CERN Program Library Long Writeup Q180

HEPDB

Database Management Package

Reference Manual

Version 1.19

Application Software and Databases Group

Computers and Network 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:	CERN Program Library Office	(CERNLIB@CERN.CH.CH)
Editor:	Jamie Shiers /CN	(Jamie.Shiers@CERN.CH)
LaTeX expert:	Michel Goossens /CN	(Michel.Goossens@CERN.CH)

Edition – March 1995

Preliminary remarks

This Reference Guide for the HEPDB system consists of four parts:

1 An overview of the system.
2 A tutorial introduction.
3 A reference section with a description of each routine.
4 An appendix with details on system dependencies and technical implementation aspects.

Examples are in `monotype face` and strings to be input by the user are underlined. In the index the page where a routine is defined is in **bold**, page numbers where a routine is referenced are in normal type.

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

```
ftp asisftp.cern.ch
Trying 128.141.202.89...
Connected to asisftp.cern.ch.
220 asis01 FTP server (SunOS 4.1) ready.
Name (asis01:username):  anonymous
Password:  your_mailaddress
ftp> cd cernlib/doc/ps.dir
ftp> get hepdb.ps.Z
ftp> quit
```

Acknowledgements

This package is heavily based on the DBL3 and OPCAL packages, written by the L3 and OPAL collaborations respectively. The help and experience of the authors of these packages is gratefully acknowledged. Particular thanks are due to Sunanda Banerjee/L3, who has made a major contribution to this package.

The tutorial section of this manual was written by Lawrence Williams/CPLEAR.

Related Documents

This document can be complemented by the following documents:

- ZEBRA – Data Structure Management System [2].
- HBOOK – Histogramming package [3].
- PAW – Physics Analysis Workstation [4].
- CMZ – Code Management using Zebra [5].
- CSPACK – Client Server package [6].
- FATMEN – Distributed File and Tape Management System [7].
- DBL3 – The L3 database system [8].
- OPCAL – OPCAL user guide [9].

Table of Contents

List of Tables

Part I

HEPDB overview

Chapter 1: Overview

High Energy Physics experiments require the use of powerful data base systems in which to store information such as detector geometry and calibration constants. The data stored usually consist of a part which is largely time independent, for instance the parameters which describe the experimental setup, and of another part whose contents may vary with time, with different frequencies, and have therefore to be recorded repeatedly, with proper time or other validity ranges.

The latter may represent quite a large amount of data. As an example, for a LEP Detector, one has to store between tens and hundreds of megabytes of data per year of operation.

At program execution time, fast access to the data base contents is essential, and an efficient system which limits the rate of transactions between the directly addressable storage medium, where the data base resides, and the computer memory available to the user, has to be implemented.

In a multi-user, multi-computer environment, keeping up to date a centralized data base and optimizing the data flow is not a trivial matter. This can only be achieved through dedicated 'service' machines under control of a data base 'server'.

Last, but not least, the system has to be robust and safe, and equipped with facilities which minimize the inconveniences of a possible program crash or of a computer hang-up.

The package HEPDB, described in this manual, is an attempt to solve the above requirements. It has greatly benefitted from similar packages developed by other experiments at CERN and elsewhere, notably the DBL3 and OPCAL packages, written by the L3 and OPAL collaborations respectively. The DBL3 package is described in [8].

HEPDB is based on the ZEBRA system [2] and relies heavily on the Random Access I/O package RZ [10], and also MZ (memory management) [11], DZ (debug and dump) [12] and FZ (sequential I/O) [13] packages.

The choice of the ZEBRA RZ package as a subsystem on which to build HEPDB was natural, as it is used by other widely used programs such as GEANT [14], HBOOK [3], PAW [4] and FATMEN [7]. The ZEBRA package provides all of basic features required for the HEPDB system, including interchange mechanism (which in fact uses ZEBRA FZ sequential exchange mode files).

Chapter 2: HEPDB concepts and basic functionality

Built upon RZ, the HEPDB package provides facilities and utilities which are sufficiently general as to be of use to all High Energy Physics experiments. It increases the functionality of RZ in a number of areas, such as data compression, for more economical storage on the random access devices and via partitioned directories, which helps optimize performance.

2.1 Overview of the Zebra RZ package

The ZEBRA RZ [10] package uses random access files, which can reside on direct access devices or in memory. RZ files are organised in a hierarchical manner, much like the Unix file system. Thus, data is accessed or stored using a Unix-like **pathname**, plus one or more identifiers known as **keys**.

The ZEBRA RZ package was designed for use in High Energy Physics experiments, and thus is well suited to applications such as HEPDB.

In HEPDB, data is stored in one or more ZEBRA RZ files. Typically, one might use separate subdirectories for the different detector components. Again, for any given subdetector, separate subdirectories could be used for the various types of information that are to be stored for that component. Thus, one might have subdirectories such as `TEMPERATURE`, `GAS`, `ALIGN` and so on. For efficiency reasons, it is also recommended to store information with widely differing update frequencies in different directories. Thus, one might have a subdirectory per subdetector for constant or pseudo-constant information, such as the number of wires in a cell, and other directories, as shown above, for information which varies more rapidly, such as gas pressures.

2.2 Data base representation in memory

The HEPDB package uses a Zebra dynamic store (which is in fact a Fortran labelled common block). This can be any of the stores used by the application program. HEPDB operates inside two divisions in that store: a HEPDB system division, of no concern for the user, and the division where, at user's request, the data objects are dumped from the random access medium and kept in memory as long as required by the user.

In the latter division, data objects from a given data base file are stored as Zebra banks, or Zebra data structures, within a main data structure which reproduces, partially and only for the directories in use, the Node/Key structure of the data base files. As many main structures can be simultaneously handled in memory as there are data base files declared by the user; however, they are expected to all share the same division. The skeleton of NODE and KEY banks grows up, following the successive user requests, and stays permanently available in memory. The data banks, appended to the corresponding KEY banks, can be either refreshed when the time dependence of their contents requires it, or dropped when they are no longer needed by the user. They can also be marked as temporarily unwanted, so that the system can drop them in case of shortage of memory, at the cost of reaccessing them from the random access device if and when required again.

2.3 HEPDB keys

In order to minimize the disk-memory transfers and to permit a tight control of the whole system, a few general standard keys have been defined. The HEPDB package assumes that all key vectors consist of at least 10 keys. These keys are defined as shown in table 2.1.

System keys		
Key number	Meaning	Parameter
1	serial number	`IDHKSN`
2	pointer	`IDHPTR`
3	flags	`IDHFLG`
4	insertion time	`IDHINS`
5	*reserved*	
Special User keys		
6	unique source identifier	`IDHUSI`
7	software reference number	`IDHSRN`
8	*reserved*	
9	*reserved*	
10	*reserved*	
Validity range pairs (Repeated `NPAIR` times)		
11	start range 1	
12	end range 1	
Normal user keys (up to RZ limit of 100)		
`11+NPAIR*2`	first normal user key	
`12+NPAIR*2`	second normal user key	

Table 2.1: HEPDB keys

The system keys should not be touched by the user or application program. The normal user keys may contain any information.

After the standard keys come a number of key pairs. The number of pairs is a database constant. For example, the L3 collaboration uses only one key pair, the start and end times for which a database object is valid. The OPAL collaboration uses 3 key pairs - the start and end experiment, run and event number for which a database object is valid.

Additional keys, the so-called user keys, can be declared by the user (within the overall RZ [10] limit of 100 keys). Their role will become more transparent when reviewing the functionality of the storage and retrieval of data objects, described below.

2.4 Data compression

In order to minimise the usage of disk storage, data objects may be packed. Several packing mechanisms are provided - one that is particularly useful is bit packing. In this case, and provided that all elements of a data object must be of the same type (integer or real), HEPDB searches for the optimum bit-length that permits the majority of the data elements to be packed. The remaining elements are stored using two words each. Another packing algorithm, activated at user's request, stores only the data elements whose absolute values are greater than a predefined number. Packing and unpacking are automatic and not seen by the user, who always deals with unpacked data in memory. Furthermore, the user is in any case given the possibility to inhibit the packing.

In order to store data as compactly as possible, one may also store only the differences from another object in the same directory. Like packing, the storing of difference records is completely automatic and transparent to the user.

Further information on packing and the storing of difference records is given in the Appendix.

2.5 Real time optimization

The RZ package is not optimized for handling directories with large numbers of data objects; the memory resident part of the data describing the keys becomes very large and storage and retrieval become more expensive, both in CPU time and real time, when the number of data objects exceeds a few thousands. The concept of partitioned directory has been introduced to attenuate these effects. Subdirectories are automatically created when needed, with keys which permit to keep track of their contents and to quickly access the right partition. Partitioning is invisible to the user, who just has to give its characteristics when creating the directory. Subsequent storage and retrieval of data objects in a partitioned directory follow the same rules as for a normal directory, as far as the calling sequences of the user interface subroutines are concerned. It should be noted however that direct calls to RZ routines for handling such directories would be extremely hazardous and should be avoided.

To further optimize real time usage in storing data, HEPDB also provides a facility to store several objects, belonging to the same directory, in one pass. In real time, the gain is about proportional to the number of objects one stores at once.

2.6 Dictionary

The HEPDB package supports a dictionary in a special directory called DICTIONARY. In the dictionary the user can enter, for any of the directories, simple aliases for the pathnames as well as mnemonic names for the data object elements (both limited to 8 characters). The DICTIONARY directory, which does not make use of the standard HEPDB keys, is automatically generated, at the top level, when the data base is created. To start with, it has only one data object, with key 1 set to `-1`. The data object contains the number of directories in the data base, followed by 25 words for each directory, 3 integers and 22 hollerith. The integers contain the unique numeric identifier assigned to the directory, the number of characters in the pathname and the date of the last modification. The 2 first hollerith are reserved to store an alias name for the pathname (up to 8 characters) and the rest to store the pathname itself (top directory name excluded).

2.7 Aliases

Aliases are names of up to 8 characters for standard `HEPDB` directories. They provide a convenient way of accessing frequently accessed directories or directories with long names. Aliases may be manipulated by the `CDALIA` routine or the `ALIAS` command.

2.8 Mnemonic names

Mnemonic names are character names of up to 8 characters for the various elements of the Zebra banks stored in `HEPDB`.

2.9 ASCII data objects and HELP directory

ASCII files, with up to 80 characters per record, may be stored in the database. An example of the use of such a facility is for Production Control, when the logs of individual production jobs are stored in the database. Another example is related to the storage of help information in a special HELP directory generated at the top level when a data base file is created. HELP directories do not make use of the standard keys. One key is used to specify the node to which the help information is relevant.

2.10 Journaling

For restoring the integrity of a data base after a crash, as well as for establishing communication with other servers, a journaling facility has been implemented. This facility is used exclusively by the database servers. It consists of recording on an FZ sequential file all kinds of updates which affect the data base, namely the addition or deletion of directory trees, the addition or deletion of data objects, the alteration of key values, the deletion of some partitions in a partitioned directory, the definition of aliases and mnemonics in the `DICTIONARY` directory or the insertion of items in the `HELP` directory. The information is recorded in different types of FZ records, consisting of at least an FZ header, with or without a data part, depending on the kind of action, recorded as first word in the FZ header. In case of several data base files, each corresponding top directory is associated with an RZ file number. The same FZ file can however serve several top directories, if the user has decided so. It is possible to update a database on a selective number of directories from a journal file.

Two journal files are supported - a 'standard' journal file, which is used to communicate updates to other servers, and a 'special backup' file, which is retained locally. In either case, the journal files are handled automatically by the servers and are no concern of general users.

Chapter 3: Using the HEPDB Fortran interface

In most of the HEPDB calling sequences, various character options can be preset by the user and transmitted as a character string, the argument CHOPT. These will be referred to as “options” in the following paragraphs.

3.1 Initialisation

The user must initialise the ZEBRA memory management system before calling any of the HEPDB routines. This may either be done explicitly, e.g. via a call to MZEBRA, or implicitly, e.g. via a call to HLIMIT. Alternatively, the routine CDSTART may be used.

Then, for each data base file, a call to CDOPEN initializes the HEPDB/RZ control for the corresponding file. When using several data base files, the user should be careful always to give the complete pathnames for all subsequent references to the directories.

For an optimum usage of the RZ system, the user is advised to have a reasonably large allocation for the system division. This is performed automatically if the option E is specified when calling CDOPEN.

A unique numeric identifier is associated to every given top directory (every file). The HEPDB package defines by default an identifier increasing monotonically as the user makes the successive calls to CDOPEN. The choice of the identifier can however be imposed by the user, by presetting IQUEST(1), a Zebra short term communication variable, to the desired value, before calling CDOPEN. The numeric identifier assigned to each directory packed with that of the top directory, is stored in the corresponding NODE bank. This permits to record in a simple way which data objects have been used in a given program execution.

The DICTIONARY directory, initially generated under control of CDNEW, is dumped into memory at every call to CDOPEN. It can be expanded at user’s request to store mnemonics (up to 8 characters) for the elements of the data objects. These are mainly used in HEPDB interactive applications. For each directory, the mnemonics are stored in a data object of the DICTIONARY directory, with key 1 set equal to the numeric identifier of the corresponding directory. The routine which can be used for entering or retrieving the information is CDNAME. The aliases, in the DICTIONARY directory, can be stored and retrieved through the routine CDALIA.

The HELP directory, also generated automatically under control of CDNEW, at the time of creating the data base, is initially empty. The user should enter the help information as an ASCII file through the routine CDHELP. The content of the file is encoded as a computer independent single data object with the key 1 set equal to the numeric identifier of the relevant directory. The help information can be subsequently retrieved, decoded and displayed, also through the routine CDHELP.

3.2 Creation of standard directories

Before storing any data, the user has to create a directory with all specifications of the key vector. The routine CDMDIR permits the creation of HEPDB standard directories, with any number of additional user keys. The option ’P’ (partitioned) can be used to create a partitioned directory. The dictionary directory is automatically updated upon creation of a new directory. In case the directories at the intermediate nodes do not exist, CDMDIR will create them automatically with 9 integer keys. Even if a directory is created as a non-partitioned directory it may be converted at any time using the routine CDPART.

3.3 Storage of data

The user can store data from memory to disk with the routine `CDSTOR` , the data in memory being simple Zebra banks or Zebra data structures. The pathname of the directory as well as the key vector have to be supplied when storing the data (the system keys 1-5 do not however need to be filled in). In order to retain complete flexibility, the directory structure must already have been created before an object is stored.

The contents of an ASCII file can converted into a packed, machine independant format and stored in a ZEBRA bank suitable for storing in the data base through the routine `CDTEXT`. This bank can then be stored in the database using `CDSTOR` as above.

3.4 Retrieval of data

The user can retrieve Zebra data structures from any directory on a disk file into memory through the routine `CDUSE`. The data objects are selected according to the given directory pathname and to the contents of the key vector. Selection is made on the pairs of validity ranges and on the specific user keys selected, if any. If not yet done, the routine creates in memory a tree structure of NODE banks, according to the pathname elements. At the lowest node, `CDUSE` inserts the information relevant to the keys in KEY banks (one or several, created as a linear chain at the next-of-same-type link of the corresponding NODE banks) and the information relevant to the data itself, in DATA banks supported by the first link of the corresponding KEY banks. As a general rule, when `CDUSE` finds that more than one data object satisfy the validity criteria, the default action is to return the most recently inserted object that matches the selection criteria. Special actions can however be taken, by using the mask `IMASK` to select on values of system or standard user keys, such as the `source identifier` or `insertion time`, or indeed any of the normal user keys. If any element *n* of the vector `IMASK` is set, then only objects with this key element equal to the corresponding element of the input key vector will be selected. In the special case of the key pairs, setting this element will select only objects with start validity less than, or end validity greater than, the corresponding value specified in the input selection vector `ISEL`. This permits selection of all objects valid before or after a given run, event or time, depending on the key pair definitions.

Independently, the user can set a global selection on insertion time, through the routine `CDBE4`, for instance to ignore modifications to the data base past a certain date and be able to reproduce the conditions of a previous program execution.

In some special cases, one may wish to retrieve all valid objects that satisfy the specified selection criteria, and not just the most recently inserted one. In this case, or if one wishes to make multiple selections in a single call, the routine `CDUSEM` should be used.

With the option 'K' (key), `CDUSE` also permits retrieval of the keys only, without loading the data objects.

In case `CDUSE` finds that the required data objects already exist in memory, it does not bother to transfer them again from disk.

For an optimum use of the memory, and in order to minimize the disk accesses, the user should call the routine `CDFREE`, in conjunction with `CDUSE`. `CDFREE` sets a flag in the specified KEY bank to signal that the corresponding data object is no longer needed, until a subsequent call to `CDUSE` requires it again. The data object bank is not dropped immediately, but only if and when any other call to `CDUSE` need more space than currently available in memory. If the 'frozen' data object has not been dropped when it is required again, the flag is cleared and no further disk transfer will take place.

To read data objects where ASCII information has been encoded, the user can call `CDTEXT`, similar to `CDUSE`, except that the logical unit number of the file reserved to write the ASCII information has to be passed as an argument.

Simple Fortran vectors may be stored and retrieved using the routine `CDVECT`.

3.5 Error handling

All HEPDB routines have an integer return code `IRC` as last argument. If, on return, this contains the value zero then the routine was successful. A non-zero return code indicates an error. The error codes may be translated to a meaningful error message using the routine `CDERR`. An explicit message is printed out when the debug log level has been set to 1 or a higher value, through a call to `CDLOGL` (the debug level is set by default to 0 in `CDOPEN`).

Some routines return additional status information in the Zebra common block `/QUEST/IQUEST(100)`. However, the user is not required to test on the values of the `IQUEST` vector on return.

3.6 Termination

The user should always close the data base files at the end of the job, through a call to `CDEND`. This routine closes all the RZ files opened during the session, or just a single file, depending on the options specified. The user should not make reference to any HEPDB routines (other than `CDOPEN` !) after the call to `CDEND`.

