert integer vector into FATMEN bank `CALL FMPUTV(LBANK,IVECT,ISTART,NWORDS,IRC*)`	65
Read integer vector from FATMEN bank `CALL FMGETV(LBANK,IVECT*,ISTART,NWORDS,IRC*)`	66
Insert integer value into FATMEN bank `CALL FMPUTI(LBANK,IVAL,IOFF,IRC*)`	66
Read integer value from FATMEN bank `CALL FMGETI(LBANK,IVAL*,IOFF,IRC*)`	67
Find existing dataset and associate with logical unit `CALL FMFIND(GENAM,DDNAME,*LBANK*,IRC*)`	67
Create new dataset `CALL FMMAKE(GENAM,DDNAME,*LBANK*,IRC*)`	68
Open a dataset for read or write `CALL FMOPEN(GENAM,DDNAME,*LBANK*,CHOPT,IRC*)`	68
Close file opened via FATMEN `CALL FMCLOS(GENAM,DDNAME,LBANK,CHOPT,IRC*)`	70
Copy a dataset and update the FATMEN catalogue `CALL FMCOPY(GN1,*LBANK1*,*KEYS1*,GN2,*LBANK2*,*KEYS2*,CHOPT,IRC*)`	71
Check whether generic name already exists `CALL FMEXST(GENAM,IRC*)`	72
List files in specified directory `CALL FMLS(GENAM,CHOPT,IRC*)`	73
Display contents of FATMEN bank `CALL FMSHOW(GENAM,*LBANK*,*KEYS*,CHOPT,IRC*)`	74
Count file names `CALL FMFILC(GENAM,NFILES*,IRC*)`	75
Scan FATMEN directory structure `CALL FMSCAN(PATH,NLEVEL,UROUT,IRC*)`	76
Loop through FATMEN file names `CALL FMLOOP(GENAM,UROUT,IRC*)`	77
Return directory names in directory structure `CALL FMLDIR(PATH,DIRS*,NFOUND,MAXDIR,*ICONT*,IRC*)`	78
Return file names in directory structure `CALL FMLFIL(GENAM,FILES*,KEYS*,NFOUND,MAXFIL,JCONT,IRC*)`	79
Sort file names and keys `CALL FMSORT(FILES,KEYS,NFILES,JSORT*,IRC*)`	80
Match file name against pattern `CALL FMATCH(CHFILE,MATCH,IRC*)`	81
Match multiple names against pattern `CALL FMMANY(MATCH,FILES,NFILES,NMATCH*,IRC*)`	81

Table G.2: FATMEN Routine calling sequences (cont.)

Function **Description**	**page**
Print contents of FATMEN keys vector `CALL FMPKEY(KEYS,NKEYS)`	82
Select files using the FATMEN keys `CALL FMSELK(GENAM,INKEYS,OUKEYS*,NKEYS*,MAXKEY,IRC*)`	82
Select files using the FATMEN bank information `CALL FMSELB(GENAM,INKEYS,NKEYS,UEXIT,ISEL*,IRC*)`	83
Select files using keys matrix `CALL FMSELM(GENAM,LBANK*,KEYS*,KEYM,NKEY,CHOPT,IRC*)`	84
Compare FATMEN entries `CALL FMCOMP(GENAM1,*LBANK1*,*KEYS1*,GENAM2,*LBANK2*,*KEYS2*,IRC*)`	85
Print user words and comment `CALL FMUPRT(GENAM,LBANK,KEYS,IUSER,COMM,IRC*)`	86
User selection `CALL FMUSEL(GENAM,LBANK,KEYS,IRC*)`	86
Allocate new piece of media `CALL FMALLO(MEDIA,DENS,COMPACT,LIB,POOL,LBANK,CHOPT,VSN*,VID*,IRC*)`	87
Move volumes between TMS pools `CALL FMPOOL(GENAM,LBANK,KEYS,CHPOOL,CHOPT,IRC)`	89
Obtain volume characteristics `CALL FMQVOL(GENAM,LBANK,KEYS,LIB*,MODEL*,DENS*,MNTTYP*,LABTYP*,IRC*)`	90
Obtain media information `CALL FMQMED(GENAM,*LBANK*,*KEYS*,IMEDIA*,IROBOT*,IRC*)`	91
Set default media information `CALL FMEDIA(MFMMED,MFMTYP,MFMGEN,MFMSIZ,MFMDEN,MFMMNT,MFMLAB,NMEDIA,IRC*)`	95
Get, Set or Delete TMS Tags `CALL FMTAGS(GENAM,*LBANK*,*KEYS*,*CHTAG*,CHOPT,IRC*)`	97
Declare location codes to FATMEN `CALL FMSETL(LOC,NLOC,IRC*)`	99
Declare media types to FATMEN `CALL FMSETM(MTP,NMTP,IRC*)`	101
Declare copy levels to FATMEN `CALL FMSETC(CPL,NCPL,IRC*)`	101
Declare selection matrix and options to FATMEN `CALL FMSETK(KEYM,NK,CHOPT,IRC*)`	101
Modify user words `CALL FMMODU(PATH,UFORM,UVECT,UCOMM,CHOPT,IRC*)`	107
Declare logical units to FATMEN `CALL FMSETU(LUN,NLUN,IRC*)`	107
Get a free logical unit `CALL FMGLUN(LUN*,IRC*)`	107
Free a logical unit `CALL FMFLUN(LUN*,IRC*)`	108