3.7 Other HEPDB facilities

In this section a few other user callable subroutines of general interest are mentioned, as well as some additional facilities which extend the functionality of the HEPDB package and make it more user friendly.

3.7.1 Print and trace-back

The user can print the contents of a directory with the routine `CDLDIR`, only the keys with the option 'K' (keys), or both the data and the keys with the option 'D' (data).

The user can print the summary of data base usage through the routine `CDSTAT`. The summary consists of the numbers of calls to `CDUSE` and actual disk accesses, for each set of user key values.

Information on the data objects used in a given program execution can be obtained through the routine `CDRINFO`.

3.7.2 Time related routines

Two sets of routines are available to pack/unpack the date and time to/from one single word - the routines `CDPKTM` and `CDUPTM` for times accurate to one minute and `CDPKTS` and `CDUPTS` when one second accuracy is required.

The maximum of the start validity and the minimum of the end validity of all data base objects used in a given program execution can be obtained through the routine `CDVALID`.

The insertion time of the last inserted object in a given directory can be enquired through the routine `CDLKEY`, and the time when a directory has been last modified, through the routine `CDLMOD`.

3.7.3 Purging operations

With the automatic updating mode, deleting data objects from a given directory is rather critical, because of the possibility of deleting a master object and leaving alive its updated version(s). Therefore, any direct call to RZ deletion routines should be avoided. The operation can be taken care of by the routines `CDPURK` and `CDPURG`. The latter provides a wide range of actions through a number of character options, while `CDPURK` can be used to purge data objects with selection on user keys like `CDUSE`.

The user can delete a complete tree of directory starting from a given node, using the routine `CDDELT`. In the situation where a normal directory has been 'a posteriori' partitioned, through the use of `CDPART`, the user should call `CDDELT` to delete the original directory.

With the routine `CDPURGP`, it is possible to delete all but the last few partitions, as specified by the user, from a partitioned directory.

The user can delete all but a few directory trees from the data base using the routine `CDKEEP`.

Part II

HEPDB tutorial

Chapter 4: A tutorial guide to HEPDB

4.1 What is HEPDB ?

HEPDB is a database management system tuned to the specific requirements of High Energy Physics experiments. In particular it is well suited to the storage and retrieval of detector geometry and calibration information. However it may also be applied to other experimental areas such as book-keeping.

4.2 How does HEPDB help ?

HEPDB helps solve problems in the above areas by providing the user with a selection of FORTRAN callable subroutines for the storage and retrieval of ZEBRA data structures, FORTRAN vectors and text files. Additionally there are facilities for help manipulation, aliases for directory names and mnemonic names for individual elements of the stored data.

Where appropriate these facilities are also provided through a KUIP based interactive interface.

4.3 Explanation of terms

HEPDB is based upon the ZEBRA RZ package. Thus data is accessed or stored using a UNIX-like pathname, plus one or more identifiers known as keys.

For example in the use of HEPDB to store and retrieve calibration constants for some experimental subdetector the aforementioned pathname would be the name of the sub-detector and the keys would provide the selection of a particular calibration.

4.4 Using HEPDB - an example scenario

Suppose some high energy physics institute has an experiment involving the use of some large detector within which the knowledge of geometrical and calibration parameters is of great importance in order to aid the understanding of systematical errors.

The database to contain these parameters giving a full description of our apparatus should therefore try to satisfy the following criteria:

1 Allow fast and simple access to large amounts of stored data
2 Provide portability between different platforms
3 Offer the capability to handle frequent updates
4 Provide an interactive interface for issuing commands to the database as well as FORTRAN library routines
5 Provide files which can easily be accessed from users own applications

4.4.1 Further problem definition

We shall now examine generally the type of procedures that must be followed to allow the implementation of a set of databases under HEPDB bearing in mind that the new databases must be built from scratch and that we want to cause the minimum possible disruption to the existing software that an experiment may already be using.

Suppose now that some experimental group who currently use the RZ file system for keeping data regarding the geometric, calibration and auxiliary data of a detector decide that a change to the HEPDB database management system would give them more flexibility with their data.

The following pages of this tutorial aim to give a general overview in the steps involved with such a conversion , stopping also to describe some other interesting features of the package.

The general plan for the conversion would be as follows:

1 Set up a server for HEPDB
2 Create the new destination databases
3 Convert existing databases to the HEPDB format including any changes to the existing structure (keys, directories etc.).
4 Construct a compatibility mode interface to existing experimental software.

4.5 The database servers

Fundamental to the operation of HEPDB is the concept of the database server. Although other modes of operation exist, the normal way of updating a HEPDB database is via a dedicated database server. The client, or user, requests changes to the database using the HEPDB Fortran or KUIP interface. These changes are not applied directly. Instead they are written into a queue, which might simply be a directory or the VM spool. Thus although the database files may be read by an unlimited number of simulatenous users, they are only ever written by a single process - the database server.

4.6 Placement of servers

The first step in setting up a new HEPDB server is that of making a home for it to reside in. In the case of UNIX and VMS platforms this merely involves the creation of a directory with some arbitrary name. In the following paragraphs we shall assume the server has been placed in the directory `hepdb/expdb`.[1] Within this directory one should now create the subdirectories which are used to hold various data essential to the server. The first of these should be named `logs` (this is where the server logs are written), the second `queue` (this directory holds new updates from HEPDB clients) [2], and the third `save` (the place the server saves updates after processing). In addition, a further directory named `bad` is used to store files that cannot be successfully processed.

In the case of VM platforms the following procedure should be adopted. There should be one account per experiment set aside for the server, this should be given a name consisting of the prefix CD followed by the experiment name. For example the CPLEAR collaborations server is named CDCPLEAR. (Note that in the following examples when refering to a VM implementation our example account will be CDEXPDB)

Within this account your 191 addressed disk (this will hold the actual database files) should be allocated some 20 cylinders of space, your 192 addressed disk (the link disk to the HEPDB 191 code) should be given 2 cylinders and finally your 193 mode disk (the journal disk for transactions) should be given 5 cylinders of space.

[1] The exact server setup at CERN is described on page 103.
[2] The above names are in fact arbitrary and are driven by a configuration file. In addition, the server input queue and client output queue are not necessarily the same, depending on whether the server is a master or slave.

The final step in placing the server is the setting of an environmental variable [3] `CDSERV` which tells `HEPDB` where your server resides. (It is recommended that this assignment be part of your standard setup procedure) The way in which the variable is set varies from platform to platform, a list of possible commands follows:

- UNIX
  Bourne Shell `CDSERV=/hepdb/expdb;export CDSERV`
  C Shell `setenv CDSERV /hepdb/expdb`
  Korn Shell `export CDSERV=/hepdb/expdb`
- VMS
  `CDSERV:==HEPDB:[EXPDB]`
- VM
  `setenv CDSERV CDEXPDB 191`

4.7 Server management

4.7.1 Server configuration

Once storage space has been created a `names` file is then constructed for the server. Below is an example names file for a server. This particular names files contains entries for two databases one for calibration and one for a geometric database. (The next section of the tutorial will demonstrate the creation of these database files.)

An example names file for calib and geom dbs files

```
:nick.config
            :list.ge ca
            :log./hepdb/expdb/logs
            :bad./hepdb/expdb/bad
            :todo./hepdb/expdb/queue
            :queue./hepdb/expdb/queue
            :save./hepdb/expdb/save
            :logl.3
            :wakeup.60

:nick.ge
            :file.<CDEXPDB>.GEO.dbs
            :desc.Geometric database (residing on VM)
            :servers.caliblist1

:nick.ca
            :file./hepdb/expdb/CAL.dbs
            :desc.Calibration Database
            :servers.caliblist2

:nick.caliblist1
            :list.ecal1 ecal2
```

[3] e.g. using the `SETENV` command on VM/CMS systems, setting a global symbol on VAX/VMS systems, or setting a shell variable on Unix systems

```
:nick.caliblist2
            :list.ecal2

:nick.ecal1
            :userid.cdl3
            :node.hepdb
            :localq./hepdb/l3/todo

:nick.ecal2
            :userid.cdl3
            :node.vxl3on
            :queue.disk$db:[cdl3.todo]
            :transport.tcpip
            :localq./hepdb/l3/tovxl3on
```

To understand the way the server uses the information in the `names` file one needs to examine the individual tags within it. A brief description of these tags starting with those in the configuration block follows however a more detailed account of this file can be found elsewhere in this text. Note that additional tags exist for use with the program **CDMOVE**. This tags are only required if you wish to distribute database updates between multiple nodes. See page 111 for more details.

`CONFIG` Indicates the start of the server configuration details.

- `LIST` A list of two character database prefixes. These prefixes act as a pointer within the names file giving specific details about each database file.
- `LOG` The directory where the server logs are written
- `BAD` The directory where the server places bad updates
- `QUEUE` The directory where new updates are placed by HEPDB clients
- `TODO` The directory which the server scans for new updates. If this is the same as the `QUEUE` directory then the server operates as the database master. In other cases it will operate in slave mode.
- `SAVE` The directory where the server saves updates after processing
- `LOGL` The log level for the server
- `WAKEUP` The wakeup interval in seconds for the server
- `SERVERS` The list of remote servers. Each database may have a different list of remote servers. The information on this tag should include all of the remote nodes listed for any of the individual databases, as described below. This tag is processed only by the **CDMOVE** server.

The next type of block in the names file are the database files blocks, identified by the string `:nick.xx` where `xx` is the two character database prefix discussed above.

`:nick.xx` The two character database prefix, e.g. `ca`.

- `FILE` The full path name of the database file. For VM/CMS systems, the syntax is `<user.address>filename.filetype`
- `DESC` Gives a brief description of the database contents. It provides a way of documenting the `names` file.

`SERVERS`	The name specified in this tag is a pointer to a list of remote servers. In our example file above the description of our `calib.dbs` file refers to a serverlist `caliblist1` which itself then points to details of the two remote servers `ecal1` and `ecal2`. As described above, there may be a different list of remote servers for each database.

The remaining entries in our example `names` file are server descriptions. These are identified by an entry `:nick.servername` and store details of remote servers.

`server` The nickname of the servers, e.g. `ecal1`.

`USERID`	Userid under which the server runs on the remote node
`NODE`	Node on which the server runs
`QUEUE`	Input queue on the remote node
`TRANSPORT`	Method by which updates are transmitted
`LOCALQ`	The local directory where updates are written pending transmission to the remote node. This may, in fact, be the same as `QUEUE`, e.g. when the directory is accessible via `NFS` or `AFS`.

4.7.2 Starting the server

Finally we must start the server up. According to the platform you are using the command to do this will vary slightly . The main forms of the command follow:

- UNIX and VMS

  The server can be run in the background or in batch mode.
- VM

  On `VM` the server can be started in one of two ways. You could autolog the machine `CDexperiment` (eg: `CDEXPDB`), or alternatively you could log on as for example `CDEXPDB`, type `HDBSTART` and then exit the session via the command `#CP DISC`.

4.8 Database Creation

Having setup and configured a `HEPDB` server, we now procede to create a new database file. This is a fairly trivial operation and code for doing this is given below. The main routine to take note of is `CDNEW`, which is described in detail in the next section of this manual.

Creating a new database file

```
      PROGRAM CREATE
*     ==============
*
*     Create a new, empty database
*
      PARAMETER   (NWPAW=100000)
      COMMON/PAWC/PAW(NWPAW)
*
*     Initialise Zebra, HBOOK and HEPDB
*
      CALL CDPAW(NWPAW,NHBOOK,IDIV,'USR-DIV',5000,50000,'ZPHU',IRC)
```

```
*
*     Unit for database access
*
      LUNCD  = 1
*
*     Database parameters
*
      NPAIR  = 1
      NREC   = 20000
      NPRE   = 200
      NTOP   = 1
*
*     Accept default record length (1024 words)
*
      LRECL  = 0
      CALL CDNEW(LUNCD,'geome','GEO.dbs',IDIV,NPAIR,NREC,NPRE,NTOP,
     +           LRECL,'F',IRC)
*
*     Set the log level
*
      CALL CDLOGL(' ',3,'A',IRC)
*
*     Terminate
*
      CALL CDEND(' ','A',IRC)

      END
```

For the sake of our example lets assume we create three databases, one for the geometric data, one for calibration and finally one for the auxiliary data. We will name these `GEO.dbs`, `CAL.dbs` and `AUX.dbs` respectively.

4.9 Opening a database

Now we can check that a database exists via the `HEPDB` interactive interface. (Later we show how to open a database via a `HEPDB`-calling `FORTRAN` program).

The following section gives a little detail upon the invocation of the interactive interface.

4.9.1 Invoking the HEPDB interactive interface

The interactive interface is invoked by typing `hepdb`. Note that the variable `CDSERV` must be set, specifying the directory where the appropriate `HEPDB` configuration file is stored.

When you log on to the system you may notice messages saying that macros `xxxSYS, xxxGRP, xxxUSR`, and `xxxLOGON` are not present. This is a normal response and refers to a `HEPDB` facility which allows you to execute a standard set of macros on entry to the interactive interface. The following section explains how the use of these macros is envisaged.

4.9.2 Using start up macros

The use of a subroutine `KULOGN` is as follows:

Description of subroutine KULOGN

```
      SUBROUTINE KULOGN (CHPACK,CHOPT)
*
*    Execute logon kumacs for package 'CHPACK' with options 'CHOPT'
*    KUMACs are xxxSYS, xxxGRP, xxxUSR and xxxLOGON
*
```

Depending on the platform you are running different ways of tailoring where the macros reside are available. The search methods used by the various platforms are as follows:

- VAX/VMS
  Look in directories defined in searchlist xxxPATH ,if xxxPATH not defined, use SYS$LOGIN, SYS$DISK:[] (i.e. current and home directories).
- Unix
  Look in path xxx PATH (If not defined, use current and home directories).
- VM/CMS
  Check disks in xxxPATH. If this is not defined default to using the A disk.

The standard set of macros currently supported by HEPDB are intended for the following use:

- xxxSYS
  This macro may contain a call to a monitoring program
- xxxGRP
  This macro would contain commands specific to a particular group.
- xxxUSR and xxxLOGON
  These two macros are intended to hold user specific information. As the default search order is the current and then home directory, one may use one macro, e.g. HDBUSR for general commands, e.g. creation of standard aliases etc. and the other, e.g. HDBLOGON for directory specific commands.

4.9.3 Opening a database interactively

HEPDB allows a database file to be opened via the OPEN command. The OPEN command must be given with the two character prefix of the database followed (optionally) by the databases' physical file name. The example below shows how to open a file interactively and the terminal output you can expect.

Opening a database interactively

```
HEPDB> open ge geo.dbs
HEPDB  1.02/14 921029 16:46 CERN PROGRAM LIBRARY HEPDB=Q180
 This version created on 921029 at 1756
CDOPNC. opened file GEO.DBS on unit  2 with top directory CDGE and record length 1024
HEPDB>
```

To demonstrate that the file has now been opened you can issue another HEPDB interactive command FILES, which shows a list of all files currently open to HEPDB.

Checking a file is open with FILES.

```
HEPDB>files
 File #   1, unit:  2, top directory:            CDGE
 CDFILC.          1  file(s) are attached
HEPDB>
```

Finally the database should now be closed. This can be done in two ways, explicitly via the CLOSE command if you wish to do more work with HEPDB during the current session or implicitly via the QUIT command if you wish to terminate the current HEPDB session. An example of the first of these cases follows:

Closing a HEPDB file interactively

```
HEPDB>close ge
CDCLSH. closing GEO on unit         2
CDCLSH. closed           1  file(s)
HEPDB>
```

4.9.4 Opening a database from FORTRAN

Below is an example section of FORTRAN code showing how HEPDB's user callable routines can be used to open a database file. Notice how the two character id code of the database is used in the call to CDPREF to obtain the full file name and top directory name of the required database. This information is then used by the call to CDOPEN which then opens the file. Finally note the use of CDEND this is the method of closing the database file. (The character option 'A' signifies that all files should be closed.)

Opening a database from FORTRAN

```
      PROGRAM EGOPEN
*     ==============
*
*     Modify an existing database
*
      PARAMETER    (NWPAW=100000)
      COMMON/PAWC/ PAW(NWPAW)
      PARAMETER    (NKEYS=10)
      PARAMETER    (MAXOBJ=1000)
      CHARACTER*8  CHTAG(NKEYS)
      CHARACTER*10 CHFOR
      CHARACTER*4  CHTOP
      CHARACTER*80 CHFILE
*
*     Initialise Zebra, HBOOK and HEPDB
*
      CALL CDPAW(NWPAW,NHBOOK,IDIV,'USR-DIV',5000,50000,'ZPHU',IRC)
*
*     Unit for database access
```

```
*
      LUNCD  = 1
*
*     Unit for database update (via journal files)
*
      LUNFZ  = 2
*
*     Find the database file and construct the top directory name
*     Pass in 'GE' (two-char id code for database) as 2nd parameter
*
      CALL CDPREF(10,'GE',CHTOP,CHFILE,IRC)
      IF(IRC.GT.4) THEN
         PRINT *,'EGOPEN. STOPPING DUE TO FATAL ERROR FROM CDPREF'
         STOP 16
      ENDIF
*
*     Open the database file
*
      LRECL  = 0
      CALL CDOPEN(LUNCD,LUNFZ,CHTOP,CHFILE,LRECL,IDIV,' ',IRC)
*
*     Do Nothing and Terminate
*
      CALL CDEND(' ','A',IRC)

      END
```

4.10 Database Structure

The HEPDB database management system organises databases in files as a collection of directories and sub-directories (Similar to the UNIX file system structure). Residing at the end of any path-name there maybe zero, one or many actual data objects. These objects are further identified beyond their pathname by keys, which may be a single word of information (integer , hollerith, string etc.) or a vector of such words.