Table G.2: FATMEN Routine calling sequences (cont.)

Function **Description**	**page**
Verify bank contents	
`CALL FMVERI(GENAM,LBANK,KEYS,CHOPT,IRC*)`	108
Pack date and time.	
`CALL FMPKTM(IDATE,ITIME,IPACK*,IRC*)`	109
Unpack date and time.	
`CALL FMUPTM(IDATE*,ITIME*,IPACK,IRC*)`	109
Pack date and time for VAX format.	
`CALL FMPKVX(CHDATE,IDATE,ITIME,IPACK*,IRC*)`	110
Unpack date and time for VAX format.	
`CALL FMUPVX(CHDATE*,IDATE*,ITIME*,IPACK,IRC*)`	110
Return file names in specified directory	
`CALL FMFNMS(PATH,FILES*,KEYS*,NKEYS*,MAXKEY,IRC*)`	110
Return file names in directory structure	
`CALL FMLIST(PATH,FILES*,KEYS*,NFOUND,MAXFIL,IRC*)`	111
Obtain names of subdirectories in specified tree	
`CALL FMTREE(PATH,SUBDIR*,NLEVEL,NFOUND*,MAXDIR,IRC*)`	112
User routine to allocate new piece of media	
`CALL FUALLO(MEDIA,VSN*,VID*,IRC*)`	113
Create a new FATMEN bank	
`CALL FMLIFT(GENAM,KEYS*,MEDIA,CHOPT,IRC*)`	114
Get the address of a FATMEN bank	
`CALL FMLINK(GENAM,LBANK*,CHOPT,IRC*)`	114
Obtain volume characteristics	
`CALL FMQTMS(VID,LIB*,MODEL*,DENS*,MNTTYP*,LABTYP*,IRC*)`	115