4.10.1 An example structure

Supposing as stated earlier that an experiment currently stores its data under the RZ file system and that therefore the required directory structure is already known and the essential keys for the database are also already known we have a firm basis on which to start to build our HEPDB database structure.

Below is a description of an existing RZ file structure for a geometric database including the directory tree (corresponding to different parts of a detector) and its associated keys.

The required directory structure

```
GEOMETRY                    TYPE |  KEYS
  |                         -----+----------------------------------------
  |                         (I)  |  VAL_STAR  start validity range of object
  |----->PC                 (I)  |  VAL_STOP  end   validity range of object
  |----->DC                 (H)  |  DETECTOR  detector name
```

```
 |----->ST              (H)  |  POINTER   some MZ bank pointer name
 |----->PID
 |----->CAL_WIRE
 |----->CAL_STRI
```

We can now asses the directory structure and the keys to see if any modifications are required as this is the best time to start the implementation of any new keys etc.

For the sake of this example lets assume that the directory structure requires no real modification but we do however think it may be useful to add an extra key to signify the source from which a data object came, this can have one of two values (on-line or off-line data).

After the next short section which explains the use of keys under HEPDB we shall decide on how we are to implement the extra key.

4.10.2 Use of keys under HEPDB

The HEPDB package assumes that all key vectors consist of at least 10 keys. These keys are defined as shown in table 2.1 on page 5 of this manual.

The system keys (the first 5 keys) should not (normally) be touched by the user or application program. The special user (keys 6 to 10) keys may contain any information decided on by the experiment. (However it has been suggested that keys 6 and 7 may contain a unique source identifier and a software reference number respectively.) For instance these keys could be used to hold a key which is common to all databases regardless of their contents.

After the standard keys come a number of key pairs. The number of pairs is a database constant. For example, the CPLEAR collaboration use only one key pair, the start and end times for which a database object is valid. The OPAL collaboration use 3 key pairs - the start and end experiment, run and event number for which a database object is valid.

Additional keys, the so-called user keys, can be declared by the user (within an overall limit of 100 keys).

4.10.3 Example use of keys

From our previous description of the existing RZ file we can see that we have 1 validity range pair VAL_STAR , VAL_STOP. These will be held in keys 11 and 12 and therefore imply a database constant NPAIR (number of pairs) equals the value 1.

The other keys from our geometry RZ files will take the form of normal user keys occupying keys 11+NPAIR*2 and 12+NPAIR*2 or in other words keys 13 and 14.

As discussed before in our example of the geometric database we have decided that an extra key to define the source (on-line, off-line) of a data object would be useful. As this will be common to all our databases (geometric, calibration and auxilliary) we can in this instance allocate a special user key for this job. We shall use key 10 (although we could have used any normal user key instead.).

4.11 Special HEPDB directories

Each HEPDB database file contains two special directories, which are created automatically. Thu our initial database file structure is as shown below.

Initial directory structure

```
CDGE
  |
  |----->HELP
  |----->DICTIONARY
```

Note that the top (root) directory name not actually part of the database itself. However, files accessed though `HEPDB` always have a top directory name formed by concatenating the letters `CD` with the two character database prefix.

4.12 Creating a directory structure

Currently, directories can only be made using the Fortran interface. This is to retain full flexibility in the specification of the various directory parameters such as the key types and tags.

The following code adds to the initial directory structure to create a directory `CALIBRATION` containing the required directories for our example case.

FORTRAN code to create directories

```
      PROGRAM GEDIRS
c|    ================
c|
c|    To Create a directory structure in a database
c|
      PARAMETER    (NWPAW=100000)
      COMMON/PAWC/ PAW(NWPAW)
      PARAMETER    (NKEYS=2)
      PARAMETER    (MAXOBJ=1000)
      PARAMETER    (NODIRS=6)
      CHARACTER*40 DITAG(NODIRS)
      CHARACTER*8  CHTAG(NKEYS)
      CHARACTER*10 CHFOR
      CHARACTER*4  CHTOP
      CHARACTER*80 CHFILE
      CHARACTER*40 DNAME
c|
c|    Initialise directory names
c|
      DATA DITAG/'PC','DC','ST','PID','CAL_WIRE','CAL_STRI'/
      CALL CDPAW(NWPAW,NHBOOK,IDIV,'USR-DIV',5000,50000,'ZPHU',IRC)
      LUNCD  = 1
      LUNFZ  = 2
      CALL CDPREF(10,'GE',CHTOP,CHFILE,IRC)
      IF(IRC.GT.4) THEN
         PRINT *,'EGOPEN. STOPPING DUE TO FATAL ERROR FROM CDPREF'
         STOP 16
      ENDIF
      LRECL  = 0
      CALL CDOPEN(LUNCD,LUNFZ,CHTOP,CHFILE,LRECL,IDIV,' ',IRC)
c|
c|    Define key types and tags
c|
```

```
      CHFOR = 'HH'
      CHTAG(1) = 'DETECTOR'
      CHTAG(2) = 'POINTER '
      IPREC=0
      DELTA=0.0
c|
c|    Loop to create directories
c|
      DO 99 IDX=1,NODIRS
         DNAME='//CDGE/GEOMETRY/'//DITAG(IDX)
         PRINT *,'CREATING DIRECTORY ',DNAME,' ...'
         CALL CDMDIR(DNAME,NKEYS,CHFOR,CHTAG,MAXOBJ,IPREC,DELTA
     +               ,'CP',IRC)
         PRINT *,'IRC AFTER CDMDIR =',IRC
99    CONTINUE
c|
c|    Terminate
c|
      CALL CDEND(' ','A',IRC)
      END
c|
```

This simple program simply loops over the directory names required which are held in the character array `DITAG` and performs a call to routine `CDMDIR` which creates the directory. Note that the keys that each directory will contain are also set at this point via the `CHTAG` array which holds the keys `HOLLERITH` tags and the `CHFOR` character array which defines the keys type (in this case the two user keys are `HOLLERITH`s so we use `HH`).

We can now examine the directory structure via the interactive interfaces' `TREE` command. An example transcript follows:

Examining directory structure with TREE

```
 HEPDB>tree
 CDTREK. directory tree structure below //CDGE down          99  levels
 //CDGE
       /HELP
       /DICTIONARY
       /GEOMETRY
                /PC
                    /1
                /DC
                    /1
                /ST
                    /1
                /PID
                    /1
                /CAL_WIRE
                         /1
                /CAL_STRI
                         /1
          16  subdirectories found
HEPDB>
```

4.13 Deleting a directory Structure

The following section for reasons of completeness describes how a directory structure may be modified.

4.13.1 Deleting a directory via FORTRAN

To remove a directory via a `FORTRAN` program we make a call to the routine `CDDDIR`. An outline of a program that would delete our `DC` directory is shown below.

FORTRAN deletion of a directory

```
      PROGRAM DELDIR
*     ==============
*
*     To delete a directory in a database structure
*     First do usual open database code.
*
      PARAMETER    (NWPAW=100000)
      COMMON/PAWC/ PAW(NWPAW)
      PARAMETER    (MAXOBJ=1000)
      CHARACTER*4  CHTOP
      CHARACTER*80 CHFILE
      CHARACTER*40 DNAME

      CALL CDPAW(NWPAW,NHBOOK,IDIV,'USR-DIV',5000,50000,'ZPHU',IRC)
      LUNCD  = 1
      LUNFZ  = 2
      CALL CDPREF(10,'GE',CHTOP,CHFILE,IRC)
      IF(IRC.GT.4) THEN
         PRINT *,'EGOPEN. STOPPING DUE TO FATAL ERROR FROM CDPREF'
         STOP 16
      ENDIF
      LRECL  = 0
      CALL CDOPEN(LUNCD,LUNFZ,CHTOP,CHFILE,LRECL,IDIV,' ',IRC)
*
*     Construct directory name to be removed
*
      DNAME='//CDGE/GEOMETRY/DC'
*
*     Delete the directory
*
      CALL CDDDIR (DNAME,' ',IRC)
*
*     Terminate
*
      CALL CDEND(' ','A',IRC)
      END
```

If we were now to issue another interactive `TREE` command we would notice that the directory is no longer part of our database structure.

4.13.2 Multiple directory deletion

As well as allowing individual directories to be deleted HEPDB also allows the deletion of multiple directories via the CDKEEP routine. As the name suggests, all subdirectories that are not specified are deleted. The call to CDKEEP is envisaged as follows:

Multiple directory deletion via CDKEEP

```
      PROGRAM MDELDIR
*     ================
*
*     To delete many directories in a database structure
*     First do usual open database code.
*
      PARAMETER    (NWPAW=100000)
      COMMON/PAWC/ PAW(NWPAW)
      PARAMETER    (MAXOBJ=1000)
      CHARACTER*4  CHTOP
      CHARACTER*80 CHFILE
      CHARACTER*40 DNAME(10)
      CALL CDPAW(NWPAW,NHBOOK,IDIV,'USR-DIV',5000,50000,'ZPHU',IRC)
      LUNCD  = 1
      LUNFZ  = 2
      CALL CDPREF(10,'GE',CHTOP,CHFILE,IRC)
      IF(IRC.GT.4) THEN
         PRINT *,'EGOPEN. STOPPING DUE TO FATAL ERROR FROM CDPREF'
         STOP 16
      ENDIF
      LRECL  = 0
      CALL CDOPEN(LUNCD,LUNFZ,CHTOP,CHFILE,LRECL,IDIV,' ',IRC)
*
*     Construct List of directory paths to be kept
*
      DNAME(1)='//CDGE/GEOMETRY/PC'
      DNAME(2)='//CDGE/GEOMETRY/DC'
*
*     Delete the directory
*
      CALL CDKEEP (DNAME,2,' ',IRC)
*
*     Terminate
*
      CALL CDEND(' ','A',IRC)
      END
```

4.13.3 Other directory related operations

As well as the commands given above for the creation and deletion of directories there are also commands to allow manipulation of directory partitions. (See elsewhere in this manual for description of directory partitioning.) However such commands are outside the scope of this tutorial and shall therefore only be mentioned in passing below.

Subroutines exist to allow the conversion of a non-partitioned directory to a partitioned one (routine CDPART) and to allow the deletion of specific directory partitions (routine CDPURP).

4.14 Conversion of existing databases

Having created a database and the required directory structure, we now procede to add the data.

As we already know in our example the data is held in RZ files in the form of ZEBRA banks. As the object type used by HEPDB is the ZEBRA bank the conversion of the old CPLEAR database is fairly simple. The only modification to the original data will be the addition of the new special user key 10 as discussed above.

The code below shows the conversion of the original RZ geometric database to HEPDB format. After the code follows a brief explanation although the comments within the code tell of its operation.

Conversion of GEOMETRY database

```
      PROGRAM GEOCONV
c     ===============
c+------------------------------------------------------+
c|    Program to convert    RZ  geo database -> HEPDB     |
c|-----------------------------+-------------------------|
c|    RZGEO keys: VAL_STAR (I)  |  For all directories     |
c|                VAL_STOP (I)  +-------------------------|
c|                DETECTOR (H)                             |
c|                POINTER  (H)                             |
c|    insertion time = RZ date/time                        |
c|---------------------------------------+----------------|
c|    HEPDB keys: NPAIR    = 1            |                |
c|                VAL_STAR = KEYS(11) (I) | For all        |
c|                VAL_STOP = KEYS(12) (I) | geometry       |
c|                NUSER    = 2            | directories    |
c|                DETECTOR = KEYS(13) (H) |                |
c|                POINTER  = KEYS(14) (H) |                |
c|    insertion time = KEYS(IDHINS)                        |
c|--------------------------------------------------------|
c|    Output pathnames:                                    |
c|    //CDGE/GEOMETRY                                      |
c|    //CDGE/GEOMETRY/PC                                   |
c|    //CDGE/GEOMETRY/DC                                   |
c|    //CDGE/GEOMETRY/ST                                   |
c|    //CDGE/GEOMETRY/PID                                  |
c|    //CDGE/GEOMETRY/CAL_WIRE                             |
c|    //CDGE/GEOMETRY/CAL_STRI                             |
c+------------------------------------------------------+
c
      PARAMETER     (NWPAW=200000)
      COMMON/PAWC/   PAW(NWPAW)
      COMMON/USRLNK/ IDIV,LADDR
      CHARACTER*4    CHTOP
      CHARACTER*80   CHFILE
      EXTERNAL       CPGEOC
c|
c|    Initialise Zebra, HBOOK and HEPDB
c|
      CALL CDPAW(NWPAW,NHBOOK,IDIV,'USR-DIV',10000,150000,'ZPU',IRC)
      CALL MZLINK(IDIV,'/USRLNK/',LADDR,LADDR,LADDR)
      LUNCD  = 1
      LUNFZ  = 2
```

```
      LUNRZ  = 3
c|
c|    Open RZ geometry database (RZGEO.DATA)
c|
      LRECL  = 0
      CALL RZOPEN(LUNRZ,'RZGEO','rzgeo.data',' ',LRECL,IRC)
      CALL RZFILE(LUNRZ,'RZGEO',' ')
c|
c|    Find the database file and construct the top directory name
c|    Open the database file
c|
      CALL CDPREF(10,'GE',CHTOP,CHFILE,IRC)
      LRECL  = 0
      CALL CDOPEN(LUNCD,LUNFZ,CHTOP,CHFILE,LRECL,IDIV,' ',IRC)
c|
c|    Loop over all sub-directories in RZGEO.DATA
c|
      CALL RZSCAN('//RZGEO',CPGEOC)
c|
c|    Terminate
c|
      CALL CDEND(' ','A',IRC)
      CALL RZCLOS(' ','A')
      END

      SUBROUTINE CPGEOC(CHDIR)
c
c+------------------------------------------------------+
c|    Routine to retrieve,convert and store old RZ objects |
c|    under HEPDB                                        |
c+------------------------------------------------------+
c
      COMMON /USRLNK/   IDIV,LADDR
      COMMON /QUEST/    IQUEST(100)
      PARAMETER         (NRZKS=5)
      PARAMETER         (MAXKEY=1000)
      PARAMETER         (IOBJECTS=18)
      INTEGER            HDBKEYS(15),RZKYTOGET(NRZKS)
      INTEGER            NKEYRET,KEYSARR(NRZKS,MAXKEY)
      CHARACTER*(*)      CHDIR
      CHARACTER*255      CHSAVE,DEST
      CHARACTER*(NRZKS)  CHFORM
      CHARACTER*8        CHTAG(NRZKS)
      CHARACTER*4        CTEMP(3)
      DATA               NENTRY/0/
      SAVE               NENTRY
c|
c|    Check for exit from RZSCAN loop
c|
      LADDR=0
      IF (NENTRY.EQ.0) THEN
         NENTRY=1
         RETURN
      ENDIF
c|
c|    Set Current RZ directory
c|
```

```
      LDIR    = LENOCC(CHDIR)
      CHSAVE  = CHDIR(1:LDIR)
      CALL RZCDIR(CHSAVE(1:LDIR),' ')
      PRINT *,'-- RZ Directory Change -----------------------'
      PRINT *,' ', CHSAVE(1:30)
      PRINT *,'---------------------------------------------'
c|
c|    Get CWD key definitions
c|
      CALL RZKEYS(NRZKS,MAXKEY,KEYSARR,NKEYRET)
      PRINT *,' No of objects in CWD          :',NKEYRET
      CALL RZKEYD (NWKEY,CHFORM,CHTAG)
      PRINT *,' Key Format for this directory :',CHFORM
c|
c|    Now loop over all objects in CWD, display and convert
c|
      DO I=1,NKEYRET
           PRINT*,' - OBJECT No. ',I,' ------------------'
           PRINT*,' ',CHTAG(1),':',KEYSARR(1,I),'  '
     +           ,' ',CHTAG(2),':',KEYSARR(2,I),'  '
           CALL UHTOC(KEYSARR(3,I),4,CTEMP(1),4)
           CALL UHTOC(KEYSARR(4,I),4,CTEMP(2),4)
           CALL UHTOC(KEYSARR(5,I),4,CTEMP(3),4)
           PRINT*,' ',CHTAG(3),':',CTEMP(1),'  '
     +           ,' ',CHTAG(4),':',CTEMP(2),'  '
           PRINT*,' ',CHTAG(5),':',CTEMP(3),'  '
c|
c|         Load object ,I, into zebra store
c|
           ICYCLE=9999
           JBIAS=2
           DO J=1,NRZKS
                RZKYTOGET(J)=KEYSARR(J,I)
           ENDDO
           CALL RZIN(IDIV,LADDR,JBIAS,RZKYTOGET,ICYCLE,' ')
c|
c|         Now set HEPDB keys, and pack time stamp
c|
           CALL RZDATE(IQUEST(14),IDATE,ITIME,1)
           CALL CDPKTM(IDATE,ITIME,IPACK,IRC)
           HDBKEYS(4)=IPACK
c|
           DO J=1,5
                HDBKEYS(10+J)=RZKYTOGET(J)
           ENDDO
c|
c|         Set key 10 to 0 (Our new key!)
c|
           HDBKEYS(10)=0
c|
c|         Now store object at LADDR under HEPDB
c|
           DEST='//CDGE/GEOMETRY/'//CHDIR(9:LDIR)
           PRINT*,' Destination path        :',DEST
           CALL CDSTOR(DEST,LADDR,LKYBK,IDIV,HDBKEYS,'H',IRC)
           IF (IRC.NE.0) THEN
               PRINT*,' Error! CDSTOR IRC=',IRC
```

```
            STOP
         ENDIF
         CALL RZCDIR(CHSAVE(1:LDIR),' ')
         CALL MZDROP(IDIV,LADDR,' ')
         LADDR=0
         PRINT*,' Object ',I,' Stored under HDB'
         PRINT *,' '
      ENDDO
      CALL CDSTSV(' ',0,IRC)
      CALL RZCDIR(CHSAVE(1:LDIR),' ')
      PRINT *,' '
      END
```

The flow of the above program starts by opening both the source RZ file and the destination HEPDB database file. It then makes a call to the RZ routine RZSCAN which visits each directory of the RZ file in turn passing execution to subroutine CPGEOC.