Table G.3: Bank offsets and datatypes

Function	Data type	Description
Key Definitions	MKSRFA (I*4)	An integer assigned by the FATMEN system to permit differentiation between entries with the same generic name. It is not normally specified by the user, unless a particular entry is required, e.g. in an RM command.
	MKFNFA (H*20)	The part of the generic name following the last slash, such as RUN123 in the name //CERN/DELPHI/TEST/RUN123
	MKCLFA (I*4)	The 'copy level', where 0=original, 1=copy, 2=copy of a copy etc. Some experiments prefer to use the copy level to indicate the data representation of the data pointed to by that entry, where 1=IEEE, 2=IBM, 3=VAX, 4=byte-swapped IEEE (e.g. DECstation), 5=CRAY
	MKLCFA (I*4)	The location code, which is used to find the nearest or most suitable copy of a dataset. The location code corresponds to a site or LAN, because it is only at this level that a meaningful choice can be made. Thus it is only a first level selection, to filter out all opies at a given location, such as CERN, RAL etc.
	MKMTFA (I*4)	The type of medium on which the dataset resides. 1=DISK, 2=3480, 3=3420, 4=8200 (Exabyte).
Generic file description	MFQNFA (H*256)$^{*+}$	This points to the fully qualified dataset name that the file has on disk or tape. For VM files, the file name should be encoded in the form <user.addr>fn.ft where addr, if omitted, defaults to 191.
	MHSNFA (H*8)$^{*+}$	The host on which the file resides, or through which it is accessed, in the case of a tape file.
	MCPLFA (I*4)$^{*+}$	The 'copy level', where 0=original, 1=copy, 2=copy of a copy etc.
	MLOCFA (I*4)$^{*+}$	The location code, which is used to find the nearest or most suitable copy of a dataset.The location code corresponds to a site or LAN, because it is only at this level that a meaningful choice can be made. Thus it is only a first level selection, to filter out all copies at a given location, such as CERN, RAL etc.
	MMTPFA (I*4)$^{*+}$	The type of medium on which the dataset resides. 1=DISK, 2=3480, 3=3420, 4=8200 (Exabyte).
Disk description	MHSTFA (H*16)$^{+}$	The host type, e.g. IBM 3090. This field is filled in by the FATMEN system.
	MHOSFA (H*12)$^{+}$	The host operating system, such as VM/XA CMS 5.5. This field is filled in by the FATMEN system.

Table G.3: Bank offsets and datatypes (cont.)

Function	Data type	Description
Tape description	`MVSNFA (H*8)`*	The **volume serial number** or magnetically recorded label on the tape.
	`MVIDFA (H*8)`*	The **visual identifier** or contents of the sticky label on the tape.
	`MVIPFA (I*4)`	The VID prefix - internal representation.
	`MDENFA (I*4)`+	The density of the medium, such as 6250, 38000. Normally, such details are retained by the Tape Management System.
	`MVSQFA (I*4)`+	The volume sequence number, only used for multi-volume files. (Not yet supported).
	`MFSQFA (I*4)`*	The file sequence number, for multi-file tapes.
File description[1]	`MSRDFA (I*4)`	The start record in the file.
	`MERDFA (I*4)`	The end record in the file.
	`MSBLFA (I*4)`	The start block in the file.
	`MEBLFA (I*4)`	The end block in the file.
Logical description	`MFLFFA (H*4)`*+	The FATMEN defined file format. The following definitions are recognised by the FATMEN system: `FA` ZEBRA ASCII `FZ` ZEBRA native `FXN` ZEBRA native, exchange file format `FX` ZEBRA exchange `FFX` ZEBRA exchange, FORTRAN I/O `RZ` ZEBRA RZ `RX` ZEBRA RZ, exchange format `EP` EPIO `AS` normal ASCII `DA` FORTRAN direct access dataset `UN` unknown `FPT` FPACK text `FPS` FPACK sequential-access `FPD` FPACK direct-access `FPK` FPACK keyed-access `FPO` FPACK ordered `YBB` YBOS binary `YBD` YBOS direct-access
	`MFUTFA (H*4)`	The user defined file type.
Physical description	`MRFMFA (H*4)`	The record format, e.g. `VBS, FB` etc.

Table G.3: Bank offsets and datatypes (cont.)