Once inside this routine the current RZ directory is set by the routine RZCDIR. The RZ key definitions and their values are retrieved via calls to RZKEYD and RZKEYS respectively. The objects within the directory are then looped over one at a time , each being brought into ZEBRA store and then output to HEPDB via the CDSTOR routine. Note that the special user key 10 we now wish to add is simply given a 0 value in this case as we assume all current data to be from the off-line source. (This is for our example although other code could be introduced to set this key accordingly) Once all objects in the RZ directory have been processed the update file is sent to the HEPDB database server. This procedure is then repeated for each subdirectory of the RZ file.

An important point to be noted at this time is the manipulation of a system key. This is key 4 `insertion time` (IDHINS). As we would like to keep the information regarding the original insertion time of the RZ objects we suppress HEPDB from re-writing this key and force it to honor the original RZ insertion time by the option `'H'` in our call to CDSTOR. This is the only legitimate way in which a system key should be altered.

4.15 Data storage

This section explains how information can be entered into database and how to retrieve or delete information.

Under HEPDB the user can store data structures created under the ZEBRA system in the database. These ZEBRA data structures can range from a single ZEBRA bank through to a complex ZEBRA data structure. When data is submitted for inclusion to the database the pathname of the directory where the data is to reside and the key vector must also be supplied.

HEPDB also provides facilities for the storage of other data types. Note that the data is always converting into ZEBRA banks, but that this conversion is automatic and transparent to the user. These facilities include the routines CDTEXT, CDCHAR and CDVECT which allow the storage of text files , character data and vectors respectively.

4.15.1 Storing a ZEBRA data structure

Providing the directory in which the data is to reside in has already been created and the data structure is already in memory the user can store the data in memory to disk by use of the routine CDSTOR. A demonstration of this routine is shown in the above program to convert an existing RZ database.

4.15.2 The storage of text file data

Whilst discussing HEPDB's storage facilities we shall examine briefly how the routine CDTEXT allows us to store text files under HEPDB. It achieves this by converting a text file to or from a ZEBRA structure suitable for storage under HEPDB.

Suppose we had a file (for this examples sake residing on VM) that we wished to store under HEPDB. Assuming the name of the file was TEST TEXT A1 we could use a call similar to the following to retrieve the file from disk and to return the address of the ZEBRA bank now containing it ready for insertion to the database.

Example of storing a text file

```
*
*     LUN for textfile access
*
      TXLUN=10
*
      CALL CDTEXT(TXLUN,'TEST.TEXT.A1',PATH,LBANK,'R',IRC)
```

The character option 'R' intructs the CDTEXT routine to read the file from disk. In this case the bank address will be returned in LBANK.

4.15.3 Storage of character based data

The storage of character based data is performed by the routine CDCHAR which has similar functionality to the previously described CDTEXT routine, except that the data is moved from and to character arrays rather than text files.

4.15.4 Storage of vector based data

Once again the storage of vector data is allowed by converting vectors to the ZEBRA bank format before committing that bank to the database. The routine available for this operation is CDVECT.

To demonstrate this imagine that one of the example databases we wish to convert from the RZ format currently consists not of ZEBRA banks but of FORTRAN integer vectors.

The conversion program must now read in the vectors and convert them to a HEPDB format before they can be stored in the database. The code shown below offers a suggestion of how the previous conversion program could be modified to handle this storage of vectors.

Example of data conversion using CDVECT

```
                  .
                  . other code
                  .
c|
c|       Zero temporary vector, then fill with next object
c|
         CALL VZERO(ITEMPVECT,80000)
         CALL RZVIN(ITEMPVECT,80000,NFILE,RZKYTOGET,ICYCLE,' ')
         IF(IQUEST(1).NE.0) THEN
              PRINT *,' Error! RZVIN gives ',IQUEST(1)
```

```
               STOP
          ENDIF
          PRINT *,' No. of elements in RZ vector :',NFILE
                   .
                   . other code
                   .
c|
c|        Convert the array to a ZEBRA data structure
c|
          CALL CDVECT(' ',ITEMPVECT,NFILE,LADDR,'PI',IRC)
          IF(IRC.NE.0) THEN
               PRINT *,' Error! CDVECT IRC=',IRC
               STOP
          ENDIF
c|
c|        Now store bank at LADDR under HEPDB
c|
          DEST='//CDAU/AUX/'//CHDIR(9:LDIR)
          PRINT*,' Destination path : ',DEST
          CALL CDSTOR(DEST,LADDR,LKYBK,IDIV,HDBKEYS,'H',IRC)
          IF (IRC.NE.0) THEN
               PRINT *,' Error! CDSTOR IRC=',IRC
               STOP
          ENDIF
                   .
                   . other code
                   .
      CALL CDSTSV(' ',0,IRC)
      CALL RZCDIR(CHSAVE(1:LDIR),' ')
      PRINT *,' '
      END
```

4.16 Data retrieval

Routines are available to allow the simple retrieval of data from the database. The following section addresses the general issue of data retrieval with a brief description of the available routines and examines the problems of retrieving data into already existing software.

4.16.1 Retrieving ZEBRA data structures based on a key vector

Now that we have created and filled our database with data we have to generate code which allows us to read the data into our programs. Taking the `CPLEAR auxiliary` database (integer vectors stored under `HEPDB`) as an example and supposing we want to simply load the auxiliary data for a given run into our program's arrays we could approach the problem as follows.

The general loop of tasks would be to set keys for the object (and consequently vector) that is required, retrieve the object into `ZEBRA` store and finally convert the object back to a vector the software can use.

We could write a subroutine called say `HDBFET` to fetch the database object with parameters as follows:

Example calling parameters of HDBFET

```
      SUBROUTINE HDBFET(PATH,NUMRUN,KY1,KY2,LBANK)
*
*     Where: PATH   is the absolute path to where the data resides in the database (string)
*            NUMRUN is the instant of validity for the object we require (integer)
*            KY1    user key 1 say the detector name (4 Character hollerith)
*            KY2    user key 2 say the name of the destination array (4 char hollerith)
*            LBANK  the address of the retrieved bank.
```

This subroutine could be implemented as follows.

Possible implementation of HDBFET

```
      SUBROUTINE HDBFET(PATH,IVALID,UKY1,UKY2,LBANK)
c|
c+-----------------------------------------------------------------+
c|    HDBFET    : Routine to retrieve HEPDB object into a bank       |
c|                valid at NUMRUN with keys UKY1/2                   |
c+-----------------------------------------------------------------+
c|
+SEQ,CPPOIN.
+SEQ,CPBANK.
      COMMON/QUEST/IQUEST(100)
      PARAMETER       (IONLINE=0)
      PARAMETER       (IOFFLINE=1)
      PARAMETER       (NHDBKEYS=14)
      INTEGER          LBANK
      INTEGER          IMASK(NHDBKEYS)
      INTEGER          HDBKEYS(NHDBKEYS)
      CHARACTER*255    PATH
      CHARACTER*4      UKY1,UKY2
      CHARACTER*8      MESS
      INTEGER          IVALID
c|
c|    Convert CHARACTER --> HOLLERITH for user keys
c|
      CALL UCTOH(UKY1,HDBKEYS(13),4,4)
      CALL UCTOH(UKY2,HDBKEYS(14),4,4)
c|
c|  Set up IMASK for relevant keys
c|
      CALL VZERO (IMASK,NHDBKEYS)
      IMASK(10)=1
      IMASK(13)=1
      IMASK(14)=1
c|
c|    Make the search for an off-line object first
c|
      HDBKEYS(10)=IOFFLINE
c|
c|    Get Bank from Database
c|
      CALL CDUSEM(PATH,LBANK,IVALID,IMASK,HDBKEYS,' ',IRC)
      IF (IRC.EQ.33) THEN
```

```
         HDBKEYS(10)=IONLINE
         CALL CDUSEM(PATH,LBANK,IVALID,IMASK,HDBKEYS,' ',IRC)
      ENDIF
      IF (IRC.NE.0) THEN
         PRINT *,'Error. CDUSEM IRC=',IRC
         PRINT *,'For user keys :',UKY1,' ',UKY2
         PRINT *,'With path :',PATH
         STOP
      ENDIF
      IF (IB(LBANK+10).EQ.IOFFLINE) THEN
         MESS='OFF-LINE'
      ELSE
         MESS='ON-LINE '
      ENDIF
      IF (IQUEST(2).NE.0) THEN
        PRINT *,
     + '[Source-->',MESS,'] [Keys-->',UKY1,' ',UKY2,' Data from DISK ]'
      ELSE
        PRINT *,
     + '[Source-->',MESS,'] [Keys-->',UKY1,' ',UKY2,' Data from CACHE]'
      ENDIF
      RETURN
      END
```

The comments within the code describe the basic flow of the code, however there are a couple of points to note. Firstly notice how the new key (key 10) we introduced is used to check for off-line versions of data objects before resorting to taking an on-line version of the data.

Finally notice how `IQUEST(2)` is checked at the end of the routine to see if data was refreshed from disk or from the cache of objects already in memory. This is a powerful feature of `HEPDB` as it saves the need for repeated access to the same object each time the current run-number changes as it will not always follow that the required object will need to be changed. Later in this section we discuss how the command `CDFREE` is used to declare that an object is to be cached.

Now let's imagine a typical call to such a routine. Suppose we want to retrieve an integer array stored under `HEPDB` with user keys `CALO` and `CNEX` (which represent the integer array of `CALO`rimeter `NEX`t wires) for run number `NUMRUN`. We also assume that the object resides in directory `//CDAU/AUX/CALPA`. A section of code to retrieve and convert the object to an array in memory would have the form shown below.

Example calling sequence to HDBFET

```
c+-----------------------------------------------------------------+
c|  Input CANEXT data from //CDAU/AUX/CALPA                         |
c|  For keys CALO , CNEX ---> to vector ICANEX (size 68850)         |
c+-----------------------------------------------------------------+
c|  Get object from HEPDB according to run number/user keys
c|
      PATH='//CDAU/AUX/CALPA'
      CALL HDBFET(PATH,NUMRUN,'CALO','CNEX',LBANK)
c|
c|     Unpack bank into a vector
c|
      CALL CDVECT(' ',ICANEX,68850,LB(LBANK-1),'GI',IRC)
```

```
      IF (IRC.NE.0) THEN
         WRITE(LUPRNT,*) 'Error. CDVECT IRC=',IRC
         STOP
      ENDIF
```

The main points to note here are that the call to CDVECT must be supplied with the type of the vector (in this case INTEGER) via the character options 'GI' (Get Integer) and the expected size of the vector in elements.

Also note that the data part of the object resides at LB(LBANK-1) of the keys bank and this is the address that must be passed to CDVECT via its fourth parameter.

As mentioned before, HEPDB has a facility for the caching of objects in memory. Objects are only retrieved from disk when no matching object exists in the cache or when explicitly requested by the user. This is performed as follows. After an object has been used, it is marked by the user as a candidate for deleted from the cache using the routine CDFREE.

We therefore add an additional subroutine called say AUCAS to declare an object to be a candidate for deletion from database memory with parameters defined as follows:

CALL **AUCAS** (SUBROUTINE AUCAS(PATH,KY1,KY2,LBANK)

PATH	Character variable specifying the directory in which the object resides
KY1	user key 1 of the object to be dropped
KY2	user key 2 of the object to be dropped
LBANK	the address of the keys bank.

This subroutine could be implemented as follows.

Possible implementation of AUCAS

```
      SUBROUTINE AUCAS(PATH,KY1,KY2,LBANK)
c|
c+-----------------------------------------------------------------+
c|   AUCAS      : Routine to declare  HEPDB object in a bank        |
c|                 at LBANK available for deletion.                 |
c+-----------------------------------------------------------------+
c|
+SEQ,CPPOIN.
+SEQ,CPBANK.
      CHARACTER*4       KY1,KY2
      COMMON/QUEST/     IQUEST(100)
      INTEGER           LBANK,IKV(14),IMASK(14)
      CHARACTER*255     PATH
c|
c|    Zero, then set the user key mask
c|
      CALL VZERO(IMASK,14)
      IMASK(13)=1
      IMASK(14)=1
      CALL UCTOH(KY1,IKV(13),4,4)
```

```
      CALL UCTOH(KY2,IKV(14),4,4)
      CALL CDFREE (PATH,LBANK,IMASK,IKV,' ',IRC)
      IF (IRC.NE.0) THEN
          PRINT *,'Error: CDFREE IRC=',IRC
          STOP
      ENDIF
      RETURN
      END
```

So given our previous example call to HDBFET with keys CALO and CNEX the appropriate call to AUCAS would be:

Possible implementation of AUCAS

```
      CALL AUCAS(PATH,'CALO','CNEX',LBANK)
```

4.17 Retrieval of data into existing software

When existing software has to be modified to accept data from HEPDB it would appear that there could be a problem in loading objects into predefined positions in ZEBRA store. To overcome this problem HEPDB provides a routine CDGET which can be used to retrieve a data structure to a user specified location.

However it must be noted that the caching facilities offered to routines such as CDUSEM are not available with CDGET. Full details of the use of this routine are give elsewhere in this manual.

As well as the routine CDGET the user may get objects in the usual way (using CDUSE, CDUSEM) and then use the ZEBRA routine ZSHUNT to move the required databank to a specified location in memory. An example of such a 'shunt' is given below , note the code uses the routines HDBFET and AUCAS as defined above.

Example use of ZSHUNT

```
               .
               .some code
               .
c+-------------------------------------------------------------------+
c|     Input universal geometry (LIGDX0)                              |
c+-------------------------------------------------------------------+
         CALL HDBFET(PATH,NUMRUN,DETNAM(IDET),'GDX0',LBANK)
         CALL ZSHUNT(IDVGDX,LB(LBANK-1),LMIGDX,0,0)
         LIGDX0(IDET)=LB(LMIGDX)
         CALL UCTOH('IGDX',IB(LIGDX0(IDET)-4),4,4)
         IB(LIGDX0(IDET)-5)=IDET
         CALL RDBGDX(IDET)
         CALL AUCAS(PATH,DETNAM(IDET),'GDX0',LBANK)
               .
               .some code
               .
```

4.18 Data removal

4.18.1 Deletion based on a key vector

To allow deletion of object from the database based upon a user specified key vector `HEPDB` provides the user callable routine `CDPURK`. Once again as in the call to `CDSTOR` discussed earlier the user sets up a key vector of size enough to hold all system, special user, validity and normal user keys and fills in the required elements for the envisaged deletion. The user then creates another vector (in our example to follow it is called `IMASK`) which acts as a mask specifying which elements of our key vector are to be considered in the deletion operation (this is set by the element taking a non-zero value).

Note also that the routine `CDPURK` can be used to ‘undelete’ previously deleted objects. The specification of the object to be restored is the same as that of deleting an object. To perform the undelete operation the user simply specifies character option `'U'`.

Part III

HEPDB callable routines

Chapter 5: Description of user callable HEPDB routines

5.1 Initialisation and termination

5.1.1 HEPDB, Zebra and HBOOK initialisation

```
CALL CDPAW (NWPAW,NHBOOK,IDIV*,CHNAME,NW,NWMAX,CHOPT,IRC*)
```

NWPAW	The number of words of dynamic store in common /PAWC/
NHBOOK	Variable containing the number of words for use by HBOOK
IDIV	Variable containing the index of user division created in common /PAWC/
CHNAME	Name of user division
NW	Initial number of words for user division
NWMAX	Maximum number of words for user division
CHOPT	Character option
	' ' Initialise HEPDB divisions in common /PAWC/, including user division CHNAME.
	'Z' Also initialise Zebra via a call to MZEBRA
	'P' Issue call to MZPAW
	'H' Also initialise HBOOK with NHBOOK words
IRC	Integer return code
	0 Normal completion

This routine initialises Zebra, HBOOK and HEPDB. Note that HEPDB is automatically initialised on the first call to CDOPEN.

5.1.2 Obtain file name from database prefix

```
CALL CDPREF (LUN,CHPREF,CHTOP*,CHFILE*,IRC*)
```

LUN	Fortran logical unit for accessing the names file.
CHPREF	Two character database prefix
CHTOP	Top directory name constructed from database prefix
CHFILE	Full file name of the database
IRC	Integer return code
	0 Normal completion

This routine returns the top directory name and full file name of a HEPDB database, identified by a unique two character prefix. In the case of VM/CMS systems, the appropriate mini-disks are linked and accessed automatically. The environmental variable CDSERV must be set before calling this routine, as shown below.

Unix	Bourne shell	`CDSERV=/hepdb/l3; export CDSERV`
	C shell	`setenv CDSERV /hepdb/l3`
	Korn shell	`export CDSERV=/hepdb/l3`
VMS		`CDSERV:==HEPDB:[L3]`
VM/CMS		`setenv CDSERV cdl3.191`

5.1.3 Access an existing database file

`CALL` **CDOPEN** `(*LUNDB*,LUNFZ,CHTOP,CHFILE,*LRECL*,IDIV,CHOPT,IRC*)`

`LUNDB` Fortran logical unit or C file pointer for accessing the database file.

`LUNFZ` Fortran logical unit to be used for sending updates to the database server.

`CHTOP` Name of the Top Directory

`CHFILE` Character variable giving the file name

`LRECL` Integer variable specifying the record length of the database file in words. If a value of zero is given on input, the record length will be automatically determined from the file itself. The actual value used is returned in this variable if the file is successfully opened.

`IDIV` User Division

`CHOPT` Character Option

- `'E'` Expand system division of the store in which `IDIV` resides
- `'Q'` Do not print messages such as version number
- `'N'` The database file is in native format (exchange mode is the default)
- `'T'` Suppress check on insertion time. [1]
- `'C'` Use C I/O instead of Fortran I/O
- `'P'` Preserve case of file name (Unix systems)
- `'R'` Access the database in read only mode - updates not permitted.