Function	Data type	Description
	MRLNFA (I*4)	The record length, in 4-byte words.
	MBLNFA (I*4)	The block length, in 4-byte words.
	MFSZFA (I*4)	The file size in Megabytes, rounded up to the next integer. A Megabyte is defined as (1,024)**2 bytes.
	MUSCFA (I*4)	The number of times that the file has been accessed (use count). A call to FMOPEN with the option W, or to FMMAKE sets the use count to 1. A call to FMOPEN with the option R, or to FMFIND increments the count.
Date and time[2]	MCRTFA (I*4)+	Creation date and time.
	MCTTFA (I*4)+	Date and time of cataloging.
	MLATFA (I*4)	Date and time of last access (when available).
Creator information	MCURFA (H*8)+	The user name of the creator.
	MCIDFA (H*8)+	The account (UUUGG or UUU$GG) of the creator (CERNVM or VAX/VMS). For Unix systems, the NFSID is used.
	MCNIFA (H*8)+	The node on which the file was created. This is not necessarily the node from which it was catalogued.
	MCJIFA (H*8)+	The job or process name which created the file.
File protection	MFPRFA (I*4)	The file protection flag, currently unused.
System area	MSYWFA(10) (I*4)	Pointer to the first of ten system defined words (reserved for future use).
User area	MUSWFA(10) (I*4)	Pointer to the first of ten user defined words.
	MUCMFA (H*80)+	A user-defined comment.

+ Items filled in automatically by the routine **FMBOOK** or **FMLIFT** (See pages 61 and 114).

* Mandatory items (defined as **NOT NULL** in the ORACLE/SQL tables).

All fields may be overwritten by the user, but care must be exercised to ensure that the new values are coded in the correct format. In particular, for hollerith values, the entire field must first be set to blanks to avoid problems if the new value is shorter than the default setting. The routines **FMPUTC** and **FMPUTI** (see on Page 64 and on Page 66) may be used to simplify bank modification. See the routine on Page 108 for information on how to check the contents of a bank.

Notes

[1] The file description information will normally be left as zero, or set to the first and last record and block numbers. They could also be used to indicate a subset of a file.

[2] The date and time fields are stored packed, using routine FMPKTM. They can be unpacked using routine FMUPTM.

[1] The file description information will normally be left as zero, or set to the first and last record and block numbers. They could also be used to indicate a subset of a file.

[2] The date and time fields are stored packed, using routine FMPKTM. They can be unpacked using routine FMUPTM.

Disk file name

The disk file name should be entered in the format of the host operating system. For VM/CMS systems, the convention `<user.address> filename.filetype` is used. If `address` is omitted, `user` may then be any valid entry in a `GIMEUSER`, `GIMEGRP` or `GIMESYS` names file. If not found in a GIME names file, the default address 191 will be used.

An extract from a GIME names file is given below:

```
:NICK.L3DSTS  :USERID.L3MAXI
              :ADDR.222
              :LINKMODE.RR
              :FILEMODE.E
```

Using this example, files such as `<L3DSTS>RUN1.DST` would be assumed to be on the 222 disk of user L3MAXI.

For VAX/VMS systems, the full file name, including disk and directory names should be entered.

Disk files and VAXclusters

If the FATMEN software finds that the current node is in the same cluster as the node specified in the FATMEN catalogue and the disk specified is available, it will treat the entry as if the file were on the current node.

Example of valid disk file names for the cluster VXCERN

```
*     Create a valid entry in an existing FATMEN bank
      CALL UCTOH('VXCRNA',IQ(LFAT+MHSNFA),4,6)
*     or we could use the VAXcluster alias rather than current node name
      CALL UCTOH('VXCERN',IQ(LFAT+MHSNFA),4,6)
*     Create a valid disk file name
*     Note that logical name for disk is unique and consistant HEP-wide!
      CALL UCTOH('DELPHI$DK123:<DELPROD.MCDSTS>RUN567',
     +            IQ(LFAT+MFQNFA,4,35)
```

Disk files and DFS or NFS

The FATMEN system can use DFS[11] or NFS[10] to access a remote disk file, if the disk or file system on which it resides is mounted locally. Users are strongly recommended to adopt a convention for DFS and NFS names, so that the same name is used throughout a LAN. For DFS it is recommended that the disk in question have the same logical name on the system to which it is directly attached as on those where it is mounted via DFS. Thus, a disk on the central cluster with logical name VXCERN$DISK999 should be mounted remotely via DFS with the same logical name. If the FATMEN system finds a DFS device of the corresponding name on the local system, it will assume that the physical device to which it points is indeed the correct one. **N.B. the FATMEN software currently considers only the 'system' logical name table** (`LNM$SYSTEM_TABLE`)**.**

Example of mounting a remote disk via DFS

```
$!Mount the VXCERN disk DELPHI$DK123
$!The 'access-point' is defined on VXCERN and points
$!to the disk in the previous example. This will allow
$!users on the current node to access data on this disk
$!that is catalogued in FATMEN transparently.
$RUN SYS$SYSTEM:DFS$CONTROL
DFS>mount access-point DELPHI$DK123 /SYSTEM
DFS>exit
```

If the dataset name in the FATMEN entry (MFQNFA) starts with a $, then on Unix systems FATMEN will assume that the text that follows up to the first slash character is an environmental variable and attempt to translate it. An example is given below.

```
 FM> ls C01 -gn # List full generic name and corresponding dataset name

 //CERN/OPAL/DDST/PASS4/FYZ1/P19R1973/C01
 Fileid:      $OPALDATA/fyz1/p19r1973.c01

 Files:    1
 FM>quit

[zfatal] > echo $OPALDATA
/u/ws/ddst
[zfatal] >
[zfatal] >
[zfatal] > telnet shift1

...

shift1 [2] printenv OPALDATA
/user/ws/data
shift1 [3]
```

Location code

This is an integer code indicating the LAN or site where the data resides. Currently, the only supported LAN is CERN, for which the location code is defined as 1. This code is used by the FATMEN system for a first pass selection of the best copy of a data set. The correspondance between the location code and the LAN will be stored in the FATMEN database itself.

Username (MCURFA)

This is obtained through QUERY USERID on VM/CMS systems, the $GETJPI system service on VAX/VMS and the getpwuid function on Unix.

Jobname (MCJIFA)

This is obtained through JOBNAM (CERN Program Library Entry Z100) on VM/CMS systems, the $GETQUI system service on VAX/VMS and the getpwuid function on Unix.

Account (MCIDFA)

This is obtained through the QUERY ACCOUNT command on VM/CMS, the $GETUAI system service on VAX/VMS or the getuid function on Unix.

Host name (MHSTFA, MCNIFA)

This is obtained through the QUERY USERID command on VM/CMS and the $GETSYI system service on VAX/VMS.

Host type (MHSTFA)

This is obtained through the QUERY CMSLEVEL command on VM/CMS and the $GETSYI system service on VAX.

Host Operating System (MHOSFA)

This is obtained using the CP QUERY USERID command on VM/CMS systems and the $GETSYI system service on VAX/VMS machines.

Bibliography

[1] L. Lamport. LaTeX *A Document Preparation System*. Addison-Wesley, 1986.

[2] Oracle Corporation. *Introduction to SQL*. Oracle Corporation, 1988.

[3] CN/ASD Group. *KUIP – Kit for a User Interface Package,* Program library I202. CERN, January 1994.

[4] CN/ASD Group and J. Zoll/ECP. *ZEBRA Users Guide,* Program Library Q100. CERN, 1993.

[5] CERN. *CSPACK – Client Server Computing Package,* Program Library Q124, 1991.

[6] J. Shiers et al. *HEPDB - High Energy Physics Data Base (Version 0.03)*. CERN, 1992.

[7] C. Curran et al. The FATMEN Report. Technical Report DD/89/15, CERN, 1989.

[8] D. O. Williams et al. The Muscle Report. Technical Report DD/88/1, CERN, 1989.

[9] J. J. Thresher et al. *Computing at CERN in the 1990s*. CERN, 1989.

[10] Sun Microsystems. *Network File System Version 2*. Sun Microsystems, 1987.

[11] Digital Equipment Corporation. *VAX Distributed File Service*. Digital Equipment Corporation, 1987.

[12] J. P. Baud, F. Cane, F. Hemmer, E. Jagel, G. Lee, L. Robertson, B. Segal, and A. Trannoy. *SHIFT, User Guide and Reference Manual (Version 1.2)*. CERN, 1991.

[13] J. D. Shiers. *VAXTAP - VAX/VMS Tape Handling Package,* Program Library Z312. CERN, 1992.

Index