`IRC` Integer return code

- `0` Normal completion
- `-1` Invalid top directory name
- `-2` The file is already open with correct `LUNDB` and `CHTOP`
- `-3` The file is already open with wrong `LUNDB` or `CHTOP`
- `-4` Already a file is opened with the same unique identifier as requested for this `CHTOP`
- `-5` Invalid process name in Online context
- `-6` Error in `IC_BOOK` for booking the `CACHE`
- `-7` Error in `CC_SETUP` for reserving the `CLUSCOM`
- `-8` Cannot open journal file in server context
- `-9` Unable to open FZ communication channel
- `-10` Host unable to open RZ file

On `VAX/VMS` systems, C I/O is automatically triggered for files of record type `STREAM_LF`.

[1] By default, only objects inserted prior to the local time as returned by the KERNLIB routine `DATIME` will be visible via `CDUSE` or `CDUSEM`.

5.1.4 Create and access a new database file

```
CALL CDNEW (*LUNDB*,CHTOP,CHFILE,IDIV,NPAIR,NQUO,NPRE,NTOP,LRECL,CHOPT,IRC*)
```

LUNDB Fortran logical unit or C file pointer

CHTOP Name of the Top Directory

CHFILE Character variable giving the file name

IDIV User Division

NPAIR Number of key pairs to be used in object selection.

NQUO Quota for the database file (see RZMAKE).

NPRE Number of records that should be preformatted (essential for VM systems or when afs access is desired)

NTOP Unique identifier for the database file (optional)

CHOPT Character Option as for RZOPEN/RZMAKE **except** exchange mode is the default.

- 'A' In the case of more than one key pair, all pairs will be checked in turn.
- ' ' (Default) Assume a hierarchy in the validity keys. That is, the second and subsequent key pairs are only checked if the match on the first pair was successful (e.g. in the case of PERIOD/RUN/EVENT, the run validity is only checked if the period matches etc.)

IRC Integer return code

5.1.5 Send or cancel pending database updates

```
CALL CDSAVE (CHTOP,CHOPT,IRC*)
```

CHTOP Name of the Top Directory (without the leading //). See below.

CHOPT Character variable specifying the options required.

- ' ' default - any pending updates are sent and the journal file reopened.
- 'P' Pending updates are purged.
- 'S' On Unix systems, a signal is sent to the server indicating that new updates have arrived. This will cause the server to wakeup immediately, rather than at the default interval.
- 'W' Wait for response from server (not yet implemented).

It is not normally necessary to call CDSAVE to send updates to the server. Each time an update is made, a check is performed to see if it is for the same database as previous updates. If not, pending updates are sent. When CDEND is called, any remaining updates are sent.

To send pending updates and switch to a new database, specify in CHTOP the top directory name of the *new* database (e.g. CDXX for a database with prefix XX).

To send pending updates without switching to a new database, specify a blank character.

CDSAVE will only send updates if the top directory name differs from

5.1.6 Terminate access to one or all database files

```
CALL CDEND (CHDIR,CHOPT,IRC*)
```

CHDIR Character variable specifying the name of the top directory of the file to be closed.

CHOPT Character variable specifying the options required.

' '	default, close file specified by the variable CHDIR
'A'	close all files - CHDIR not used
'C'	journal file is closed and sent to the server. (By default, the journal file is reopened.)
'P'	journal file is purged - updates will not be performed.
'S'	For CDSERV only - suppresses call to CDSAVE
'W'	call MZWIPE for the division(s) associated with the files that are closed.

IRC Integer return code

0	Normal completion

This routine closes the specified database file by calling the RZ routine RZCLOS. Any existing journal file is closed and sent to the database server. If option A is not specified, the journal file will be reopened after being sent to the server.

5.2 Alias manipulation

5.2.1 Enter, delete or retrieve an alias

```
CALL CDALIA (PATH,ALIAS,CHOPT,IRC*)
```

PATH Character string specifying the directory path name

ALIAS Character string specifying the alias or in which the alias is returned (option R)

CHOPT Character variable specifying the options required.

'D'	Delete the current alias definition for the specified path name (ALIAS is ignored).
'P'	Print the current alias definition for PATH (ALIAS is ignored).
'G'	Get the current alias for PATH. The alias is returned in ALIAS.
'R'	Return the current equivalence name for ALIAS in PATH
'S'	Set the specified alias for the specified path name for this session
'U'	As option S, but also updates the database

IRC Integer return code

0	Normal completion
182	Illegal path name
185	Illegal top directory name
187	FZOUT fails to write on the sequential file
188	Error in RZ for writing to the random access file
201	Dictionary directory not found
205	Not a valid alias

This routine may be used to enter, remove, print or retrieve aliases.
See page 6 for a discussion of HEPDB aliases.

5.3 Mnemonic name manipulation

5.3.1 Enter, delete or retrieve a mnemonic name

```
CALL CDNAME (PATH,NW,CHTAG,CHOPT,IRC*)
```

`PATH` Character string specifying the directory path name in which the alias is returned (option `R`)

`NW` Integer variable specifying the number of significant elements of `CHTAG`

`CHTAG` Character array of significant length `NW` containing the mnemonic names for each element of objects stored in the directory `PATH`

`CHOPT` Character variable specifying the options required.

- `'D'` Delete the current mnemonic name definitions for the specified path name (`NW` and `CHTAG` are ignored).
- `'R'` Return the current mnemonic name definitions in `CHTAG`
- `'P'` Print the current mnemonic name definitions for objects stored in the directory `PATH` (`NW` and `CHTAG` are ignored)
- `'U'` Update the `DICTIONARY` directory with the specified mnemonic names.

`IRC` Integer return code

- `0` Normal completion
- `182` Illegal path name
- `183` Illegal number of data words
- `201` Dictionary directory not found
- `205` No mnemonic name definitions found

See page 6 for a discussion of `HEPDB` mnemonic names.

5.4 Help manipulation

5.4.1 Enter, delete or retrieve a help file

```
CALL CDHELP (LUN,CHFILE,PATH,CHOPT,IRC*)
```

`LUN` Logical unit on which to read or write the help file

`CHFILE` Character variable giving the file name

`PATH` Character string specifying the path name of the help directory

`CHOPT` Character variable specifying the options required.

- `'A'` File is already open on unit `LUN`
- `'D'` Delete the help information for the specified path name (`LUN` ignored)
- `'P'` Print the help information for the specified path name If `LUN` is non-zero, then the information will be printed on this unit. Otherwise, `IQPRNT` will be used.
- `'R'` Read the help information from the specified logical unit and enter into the database

	'W'	Write the help information for the specified path name onto the specified logical unit
IRC	Integer return code	
	0	Normal completion
	66	Illegal logical unit number
	182	Illegal path name
	203	No help directory in database
	204	No help information for specified path name

5.5 Text file manipulation

5.5.1 Enter or retrieve a text file

```
CALL CDTEXT (LUN,CHFILE,CHPATH,LBANK*,CHOPT,IRC*)
```

- `LUN` Fortran logical unit on which the file should be read or written
- `CHFILE` Character variable giving the file name
- `CHPATH` Character string specifying the path name
- `LBANK` Address of data bank (option `R`)
- `CHOPT` Character string with any of the following characters
  - 'A' File on unit `LUN` is already open
  - 'R' Read the text file from disk and convert to a bank at `LBANK`
  - 'P' Print the text from the bank at `LBANK`
  - 'W' Write the text from the bank at `LBANK` to the file `CHFILE`
- `IRC` Integer return code
  - 0 Normal completion
  - 66 Illegal logical unit number
  - 67 Array too long; no space in buffer

Subroutine `CDTEXT` can be used to read a text file from disk and convert it into a bank for insertion into the database with `CDSTOR`, or to write a file from information in an existing bank. Alternatively, the information from the bank may be printed in character format.

5.5.2 Copy character data to or from a bank

```
CALL CDCHAR (CHTEXT,NTEXT,LTEXT,CHPATH,LBANK,CHOPT,IRC*)
```

- `CHTEXT` Character array of length `NTEXT` in which the text is written (option `W`) or from which the text is read (option `R`)
- `NTEXT` Length of array `CHTEXT`
- `LTEXT` Maximum number of characters per element of `CHTEXT`

CHPATH Character string specifying the path name

LBANK Address of Keys bank KYDB (with option C)

CHOPT Character string with any of the following characters

- 'R' Read the text file from the character array CHTEXT and convert to a bank at LBANK
- 'P' Print the text from the bank at LBANK
- 'W' Write the text from the bank at LBANK to the character array CHTEXT

IRC Integer return code

- 0 Normal completion
- 61 Too many keys
- 66 Illegal logical unit number
- 67 File too long; no space in buffer

Subroutine CDCHAR is functionally similar to CDTEXT, but reads or writes the text from the array CHTEXT.

5.6 Vector manipulation

5.6.1 Enter or retrieve a vector

```
CALL CDVECT (CHPATH,IVECT,LVECT,LBANK,CHOPT,IRC*)
```

CHPATH Character string describing the pathname

IVECT Vector to be stored. In case of option G, IVECT is an output vector in which the data retrieved from the database is written.

LVECT Length of vector IVECT

LBANK Input address of the data bank (option G) or output address of the data bank (option P).

CHOPT Character string with any of the following characters

- 'G' Get the vector from the bank at LBANK
- 'P' Put the vector into the bank whose address is returned in LBANK
- 'B' Vector is bit string
- 'R' Vector is real
- 'I' Vector is integer
- 'D' Vector is double precision
- 'H' Vector is hollerith

IRC Integer return code

- 0 Normal completion
- 53 Path name does not exist in database
- 61 Too many keys
- 63 Database structure in memory clobbered
- 64 Error in MZCOPY while copying Data bank

Routine CDVECT can be used to store a vector in a bank suitable for insertion into the database or to retrieve data from an existing bank.

5.7 Directory manipulation

5.7.1 Creation

```
CALL CDMDIR (PATH,NKEYS,CHFOR,CHTAG,MAXOBJ,IPREC,DELTA,CHOPT,IRC*)
```

PATH Character string specifying the path name to be created

NKEYS Integer variable specifying the number of user keys, if any. The directory will automatically be created with the standard keys (system keys, experiment keys and the number of validity range pairs as specified when the database was created).

CHFOR Character string specifying the user key type, as for the routine RZMDIR

CHTAG Character array of significant length NKEYS containing the tags for the user keys

MAXOBJ Integer variable specifying the maximum number of objects to be stored in each partition of a partioned directory

IPREC Precision word specifying the number of significant digits to be stored (which may be zero). If IPREC is negative, then it specifies the number of digits to the left of the decimal point to retain.

DELTA Variable specifying the absolute value below which data is treated as zero

CHOPT
- ' ' Default - non-partitioned directory
- 'C' Data in this directory will be compressed by default
- 'P' Create partitioned directories

IRC Integer return code
- 0 Normal completion
- 43 Illegal number of user keys

This routine creates directories in a database and sets various directory constants.

5.7.2 Conversion

```
CALL CDPART (PATHI,PATHO,MXPART,CHOPT,IRC*)
```

PATHI Character string describing the input pathname

PATHO Character string describing the output pathname

MXPART Maximum number of objects in each partition

CHOPT Character string

IRC Integer return code
- 0 Normal completion
- 68 Input directory is already partitioned
- 71 Illegal path name
- 73 RZOUT fails to write on disk

74 Error in `RZRENK` in updating key values for data set
75 Cannot find the Top directory name in pathname
76 Cannot form the IO descriptor for the FZ header
77 `FZOUT` fails to write on to the sequential file

Transforms the contents of a non-partitioned directory to a partitioned directory.

5.7.3 Deletion

CALL **CDDDIR** (PATH,CHOPT,IRC*)

`PATH` Character string specifying the path name to be deleted

`CHOPT` Character variable specifying the options required.

`IRC` Integer return code

0 Normal completion
171 Illegal Path name
172 Cannot find the top directory for the path name
173 Error in RZ for reading the dictionary object
174 Error in `FZOUT` for saving the journal file
175 Error in RZ in writing the dictionary object
176 Error in RZ in purging the dictionary directory
177 Error in RZ in deleting the tree

5.7.4 Deleting partitions from a partitioned directory

CALL **CDPURP** (PATH,NKEEP,CHOPT,IRC*)

`PATH` Character array specifying the path names of the directory tree to be purged.

`NKEEP` Integer variable specifying the number of partitions to keep. Only the last `NKEEP` partitions will be kept.

`CHOPT` Character variable specifying the options required.

`IRC` Integer return code

0 Normal completion
69 Input directory is not partitioned
70 Error in deleting a partition
71 Illegal path name
73 `RZOUT` fails to write on disk
74 Error in `RZRENK` in updating key values for partitioned data set
75 Cannot find the Top directory name in pathname
77 `FZOUT` fails to write on to the sequential file

This routine can be used to delete partitions from a partitioned directory. Only the last `NKEEP` partitions will be kept.

5.7.5 Deleting multiple directory trees

```
CALL CDKEEP (PATH,NPATH,CHOPT,IRC*)
```

`PATH` Character array specifying the path names of the directory trees to be **kept**. All directory trees that are not in this list will be **deleted**.

`NPATH` Integer variable specifying the number of path names in the character array `PATH`.

`CHOPT` Character variable specifying the options required.

`IRC` Integer return code

- `0` Normal completion
- `69` Input directory is not partitioned
- `70` Error in deleting a partition
- `71` Illegal path name
- `73` RZOUT fails to write on disk
- `74` Error in `RZRENK` in updating key values for partitioned data set
- `75` Cannot find the Top directory name in pathname
- `77` `FZOUT` fails to write on to the sequential file

This routine can be used to delete all bar the specified directory trees from a database.

5.8 Storing information in the database

5.8.1 Storing single ZEBRA datastructures

```
CALL CDSTOR (PATH,LADDR,LKYBK*,IUDIV,KEYS,CHOPT,IRC*)
```

`PATH` Character string describing the pathname

`LADDR` Address of the datastructure to be stored **N.B. LADDR must be protected against relocations or garbage collection, e.g. by using a** *link area*.

`LKYBK` Returned address of the key bank (option `C`)

`IUDIV` Division where the datastructure resides

`KEYS` Vector of keys. This vector must include space for the standard system and experiment keys and validity range pairs. The user must fill in the validity range pairs and user keys, if any.

`CHOPT` Character string with any of the following characters

- `' '` Datastructure is stored asis, i.e. uncompressed
- `'C'` create Node/Key data structure ala `CDUSE`
- `'D'` store only differences from existing object in directory specified by `PATH`
- `'F'` With option `D`, differences are calculated from an object with which all user keys match (FULL match)
- `'K'` Store only the keys (`LADDR` ignored)

- `'H'` Insertion time in input `KEYS` vector is to be honoured
- `'P'` Data is compressed (bit packing)
- `'Y'` Store complete datastructure. **N.B. by default, only the bank at `LADDR` will be stored.**
- `'Z'` Store only nonzero elements. An element is considered to be zero if its absolute value is less than `DELTA` (a directory constant set by `CDMDIR` on page 50.

`IRC` Integer return code

- `0` Normal completion
- `53` Path name does not exist in database
- `61` Too many keys
- `63` Database structure in memory clobbered
- `64` Error in `MZCOPY` while copying Data bank

Routine `CDSTOR` stores a ZEBRA datastructure in the database. The data, if packed, is stored with a precision determined by the directory constant `IPREC` as specified in the call to `CDMDIR` on page 50

If the pathname begins with a % character then it is assumed to be an alias and is automatically translated by the internal `HEPDB` routines.

5.8.2 Storing multiple ZEBRA datastructures

```
CALL CDSTOM (PATH,LADDR,LKYBK*,IUDIV,NWKEY,NOBJ,KEYO,KEYN,CHOPT,IRC*)
```

`PATH` Character string describing the pathname

`LADDR` Vector of `NOBJ` bank addresses, created by `CDBOOK` **N.B. LADDR must be protected against relocations or garbage collection, e.g. by using a** *link area*.

`LKYBK` Output address of first key bank (if option `C` is specified)

`IUDIV` Division index of the user data bank

`NWKEY` Number of keys associated with the data banks

`NOBJ` Number of objects to be stored. The key vectors `KEYO` and `KEYN` must be dimensioned (`NWKEY`,`NOBJ`)

`KEYO` Vector/matrix of old keys

`KEYN` Vector/matrix of new keys

`CHOPT` Character string with any of the following characters

- `' '` Datastructure(s) are stored asis, i.e. uncompressed
- `'C'` create Node/Key data structure ala `CDUSE`
- `'D'` store only differences from existing object in directory specified by `PATH`
- `'F'` With option `D`, differences are calculated from an object with which all user keys match (FULL match)
- `'K'` Store only the keys (`LADDR` ignored)
- `'H'` Insertion time in input keys vector is to be honoured. The insertion time is stored in `KEYSN(KOFINS)`, packed using the routine `CDPKTM`.
- `'P'` Data is compressed (bit packing) according to the precision IPREC specified in the call to `CDMDIR`.

’Y’ Store complete datastructure. **N.B. by default, only the bank at LADDR will be stored.**

’Z’ Store only nonzero elements. An element is considered to be zero if its absolute value is less than DELTA (a directory constant set by CDMDIR on page 50

’R’ Replace existing objects as specified by the vector KEYO

IRC Integer return code

0 Normal completion

53 Path name does not exist in database

61 Too many keys

62 Too many keys with option N

63 Database structure in memory clobbered

64 Error in MZCOPY while copying Data bank

This routine permits multiple datastructures to be stored in a single call, or to replace one or more existing datastructures.

If the pathname begins with a % character then it is assumed to be an alias and is automatically translated by the internal HEPDB routines.

5.9 Retrieving information from the database

5.9.1 Retrieving ZEBRA datastructures

```
CALL CDUSE  (PATH,*LSUP*,ISEL,CHOPT,IRC*)
```

PATH Character string describing the pathname

LSUP Address of the supporting link of the Keys bank(s) KYDB (input or output) **N.B. LSUP must be protected against relocations or garbage collection, e.g. by using a** *link area*.

ISEL Integer vector specifying the instant of validity for which a database object is required. The length of the vector is equal to the number of validity range pairs, itself a database constant. For example, in the case when we select by run and event, the number of pairs is 2. ISEL then contains the specific run and event for which an object is required.

CHOPT Character string with any of the following characters

A Accept existing datastructure if one already exists in memory.

D Drop datastructure at LSUP before retrieving new datastructure

F Force retrieval of new datastructure even if current information as still valid

K read only the keys (no data is required)

IRC Integer return code

0 Successful operation

1 Illegal character option

2 Illegal path name

3 Database structure in memory clobbered

4 Illegal key option

5	Error in `DBCHLD` in `P3` communication
36	Data bank address zero on return from `DBKXIN`
37	Insufficient space in `USER` store array

Prepares the database data structure in memory for any required Pathname and set of Keys, unless already done. Returns (optionally) the addresses in memory for the corresponding Key banks and Data banks after checking their validity for the given time and keys.

If the pathname begins with a % character then it is assumed to be an alias and is automatically translated by the internal `HEPDB` routines.

Supplementary return information

`IQUEST(2)` Return status (if `IRC = 0`)

0	No disk I/O has been performed
1	Data have been refreshed from the disk

5.9.2 Retrieving multiple ZEBRA datastructures

`CALL` **CDUSEM** `(PATH,LSUP*,ISEL,IMASK,KEYS,CHOPT,IRC*)`

`PATH` Character string describing the pathname

`LSUP` Address of the supporting link of the Keys bank(s) `KYDB` **N.B. LSUP must be protected against relocations or garbage collection, e.g. by using a** *link area*.

`ISEL` Integer vector specifying the instant of validity for which a database object is required. The length of the vector is equal to the number of validity range pairs, itself a database constant. For example, in the case when we select by `run` and `event`, the number of pairs is 2. ISEL then contains the specific `run` and `event` for which an object is required.

`IMASK` Integer vector indicating which elements of `KEYS` are significant in selection. If an element of `IMASK` is non-zero, then the corresponding element of `KEYS` is used in selecting a dataset.

`KEYS` Vector of keys. Only the elements for which the corresponding element of `IMASK` is non-zero are assumed to contain useful information.
If MASK corresponding to one of the fields of 'Beginning' validity range is set, it will select objects with start validity smaller than those requested in KEYS. If MASK corresponding to one of the fields of 'End' validity range is set, it will select objects with end validity larger than those in KEYS. If MASK corresponding to time of insertion is set, objects inserted earlier than KEYS(IDHINS) are selected.

`CHOPT` Character string with any of the following characters

'A'	Accept existing datastructure if one already exists in memory. **N.B. if this option is specified, ANY datastructure in memory for this path and validity instant will be accepted, regardless of other selections specified.**
'K'	read only the keys (no data is required)
'D'	Drop datastructure at `LSUP` before retrieving new datastructure
'F'	Force retrieval of new datastructure even if current information is still valid

'M' Perform multiple selection: select according to **input** keys banks, rather than KEYS vector. The user must prepare a linear chain of key banks prior to calling this routine using the routine CDBOOK.

'N' If appropriate object does not exist, take nearest neighbour

'S' Select all objects satisfying the input selection criteria

IRC Integer return code

- 0 Successful operation
- 1 Illegal character option
- 2 Illegal path name
- 3 Database structure in memory clobbered
- 4 Illegal key option
- 5 Error in DBCHLD in P3 communication
- 36 Data bank address zero on return from DBKXIN
- 37 Insufficient space in USER store array

Prepares the database data structure in memory for any required Pathname and set of Keys, unless already done. Returns (optionally) the addresses in memory for the corresponding Key banks and Data banks after checking their validity for the given time and keys.

If the pathname begins with a % character then it is assumed to be an alias and is automatically translated by the internal HEPDB routines.

Supplementary return information

IQUEST(2) Return status (if IRC = 0)

- 0 No disk i/o has been performed
- 1 Data have been refreshed from the disk

5.9.3 Retrieving data structure to user specified address

```
CALL CDGET (PATH,IXDIV,LSUP*,JBIAS,ISEL,KEYS,CHOPT,IRC*)
```

PATH Character string describing the pathname

IXDIV Index of the division in which the data structure should be returned

LSUP Address of the data bank **N.B. LSUP must be protected against relocations or garbage collection, e.g. by using a** *link area*.

JBIAS LSUP and JBIAS together specify the address at which the structure is to be inserted. If JBIAS < 1 it is used as an offset to the value of LSUP, so the structure will be connected at LQ(LSUP+JBIAS). If JBIAS = 1 the data-structure will be a top level data-structure connected at LSUP. If JBIAS = 2 the data-structure will be created as a standalone structure

ISEL Integer vector specifying the instant of validity for which a database object is required. The length of the vector is equal to the number of validity range pairs, itself a database constant. For example, in the case when we select on run and event, the number of pairs is 2. ISEL then contains the specific run and event for which an object is required.

`KEYS` Integer vector in which the keys vector associated with the retrieved data object are returned.

`CHOPT` Character string with any of the following characters

- `'A'` Accept existing datastructure if one already exists in memory.
- `'K'` read only the keys (no data is required)
- `'D'` Drop datastructure at `LSUP` before retrieving new datastructure
- `'N'` Force retrieval of new datastructure even if current information is still valid

`IRC` Integer return code

- `0` Successful operation
- `1` Illegal character option
- `2` Illegal path name
- `3` Database structure in memory clobbered
- `4` Illegal key option
- `5` Error in `DBCHLD` in `P3` communication
- `36` Data bank address zero on return from `DBKXIN`
- `37` Insufficient space in `USER` store array

In some cases, for example in existing applications, one may wish to retrieve a data structure to a user specified location. In this case, `CDGET` may be used. Note that the data caching provided by `CDUSE` and `CDFREE` is not supported by this routine.

If the pathname begins with a % character then it is assumed to be an alias and is automatically translated by the internal `HEPDB` routines.

Supplementary return information

`IQUEST(2)` Return status (if `IRC = 0`)

- `0` No disk i/o has been performed
- `1` Data have been refreshed from the disk

5.10 Freeing information from memory

```
CALL CDFREE (PATH,LSUP,MASK,KEYS,CHOPT,IRC*)
```

`PATH` Character string describing the pathname

`LSUP` Address of Keys bank(s) `KYDB`

`MASK` Integer vector indicating which elements of KEYS are significant for selection.

`KEYS` Vector of keys

`CHOPT` Character string with any of the following characters

- `'A'` trust `LSUP` address if non-zero
- `'D'` drop the Data bank(s) supported at link 1 of Key bank(s)
- `'K'` drop the Key bank(s) as well as Data bank(s)

`IRC` Integer return code

- `0` Normal completion
- `51` Illegal character option
- `52` No access to the Key banks
- `53` Pathname not found in the RZ directory
- `54` Pathname not matched to that found in bank `NODB`
- `55` Too many keys with option `M`
- `56` Illegal Key option
- `57` Illegal pathname
- `58` Database structure in memory clobbered
- `59` Some of the expected key banks not found

Routine `CDFREE` declares the given data bank(s) as candidates to be dropped in case space is needed in the database division. Optionally it deletes the Data bank(s) (with option `D`) or the Keys as well as the Data bank(s) (with option `K`).
If the pathname begins with a % character then it is assumed to be an alias and is automatically translated by the internal `HEPDB` routines.

5.11 Deleting information from the database

5.11.1 Deleting a range of objects

`CALL` **CDPURG** `(PATH,KYDAT,KYTIM,CHOPT,IRC*)`

`PATH` Character string describing the pathname

`KYDAT` Key element number (for option `K` in `CHOPT`) or Minimum value of Key 1 to be deleted (for option `S`)

`KYTIM` Cutoff value for the key (for option `K`) or Maximum value of Key 1 to be deleted (for option `S`)

`CHOPT` Character string with any of the following characters

- `'A'` Deletes all data objects
- `'K'` Deletes all data objects for which `KEY(KYDAT)<KYTIM`
- `'L'` Deletes all but the last (one with highest `KEY(1)` value) data objects
- `'P'` Deletes all data objects with identical start and end validity but those having the highest Program Version number (i.e., `KEY(6)` value)
- `'S'` Deletes all data objects with Serial number (`KEY(1)`) in the range `KYDAT-KYTIM` (the terminal points included)

`IRC` Integer return code

- `0` Normal completion
- `111` Illegal path name
- `112` No key or data found for specified path

If the pathname begins with a % character then it is assumed to be an alias and is automatically translated by the internal `HEPDB` routines.

Supplementary return information

IQUEST(2) Number of objects deleted (if IRC = 0)

5.11.2 Deleting objects based on key vector

```
CALL CDPURK (PATH,ISEL,IMASK,KEYS,CHOPT,IRC*)
```

PATH Character string describing the pathname

ISEL Vector of length NPAIR specifying the validity instant. Only objects valid for this instant of validity are candidates for purging.

IMASK Integer vector specifying which elements of the KEYS vector are to be considered.

KEYS Vector of keys. Only the elements for which the corresponding element of IMASK is non-zero are assumed to contain useful information

CHOPT Character string with any of the following characters

- 'K' Delete object with key serial number as given in KEYS vector. IMASK is ignored.
- 'F' Require a full match of the entire KEYS vector. IMASK is ignored.
- 'U' Undelete a previously deleted object.

IRC Integer return code

- 0 Normal completion
- 111 Illegal path name
- 112 No key for the path name satisfying the Key assignments
- 113 Illegal character option
- 114 Valid data objects in the Node/Key structure

Routine CDPURK deletes objects in a directory path name steered by a selection on a number of key elements. The elements are specified by a non-zero value in the corresponding element of the IMASK vector. If the pathname begins with a % character then it is assumed to be an alias and is automatically translated by the internal HEPDB routines.

Supplementary return information

IQUEST(2) Number of objects deleted (if IRC = 0)

5.12 Modifying information in the database

5.12.1 Changing the keys for an existing object

```
CALL CDRENK (PATHN, KEYO, KEYN, IRC*)
```

PATHN Name of the directory containing the object

KEYO Old key vector

KEYN New key vector

IRC Integer return code

- 0 Normal completion
- 191 Illegal path name
- 192 Illegal KEYO values (no matching object)
- 194 Error in getting the IO descriptor
- 195 Error in FZOUT in saving the journal file
- 196 Error in RZRENK in renaming key value

5.13 Utility routines

5.13.1 Changing the HEPDB logging level

```
CALL CDLOGL (PATH,LOGLEV,CHOPT,IRC)
```

PATH Top directory name of HEPDB file.

LOGLEV Log level

- <0 no messages are printed
- 1 error messages from main HEPDB routines (default value)
- 2 error messages and warnings from main HEPDB routines
- 3 error messages, warnings and informative messages from main HEPDB routines
- 4 as above, but also error messages from HEPDB internal routines
- 5 as above, but also warnings from HEPDB internal routines
- >5 all messages are printed, and also additional debug.

CHOPT Character option

- 'A' Set RZ loglevel for all HEPDB files that are currently open

IRC Integer return code

- 0 Normal completion
- 1 Invalid path name
- 2 Database corresponding to specified path name is not open

The level of diagnostic print out can be set at any time using the routine CDLOGL. By default, the log level for each database file is set to 0 when the routine CDOPEN is called. Note that this routine sets both the HEPDB and RZ log levels.

5.13.2 Create a linear chain of keys banks

```
CALL CDBOOK (PATH,LSUP,NBANKS,CHOPT,IRC*)
```

`PATH`	Character variable specifying the pathname for which the banks are to be created
`LSUP`	Address of the first bank in the linear chain
`NBANKS`	Number of banks to create
`CHOPT`	Character options (not used at present)
`IRC`	Integer return code `0` Normal completion `99` Insufficient dynamic storage to create all of the requested banks

This routine creates a linear structure of `NBANKS` key banks. Each bank has a single structural link, to which a databank may eventually be attached.

5.13.3 Create information bank containing usage information

```
CALL CDINFO (IUDIV,LAD*,LSUP,JBIAS,IRC*)
```

`IUDIV`	User division where the `DBTB` bank has to be created
`LAD`	Address of the `DBTB` bank (should be in the same store as all DB objects)
`LSUP`	Address of the supporting bank
`JBIAS`	Link bias as described in ZEBRA manual
`IRC`	Integer return code `0` Normal completion

This routine creates the `DBTB` bank with information of the database objects used for this event (since the last call to `DBINFO`). It stores two words per object used, a unique identifier corresponding to the path name and the serial number of the object (`KEY(1)` value)

5.13.4 Print statistics on database usage

```
CALL CDSTAT (LUN,IRC*)
```

`LUN`	Logical unit on which the statistics should be printed
`IRC`	Integer return code `0` Normal completion `98` Invalid path name in node bank

This routine prints a summary of database usage for the current job on the specified logical unit.

5.13.5 List directory

```
CALL CDLDIR (PATH,LUN,NLEVEL,CHOPT,IRC*)
```

`PATH` Character string describing the pathname

`LUN` Integer variable specifying the Fortran logical unit on which the information should be printed

`NLEVEL` The number of levels of subdirectories that should be scanned.

`CHOPT` Character string with any of the following characters

- `H` Write a header showing the command and options
- `C` List the creation date and time of the directory/ies
- `M` List the modification date and time of the directory/ies
- `O` Display the number of objects
- `R` List subdirectories recursively
- `S` Display number of subdirectories
- `T` Display the tags for the directory/ies
- `V` Generate a ”very wide” listing (132 columns)
- `W` Generate a ”wide” listing (80 columns)

`IRC` Integer return code

- `0` Normal completion
- `101` Illegal path name
- `102` No key or data for specified path

5.13.6 List objects in a directory

```
CALL CDLIST (CHPATH,KSN,CHBANK,ILNK1,ILNK2,IDAT1,IDAT2,CHOPT,IRC)
```

`CHPATH` Character variable specifying the path name to list

`KSN` Key serial number of a specific object, or 0. If 0 is specified, no check will be made on key serial number.

`CHBANK` Character bank identifier (for option Z)

`ILNK1` First link to be printed (as in `DZSHOW`)

`ILNK2` Last link to be printed

`IDAT1` First data word to be printed

`IDAT2` Last data word to be printed

`CHOPT` Character string with any of the following characters

- `C` display object count
- `D` display key definitions
- `E` display the experiment keys
- `G` display keys using generic routine (RZPRNK)

K display all keys
L list only lowest level (end node) directories
M show maxima and minima of validity ranges
N display number of data words
P display pathname
S display the system keys
T display insertion date and time (RZ value)
U display user keys
V display validity range pairs
Z dump ZEBRA bank with DZSHOW

IRC

5.13.7 Display objects in a directory

```
CALL CDSHOW (CHPATH,ISEL,IMASK,KEYS,CHBANK,IDBANK,IDAT1,DAT2,CHOPT,IRC)
```

CHPATH Character variable specifying the path name to list

ISEL Integer vector specifying the instant of validity

IMASK Integer vector indicating which elements of KEYS are significant for selection. If MASK corresponding to one of the fields of 'Beginning' validity range is set, it will select objects with start validity smaller than those requested in KEYS. If MASK corresponding to one of the fields of 'End' validity range is set, it will select objects with end validity larger than those in KEYS. If MASK corresponding to time of insertion is set, objects inserted earlier than KEYS(IDHINS) are selected

KEYS Vector of keys.

CHBANK ZEBRA bank name

IDBANK ZEBRA bank identifier

IDAT1 First data word to be printed

IDAT2 Last data word to be printed

CHOPT Character string with any of the following characters

Find all banks with position in walk ¿ IDBANK
S FInd bank with Zebra ID = IDBANK

IRC

5.13.8 Obtain last object inserted into directory

```
CALL CDLKEY (PATH,KEY*,IDATE*,ITIME*,CHOPT,IRC*)
```

PATH Character string describing the pathname

`KEY` Scalar in which the highest KEY(1) (key serial number) is returned. If option `K` is specified, then `KEY` must be a vector of sufficient length to retrieve the entire key vector for objects in the specified directory. The length of the key vector may be obtained using the routine `RZKEYD`.

`IDATE` Date (`YYMMDD`) of the insertion of the last element

`ITIME` Time (`HHMM`) of the insertion of the last element

`CHOPT` Character string with any of the following characters

- `' '` Return just the key serial number (`KEY(1)`)
- `'K'` Return complete key vector in `KEY`

`IRC` Integer return code

- `0` Normal completion
- `131` Illegal pathname
- `132` Illegal number of keys in the directory

This routine returns the key serial number (optionally the complete key vector) of the last object inserted into the specified directory. If the option `K` is specified, then the vector `KEY` must be of sufficient size as to receive the complete vector.

5.13.9 Date of last directory modification

```
CALL CDLMOD (PATH,IDATE*,ITIME*,CHOPT,IRC*)
```

`PATH` Character variable specifying the path name of interest

`IDATE` Date of last modification in `YYMMDD` format

`ITIME` Time of last modification in `HHMM` format

`IRC` Integer return code

- `0` Normal completion
- `131` Invalid path name

This routine returns the date and time of the last modification of the specified directory.

5.13.10 Pack and unpack date and time

Two sets of routines are provided: `CDPKTM` and `CDUPTM`, for storing times with 1 minute precision, and `CDPKTS` and `CDUPTS` which provide 1 second precision. In the case of the latter two routines, only times relating to 1980 and after may be stored.

```
CALL CDPKTM (IDATE,ITIME,IPACK*,IRC*)
```

`CALL` **CDPKTS** `(IDATE,ITIME,IPACK*,IRC*)`

`IDATE` Integer variable with date in `YYMMDD` format

`ITIME` Integer variable with time in `HHMM` format for `CDPKTM` and in `HHMMSS` format for `CDPKTS`.

`IPACK*` Integer variable to store the date and time in packed format.

`IRC` Integer return code

- `0` Normal completion
- `93` Illegal date or time

`CALL` **CDUPTM** `(IDATE*,ITIME*,IPACK,IRC*)`

`CALL` **CDUPTS** `(IDATE*,ITIME*,IPACK,IRC*)`

`IDATE*` Integer variable to store the date in `YYMMDD` format.

`ITIME*` Integer variable with time in `HHMM` format for `CDUPTM` and in `HHMMSS` format for `CDUPTS`.

`IPACK` Integer variable with date and time in packed format.

`IRC` Integer return code

- `0` Normal completion
- `93` Illegal packed time

These routines allow a date and time to be stored in, and retrieved from a 4 byte integer word. `IPACK` must be the result of a previous call to `CDPKTM`, or is the output packed time. The CERNLIB routine `DATIME`, entry `Z007`, can be used to obtain `IDATE` and `ITIME` in the correct format. The time can be obtained in the `HHMMSS` format required by `CDPKTS` as shown in the following example.

Example of obtaining the time in `HHMMSS` format

```
*
*     Subroutine DATIME returns the date and time in the format
*     datime(id,it) ID=YYMMDD, e.g. 930425, IT=hhmm, e.g. 1230
*     Additional information is also returned in the common SLATE,
*     e.g. IS(6) = seconds
*
      COMMON/SLATE/IS(40)

      CALL DATIME(ID,IT)
      IT = IT*100 + IS(6)
```

5.14 Ranges or values for selection by keys

5.14.1 Retrieving ranges for key pairs

```
CALL CDVALI (IVECT*,IRC*)
```

IVECT Vector returning the minimum and maximum values for the various key pairs since the last call to CDINFO

IRC Integer return code

- 0 Normal completion

This routine returns the overlapping validity ranges for the various key pairs since the last call to CDINFO

5.14.2 Setting the historical retrieval time

```
CALL CDBFOR (TOPN,IDATE,ITIME,IRC*)
```

TOPN Name of the top directory ('*' means all)

IDATE Date : 6 Decimal integer : YYMMDD

ITIME Time : 4 Decimal integer : HHMM

IRC Integer return code

- 0 Normal completion

Set the maximum insertion time for retrieval of all subsequent data objects for a given top directory Only objects inserted before the specified time will be retrieved, until a further call to CDBFOR is made.

Chapter 6: Description of command line interface

A simple shell interface to HEPDB is provided, using the KUIP [15] package. To run this interface, simply type `hepdb`.

COUNT `[ PATH CHOPT ]`

`PATH` Character variable specifying the pathname in which the objects are to be counted.

`CHOPT` Character variable specifing the options required

- `D` display number of subdirectories at each level
- `O` display number of objects at each level
- `L` display lowest level only, i.e. directories with no subdirectories
- `Z` display only directories with no (zero) objects

Use the `COUNT` command to count the number of objects in the specified directories.

FILES

Use the `FILES` command to display the files that are currently open.

LOGLEVEL `[ PATH LOGLEVEL CHOPT ]`

`PATH` Top directory name of HEPDB file.

`LOGLEV` Log level

- `<0` no messages are printed
- `1` error messages from main HEPDB routines (default value)
- `2` error messages and warnings from main HEPDB routines
- `3` error messages, warnings and informative messages from main HEPDB routines
- `4` As above, but also error messages from HEPDB internal routines
- `5` As above, but also warnings from HEPDB internal routines
- `>5` all messages are printed, and also additional debug.

`CHOPT` Character option

- `A` Set RZ loglevel for all HEPDB files that are currently open

Use the `LOGLEVEL` command to set the HEPDB logging level.

OPEN `PREFIX [ CHFILE ] [ CHOPT ]`

`PREFIX` Character variable specifying the two character database prefix

`CHFILE` Character variable specifying the file name

`CHOPT` Character option, as for the `CDOPEN` routine.

Use the `OPEN` command to open a `HEPDB` file

CLOSE PREFIX [CHOPT]

PREFIX Character variable specifying the database prefix of the database to be cloed.

CHOPT Character option, as for the CDEND routine.

Use the CLOSE command to close a HEPDB file. previously opened using the OPEN command.
The CLOSE command accepts a two character database prefix, top directory name, or pathname. Thus, for a file with database prefix BK, the following commands are identical.

```
CLOSE BK

CLOSE CDBK

CLOSE //CDBK
```

RZOPEN CHTOP CHFILE [CHOPT]

CHTOP Character variable specifying the top directory name

CHFILE Character variable specifying the file name

CHOPT Character option

Use the RZOPEN command to open an RZ file

RZCLOSE CHTOP [CHFILE] [CHOPT]

CHTOP Character variable specifying the top directory name

CHFILE Character variable specifying the file name

CHOPT Character option

- A Close all files

Use the RZCLOSE command to close a file previously opened with RZOPEN

OUTPUT [FILE CHOPT]

FILE Character variable specifying the name of the file

CHOPT Character options

- C close file and redirect output to terminal
- P preserve case of file
- R replace existing file
- S switch back to previously opened file
- T redirect output back to terminal

Use the OUTPUT command to redirect output to a specified file or to the terminal.

VERSION

Use the VERSION command to display the version of the HEPDB software that you are running.

CD [CHPATH] [CHOPT]

CHPATH Character variable specifying the name of the new directory

CHOPT Character option

- Q show quota for new directory
- S show number of subdirectories
- T show creation and modification times
- U show usage information
- A all of the above

Use the CD command to change the current working directory.

If the pathname begins with a % character then it is assumed to be an alias and is automatically translated by the internal HEPDB routines.

LD [CHPATH NLEVEL CHOPT]

CHPATH Character variable specifying the pathname

KSN Serial number of the object to be displayed

NLEVEL Number of levels to display

CHOPT Character options

- C Display the RZ creation date and time
- M Display the RZ modification date and time
- O List the number of objects at each level
- R List subdirectories recursively
- S Display the number of subdirectories at each level
- T Display the tags as specified in the call to RZMDIR

Use the LD command to display subdirectories below the specified level.

LS [CHPATH KSN BANK CHOPT]

CHPATH Character variable specifying the path name to be listed

BANK Character variable specifying the name of the bank to display

ILNK1 Index of the first link to be printed

ILNK2 Index of the last link to be printed

IDAT1 Index of the first word to be printed

IDAT2 Index of the last word to be printed

`CHOPT` Character options

- `C` display object count
- `D` display key definitions
- `S` display the system keys
- `E` display the experiment keys
- `G` display keys using generic routine (`RZPRNK`)
- `V` display validity range pairs
- `U` display user keys
- `K` display all keys
- `L` list only lowest level (end node) directories (default)
- `M` show maxima and minima of validity range pairs
- `N` display number of data words
- `P` display pathname (default)
- `T` display insertion date and time (RZ value)
- `Z` dump ZEBRA bank with `DZSHOW`

Use the `LS` command to display the contents of a directory. If option `Z` is specified, the name of the bank(s) to be displayed may be given. If multiple banks are to be displayed, their names should be separated by commas. Wild cards are permitted in bank names.

PWD

Use the `PWD` command to print the current (working) directory.

`CHOPT` Character options

- `A` display alias name for current directory

STATUS `[ CHPATH NLEVEL CHOPT ]`

`CHPATH` Character variable specifying the pathname

`NLEVEL` Number of levels to display

`CHOPT` Character options as for `RZSTAT`

Use the `STATUS` command to print usage statistics on the specified directory down `NLEVEL` levels.

TREE `[ CHPATH NLEVEL ]`

`CHPATH` Character variable specifying the pathname

`NLEVEL` Number of levels to display

`CHOPT` Character options

- `A` show alias name (if any) for each directory
- `N` show the number of objects for each directory if non-zero
- `O` show the number of objects for each directory
- `S` show the number of subdirectories for each directory
- `C` show the date and time the directory was created
- `M` show the date and time of the last modification (RZ)

Use the `TREE` command to draw a directory tree starting at the specified directory down `NLEVEL` levels.

SELECT `ISEL1 [ ISEL2 ISEL3 ]`

`ISEL1` Integer variable specifying the primary selection

`ISEL2` Integer variable specifying the secondary selection

`ISEL3` Integer variable specifying the tertiary selection

Use the `SELECT` command to specify the instant of validity for which objects are required.

ZOOM `PATH`

`CHPATH` Character variable specifying the pathname

Use the `ZOOM` command to descend the specified directory tree to the first lowest level directory that contains one or more entries. The directory specification may contain wild-cards.

DIR `[ CHPATH CHOPT ]`

`CHPATH` Character variable specifying the pathname

`CHOPT` Character variable specifying the 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.

EXTRACT `[ CHPATH OUTPUT CHOPT ]`

`CHPATH` Character variable specifying the pathname

`OUTPUT` Character variable specifying the output filename

`CHOPT` Character variable specifying the options

Use the `EXTRACT` command to copy a subset of the HEPDB catalogue

MERGE `[ INPUT CHPATH CHOPT ]`

`INPUT` Character variable specifying the input file name

`CHPATH` Character variable specifying the pathname

`CHOPT` Character variable specifying the options

Use the `MERGE` command to merge an update file created by the `EXTRACT` command into the specified path.

MKDIR `CHPATH`

`CHPATH` Character variable specifying the pathname

Use the MKDIR command to create a directory

RM `KEY1`

`KEY1` Integer variable specifying the key serial number of the object to be deleted

RMDIR `CHPATH`

`CHPATH` Character variable specifying the pathname

Use the `RMDIR` command to remove a directory from the catalogue.
Note that directories can only be removed if:

1 They contain no subdirectories
2 They contain no entries

RMTREE `CHPATH`

`CHPATH` Character variable specifying the pathname

Use the `RMTREE` command to remove a complete directory tree.
Note that if any of the directories below the named directory contain entries, then the command will be refused.

Appendix A: Conversion of existing database files

Existing database files may be converted to HEPDB format using a simple program. The following examples describe the conversion of a file that is already in Zebra RZ format and the conversion of a file that does not use Zebra RZ.

A.1 Conversion of the CPLEAR calibration database

The CPLEAR calibration database consists of a single Zebra RZ file containing a number of directories corresponding to the long term, medium term and short term calibration constants of the various subdetectors.

Information is stored in these directories as individual Zebra banks, identified by the directory name and four keys. These keys contain the following information:

VAL_STAR	Run number defining the lower bound of the validity range (integer)
VAL_STOP	Run number defining the upper bound of the validity range (integer)
DETECTOR	Detector name to which the information corresponds (hollerith)
POINTER	Hollerith bank identifier of the Zebra bank (hollerith)

This corresponds to a HEPDB database with one validity range pair. The detector and pointer information are stored as user keys.

The first thing that must be performed is the creation of a new HEPDB database file. This is performed by the following program.

Creating a new HEPDB database for CPLEAR

```
*CMZ :          23/10/92  10.16.19  by  Jamie Shiers
*-- Author :
      PROGRAM CDEXA1
*     ==============
*
*     Create a new, empty database
*
      PARAMETER   (NWPAW=100000)
      COMMON/PAWC/PAW(NWPAW)
*
*     Initialise Zebra, HBOOK and HEPDB
*
      CALL CDPAW(NWPAW,NHBOOK,IDIV,'USR-DIV',5000,50000,'ZPHU',IRC)
*
*     Unit for database access
*
      LUNCD  = 1
*
*     Database parameters
*
      NPAIR  = 1
      NREC   = 20000
      NPRE   = 200
      NTOP   = 1
      NQUO   = 65000
```

```
*
*     Accept default record length (1024 words)
*
      LRECL  = 0
      CALL CDNEW(LUNCD,'HEPDB','RZKAL.DBS',IDIV,NPAIR,NQUO,NREC,NPRE,NTOP,
     +           LRECL,'F',IRC)
*
*     Set the log level
*
      CALL CDLOGL(' ',3,'A',IRC)
*
*     Terminate
*
      CALL CDEND(' ','A',IRC)

      END
```

The following program shows how the directory structure is created in the HEPDB database. Note that the directory structure is somewhat simplified in the conversion, but this is of course optional.

Creating the directory structure in the HEPDB database

```
CDECK  ID>, KALCONV.
      PROGRAM KALCONV
*
*     Program to convert CPLEAR calibration database
*     to HEPDB format
*
*     RZKAL keys: VAL_STAR (I)
*                 VAL_STOP (I)
*                 DETECTOR (H)
*                 BANK ID  (H)
*     insertion time = RZ date/time
*
*     HEPDB keys: NPAIR    = 1
*                 VAL_STAR = KEYS(11) (I)
*                 VAL_STOP = KEYS(12) (I)
*                 NUSER    = 2
*                 DETECTOR = KEYS(13) (H)
*                 BANK ID  = KEYS(14) (H)
*     insertion time = KEYS(IDHINS)
*
*     Output pathnames:
*
*     //CDCD/CALIBRATION/DC_ST
*     //CDCD/CALIBRATION/DC_LT
*     //CDCD/CALIBRATION/DC_MT
*
*     //CDCD/CALIBRATION/PC_ST
*     //CDCD/CALIBRATION/PC_LT
*     //CDCD/CALIBRATION/PC_MT
*
*     //CDCD/CALIBRATION/PID_ST
*     //CDCD/CALIBRATION/PID_LT
*     //CDCD/CALIBRATION/PID_MT
*
```

```
*     //CDCD/CALIBRATION/ST_MT
*
*     //CDCD/CALIBRATION/CALO_LT
*
      PARAMETER    (NWPAW=100000)
      COMMON/PAWC/ PAW(NWPAW)
      COMMON/USRLNK/IDIV,LADDR
      CHARACTER*4  CHTOP
      CHARACTER*80 CHFILE
      EXTERNAL     CPKALC
*
*     Initialise Zebra, HBOOK and HEPDB
*
      CALL CDPAW(NWPAW,NHBOOK,IDIV,'USR-DIV',5000,50000,'ZPHU',IRC)
*
*     Link area of banks retrieved from database
*
      CALL MZLINK(IDIV,'/USRLNK/',LADDR,LADDR,LADDR)
*
*     Unit for database access
*
      LUNCD  = 1
*
*     Unit for database update (via journal files)
*
      LUNFZ  = 2
*
*     Unit for RZKAL file
*
      LUNRZ  = 3
*
*     Open CPLEAR calibration database (RZKAL.DATA)
*
      LRECL  = 0
      CALL RZOPEN(LUNRZ,'RZKAL','rzkal.data',' ',LRECL,IRC)
      CALL RZFILE(LUNRZ,'RZKAL',' ')
*
*     Find the database file and construct the top directory name
*
      CALL CDPREF(10,'CD',CHTOP,CHFILE,IRC)
*
*     Open the database file
*
      LRECL  = 0
      CALL CDOPEN(LUNCD,LUNFZ,CHTOP,CHFILE,LRECL,IDIV,' ',IRC)
*
*     Loop over directories in RZKAL.DATA
*
      CALL RZSCAN('//RZKAL',CPKALC)
*
*     Terminate
*
      CALL CDEND(' ','A',IRC)
      CALL RZCLOS(' ','A')

      END
CDECK  ID>, CPKALC.
```

```
      SUBROUTINE CPKALC(CHDIR)
      CHARACTER*(*) CHDIR
      PARAMETER    (NKEYS=2)
      PARAMETER    (MAXOBJ=1000)
      CHARACTER*8   CHTAG(NKEYS)
      CHARACTER*2   CHFOR
      CHARACTER*255 CHPATH,CHSAVE
      DATA          NENTRY/0/
      SAVE          NENTRY

      IF(NENTRY.EQ.0) THEN
         NENTRY = 1
         RETURN
      ENDIF
*
*     Must save directory in local variable: calls to RZ
*     overwrite it!
*
      LDIR   = LENOCC(CHDIR)
      CHSAVE = CHDIR(1:LDIR)
*
*     Make directories in HEPDB database
*
      DELTA = 0.0
      IPREC = 0
      CHFOR = 'HH'
      CHTAG(1) = 'DETECTOR'
      CHTAG(2) = 'POINTER '
*
*     Construct directory name for HEPDB file
*
      LSLASH   = INDEXB(CHSAVE(1:LDIR),'/') + 1
      IF(INDEX(CHSAVE(1:LDIR),'MONTE').EQ.0) THEN
         CHPATH   = '//CDCD/CALIBRATION/'//CHSAVE(LSLASH:LDIR)
         LPATH    = LDIR - LSLASH + 20
      ELSE
         CHPATH   = '//CDCD/'//CHSAVE(LSLASH:LDIR)
         LPATH    = LDIR - LSLASH + 8
      ENDIF
      CALL CDMDIR(CHPATH(1:LPATH),NKEYS,CHFOR,CHTAG,MAXOBJ,
     +            IPREC,DELTA,'CP',IRC)

   99 CONTINUE

      CALL RZCDIR(CHSAVE(1:LDIR),' ')

      END
```

The data is then entered using a program that is very similar to the above.

Entering the data into the HEPDB database

```
CDECK  ID>, KALCONV.
      PROGRAM KALCONV
*
*     Program to convert CPLEAR calibration database
```

```
*     to HEPDB format
*
*     RZKAL keys: VAL_STAR (I)
*                 VAL_STOP (I)
*                 DETECTOR (H)
*                 BANK ID  (H)
*     insertion time = RZ date/time
*
*     HEPDB keys: NPAIR    = 1
*                 VAL_STAR = KEYS(11) (I)
*                 VAL_STOP = KEYS(12) (I)
*                 NUSER    = 2
*                 DETECTOR = KEYS(13) (H)
*                 BANK ID  = LEYS(14) (H)
*     insertion time = KEYS(IDHINS)
*
*     Output pathnames:
*
*     //CDCD/CALIBRATION/DC_ST
*     //CDCD/CALIBRATION/DC_LT
*     //CDCD/CALIBRATION/DC_MT
*
*     //CDCD/CALIBRATION/PC_ST
*     //CDCD/CALIBRATION/PC_LT
*     //CDCD/CALIBRATION/PC_MT
*
*     //CDCD/CALIBRATION/PID_ST
*     //CDCD/CALIBRATION/PID_LT
*     //CDCD/CALIBRATION/PID_MT
*
*     //CDCD/CALIBRATION/ST_MT
*
*     //CDCD/CALIBRATION/CALO_LT
*
      PARAMETER    (NWPAW=100000)
      COMMON/PAWC/ PAW(NWPAW)
      COMMON/USRLNK/IDIV,LADDR
      CHARACTER*4  CHTOP
      CHARACTER*80 CHFILE
      EXTERNAL     CPKALC
*
*     Initialise Zebra, HBOOK and HEPDB
*
      CALL CDPAW(NWPAW,NHBOOK,IDIV,'USR-DIV',5000,50000,'ZPHU',IRC)
*
*     Link area of banks retrieved from database
*
      CALL MZLINK(IDIV,'/USRLNK/',LADDR,LADDR,LADDR)
*
*     Unit for database access
*
      LUNCD  = 1
*
*     Unit for database update (via journal files)
*
      LUNFZ  = 2
*
```

```
*     Unit for RZKAL file
*
      LUNRZ  = 3
*
*     Open CPLEAR calibration database (RZKAL.DATA)
*
      LRECL  = 0
      CALL RZOPEN(LUNRZ,'RZKAL','rzkal.data',' ',LRECL,IRC)
      CALL RZFILE(LUNRZ,'RZKAL',' ')
*
*     Find the database file and construct the top directory name
*
      CALL CDPREF(10,'CD',CHTOP,CHFILE,IRC)
*
*     Open the database file
*
      LRECL  = 0
      CALL CDOPEN(LUNCD,LUNFZ,CHTOP,CHFILE,LRECL,IDIV,' ',IRC)
*
*     Loop over directories in RZKAL.DATA
*
      CALL RZSCAN('//RZKAL',CPKALC)
*
*     Terminate
*
      CALL CDEND(' ','A',IRC)
      CALL RZCLOS(' ','A')

      END
CDECK  ID>, CPKALC.
      SUBROUTINE CPKALC(CHDIR)
      CHARACTER*(*) CHDIR
      COMMON/USRLNK/IDIV,LADDR
      PARAMETER    (NKEYS=2)
      PARAMETER    (MAXOBJ=1000)
      DIMENSION     KEYS(13)
      DIMENSION     KEYZ(4)
      CHARACTER*8   CHTAG(NKEYS)
      CHARACTER*2   CHFOR
      CHARACTER*255 CHPATH,CHSAVE
      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 (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
      DATA          NENTRY/0/
      SAVE          NENTRY

      IF(NENTRY.EQ.0) THEN
         NENTRY = 1
         RETURN
      ENDIF
*
*     Must save directory in local variable: calls to RZ
*     overwrite it!
*
      LDIR   = LENOCC(CHDIR)
      CHSAVE = CHDIR(1:LDIR)
*
*     Retrieve the keys in this directory
*
      IF(LQRS.EQ.0)  GOTO 99
      IF(LCDIR.EQ.0) GOTO 99
      LS = IQ(KQSP+LCDIR+KLS)
      LK = IQ(KQSP+LCDIR+KLK)
      NK = IQ(KQSP+LCDIR+KNKEYS)
      NWK= IQ(KQSP+LCDIR+KNWKEY)
      DO 10 I=1,NK

         K=LK+(NWK+1)*(I-1)
         DO 20 J=1,NWK
            IKDES=(J-1)/10
            IKBIT1=3*J-30*IKDES-2
            IF(JBYT(IQ(KQSP+LCDIR+KKDES+IKDES),IKBIT1,3).LT.3)THEN
               KEYZ(J)=IQ(KQSP+LCDIR+K+J)
            ELSE
               CALL ZITOH(IQ(KQSP+LCDIR+K+J),KEYZ(J),1)
            ENDIF
   20    CONTINUE

         CALL VZERO(KEYS,10)
```

```
         CALL UCOPY(KEYZ(1),KEYS(11),4)
*
*     Retrieve the highest cycle of this object
*     (will need modification if all cycles are to be converted)
*
         ICYCLE = 9999
         JBIAS = 2
         CALL RZIN(IDIV,LADDR,JBIAS,KEYZ,ICYCLE,' ')
         IF(IQUEST(1).NE.0) THEN
            PRINT *,'CPKALC. error ',IQUEST(1),' from RZIN for ',KEYZ
            GOTO 10
         ENDIF
*
*     Date/time of insertion
*
         CALL RZDATE(IQUEST(14),IDATE,ITIME,1)
         CALL CDPKTM(IDATE,ITIME,IPACK,IRC)
         KEYS(4) = IPACK
*
*     Store objects in HEPDB with appropriate keys
*     Option H: honour insertion time in KEYS(IDHINS)
*
         CALL CDSTOR(CHPATH(1:LPATH),LADDR,LKYBK,IDIV,KEYS,'H',IRC)
*
*     Reset directory
*
         CALL RZCDIR(CHSAVE(1:LDIR),' ')
*
*     Drop this bank
*
         CALL MZDROP(IDIV,LADDR,' ')
         LADDR = 0

   10 CONTINUE

   99 CONTINUE
*
*     Send updates to server one directory at a time
*
      CALL CDSTSV(' ',0,IRC)
      CALL RZCDIR(CHSAVE(1:LDIR),' ')

      END
```

A.2 Creation of the CHORUS database

The programs in this section were all written by `J. Brunner/CHORUS`.

The following program shows the creation of the directory structure and aliases for the CHORUS geometry database.

Once the directories have been created by the server, the program can be rerun to enter the aliases.

Creating the directories and aliases for CHORUS

```
      PROGRAM MKDIRHDB
C     ----------------
C     CREATES THE DIRECTORY STRUCTURE
C     FOR THE GEOMETRY DATABASE OF CHORUS
C
      PARAMETER (NPAW=100000,NHBOOK=0,NDX=43)
      COMMON /PAWC/ PAW(NPAW)
      CHARACTER*4 CHTOP
      CHARACTER*80 CHFILE
      CHARACTER*80 DNAME
      CHARACTER*40 DITAG(NDX)
      CHARACTER*4  ALIAS(NDX)
      DATA ALIAS /'3021','3022','3023','3024','3061','3062','3063',
     +'3041','3042','3043','3044','3051','3052','3053','3054',
     +'3011','3012','3015','3016','3017','3014','3018',
     +'3031','3032','3033','3034','3091',
     +'3071','3072','3073','3074','3075','3076','3077','3078',
     +'3081','3082','3083','3084','3085','3086','3087','3088'/
      DATA DITAG /'TUBES/X-COORD',
     +            'TUBES/V-COORD',
     +            'TUBES/V-OFFSET',
     +            'TUBES/ORIENTATION',
     +            'TUBES/ANALOG-V-COORD',
     +            'TUBES/ANALOG-V-OFFSET',
     +            'TUBES/MAGNET',
     +            'BREMS/X-COORD',
     +            'BREMS/V-COORD',
     +            'BREMS/V-OFFSET',
     +            'BREMS/ORIENTATION',
     +            'DRIFT/X-COORD',
     +            'DRIFT/V-COORD',
     +            'DRIFT/V-OFFSET',
     +            'DRIFT/ORIENTATION',
     +            'CALOR/X-COORD',
     +            'CALOR/V-COORD',
     +            'CALOR/ELM-V-OFFSET',
     +            'CALOR/HA1-V-OFFSET',
     +            'CALOR/HA2-V-OFFSET',
     +            'CALOR/ORIENTATION',
     +            'CALOR/MASK',
     +            'FIBER/X-COORD',
     +            'FIBER/V-COORD',
     +            'FIBER/V-OFFSET',
     +            'FIBER/ORIENTATION',
     +            'DIAMO/X-COORD',
     +            'TRIGG/X-COORD-PLAN',
     +            'TRIGG/Y-COORD-PLAN',
     +            'TRIGG/Z-COORD-PLAN',
     +            'TRIGG/X-WIDTH-PLAN',
     +            'TRIGG/Y-WIDTH-PLAN',
     +            'TRIGG/Z-WIDTH-PLAN',
     +            'TRIGG/Z-ANGLE-PLAN',
     +            'TRIGG/POINTER-TO-BAR',
     +            'TRIGG/X-COORD-BAR',
     +            'TRIGG/Y-COORD-BAR',
     +            'TRIGG/Z-COORD-BAR',
     +            'TRIGG/X-WIDTH-BAR',
```

```
     +            'TRIGG/Y-WIDTH-BAR',
     +            'TRIGG/Z-WIDTH-BAR',
     +            'TRIGG/Z-ANGLE-BAR',
     +            'TRIGG/POINTER-TO-PLAN'/
C
C---      INITIALISATION
C
      CALL CDPAW(NPAW,NHBOOK,IDIV,'USR-DIV',5000,50000,'ZPHU',IRC)
      PRINT '('' IRC FROM CDPAW '',I5)',IRC
      LUNCD=1
      LUNFZ=2
      CALL CDPREF(10,'CH',CHTOP,CHFILE,IRC)
      PRINT '('' IRC FROM CDPREF '',I5)',IRC
      LRECL = 0
      CALL CDOPEN(LUNCD,LUNFZ,CHTOP,CHFILE,LRECL,IDIV,' ',IRC)
      PRINT '('' IRC FROM CDOPEN '',I5)',IRC
C
C--- CREATE DIRECTORIES
C
      IPREC = -8
      MAX   = 100
      DELTA = 0.0
      NKEYS = 0
      DO IDX=1,NDX
      DNAME = '//CDCH/GEOMETRY/'//DITAG(IDX)
*
*     First run with the following call to CDMDIR
*
      CALL CDMDIR(DNAME,NKEYS,' ',' ',MAX,IPREC,DELTA,' ',IRC)
      PRINT '('' IRC FROM CDMDIR '',I5)',IRC
*
*     Then rerun with the following call uncommented and
*     the previous call to CDMDIR commented out
*
*     CALL CDALIA(DNAME,ALIAS(IDX),'P',IRC)
*     PRINT '('' IRC FROM CDALIA '',I5)',IRC
      END DO
C
C--- TERMINATION
C
      CALL CDEND(' ','A',IRC)
      END
```

The following program shows an example of how the directories created by the previous program can be populated with vectors.

HEPDB always stores objects as Zebra banks and so the first operation is to convert the vectors into banks using the routine CDVECT. The banks can then be stored using CDSTOR.

Storing vectors in a HEPDB database

```
      PROGRAM FILLHDB
C     ----------------
C     FILLS THE DIRECTORY STRUCTURE
C     FOR THE GEOMETRY DATABASE OF CHORUS
C
```

```
      DIMENSION KEYDBS(100)
      PARAMETER (NPAW=400000,NHBOOK=0,NDX=42)
      COMMON /PAWC/ PAW(NPAW)
      CHARACTER*4 CHTOP
      CHARACTER*80 CHFILE
      CHARACTER*80 DNAME
      CHARACTER*40 DITAG(NDX)
      CHARACTER*4  ALIAS(NDX)
      DIMENSION IPO(1300)
      DATA (IPO(L),L=1,90)/
     + 13633, 13634, 13635, 13636, 13637, 13638, 13639, 13640,     0,
     +     0, 13641, 13642, 13643, 13644, 13645, 13646, 13647, 13648,
     + 13649, 13650, 13651, 13652, 13653, 13654, 13655, 13656, 13657,
     + 13658, 13659, 13660, 13661, 13662, 13663, 13664, 13665, 13666,
     + 13667, 13668,     0,     0, 13377, 13378, 13379, 13380, 13381,
     + 13382, 13383, 13384,     0,     0, 13385, 13386, 13387, 13388,
     + 13389, 13390, 13391, 13392, 13393, 13394, 13395, 13396, 13397,
     + 13398, 13399, 13400, 13401, 13402, 13403, 13404, 13405, 13406,
     + 13407, 13408, 13409, 13410, 13411, 13412,     0,     0, 13121,
     + 13122, 13123, 13124, 13125, 13126, 13127, 13128,     0,     0/
      data (ipo(L),L=91,180)/
     + 13129, 13130, 13131, 13132, 13133, 13134, 13135, 13136, 13137,
     + 13138, 13139, 13140, 13141, 13142, 13143, 13144, 13145, 13146,
     + 13147, 13148, 13149, 13150, 13151, 13152, 13153, 13154, 13155,
     + 13156,     0,     0, 12865, 12866, 12867, 12868, 12869, 12870,
     + 12871, 12872,     0,     0, 12873, 12874, 12875, 12876, 12877,
     + 12878, 12879, 12880, 12881, 12882, 12883, 12884, 12885, 12886,
     + 12887, 12888, 12889, 12890, 12891, 12892, 12893, 12894, 12895,
     + 12896, 12897, 12898, 12899, 12900,     0,     0, 12609, 12610,
     + 12611, 12612, 12613, 12614, 12615, 12616,     0,     0, 12617,
     + 12618, 12619, 12620, 12621, 12622, 12623, 12624, 12625, 12626/
      data (IPO(L),L=181,270)/
     + 12627, 12628, 12629, 12630, 12631, 12632, 12633, 12634, 12635,
     + 12636, 12637, 12638, 12639, 12640, 12641, 12642, 12643, 12644,
     +     0,     0, 13697, 13698, 13699, 13700, 13701, 13702, 13703,
     + 13704,     0,     0, 13705, 13706, 13707, 13708, 13709, 13710,
     + 13711, 13712, 13713, 13714, 13715, 13716, 13717, 13718, 13719,
     + 13720, 13721, 13722, 13723, 13724, 13725, 13726, 13727, 13728,
     + 13729, 13730, 13731, 13732,     0,     0, 13441, 13442, 13443,
     + 13444, 13445, 13446, 13447, 13448,     0,     0, 13449, 13450,
     + 13451, 13452, 13453, 13454, 13455, 13456, 13457, 13458, 13459,
     + 13460, 13461, 13462, 13463, 13464, 13465, 13466, 13467, 13468/
      data (IPO(L),L=271,360)/
     + 13469, 13470, 13471, 13472, 13473, 13474, 13475, 13476,     0,
     +     0, 13185, 13186, 13187, 13188, 13189, 13190, 13191, 13192,
     +     0,     0, 13193, 13194, 13195, 13196, 13197, 13198, 13199,
     + 13200, 13201, 13202, 13203, 13204, 13205, 13206, 13207, 13208,
     + 13209, 13210, 13211, 13212, 13213, 13214, 13215, 13216, 13217,
     + 13218, 13219, 13220,     0,     0, 12929, 12930, 12931, 12932,
     + 12933, 12934, 12935, 12936,     0,     0, 12937, 12938, 12939,
     + 12940, 12941, 12942, 12943, 12944, 12945, 12946, 12947, 12948,
     + 12949, 12950, 12951, 12952, 12953, 12954, 12955, 12956, 12957,
     + 12958, 12959, 12960, 12961, 12962, 12963, 12964,     0,     0/
      data (IPO(L),L=361,450)/
     + 12673, 12674, 12675, 12676, 12677, 12678, 12679, 12680,     0,
     +     0, 12681, 12682, 12683, 12684, 12685, 12686, 12687, 12688,
     + 12689, 12690, 12691, 12692, 12693, 12694, 12695, 12696, 12697,
```

```
+ 12698, 12699, 12700, 12701, 12702, 12703, 12704, 12705, 12706,
+ 12707, 12708,     0,     0,  9537,  9538,  9539,  9540,  9541,
+  9542,  9543,  9544,  9545,  9546,  9547,  9548,  9549,  9550,
+  9551,  9552,  9553,  9554,  9555,  9556,  9557,  9558,  9559,
+  9560,  9561,  9562,  9563,  9564,  9565,  9566,  9567,  9568,
+  9569,  9570,  9571,  9572,  9573,  9574,  9575,  9576,  9281,
+  9282,  9283,  9284,  9285,  9286,  9287,  9288,  9289,  9290/
 data (IPO(L),L=451,540)/
+  9291,  9292,  9293,  9294,  9295,  9296,  9297,  9298,  9299,
+  9300,  9301,  9302,  9303,  9304,  9305,  9306,  9307,  9308,
+  9309,  9310,  9311,  9312,  9313,  9314,  9315,  9316,  9317,
+  9318,  9319,  9320,  9025,  9026,  9027,  9028,  9029,  9030,
+  9031,  9032,  9033,  9034,  9035,  9036,  9037,  9038,  9039,
+  9040,  9041,  9042,  9043,  9044,  9045,  9046,  9047,  9048,
+  9049,  9050,  9051,  9052,  9053,  9054,  9055,  9056,  9057,
+  9058,  9059,  9060,  9061,  9062,  9063,  9064,  8769,  8770,
+  8771,  8772,  8773,  8774,  8775,  8776,  8777,  8778,  8779,
+  8780,  8781,  8782,  8783,  8784,  8785,  8786,  8787,  8788/
 data (IPO(L),L=541,630)/
+  8789,  8790,  8791,  8792,  8793,  8794,  8795,  8796,  8797,
+  8798,  8799,  8800,  8801,  8802,  8803,  8804,  8805,  8806,
+  8807,  8808,  8513,  8514,  8515,  8516,  8517,  8518,  8519,
+  8520,  8521,  8522,  8523,  8524,  8525,  8526,  8527,  8528,
+  8529,  8530,  8531,  8532,  8533,  8534,  8535,  8536,  8537,
+  8538,  8539,  8540,  8541,  8542,  8543,  8544,  8545,  8546,
+  8547,  8548,  8549,  8550,  8551,  8552,  9601,  9602,  9603,
+  9604,  9605,  9606,  9607,  9608,  9609,  9610,  9611,  9612,
+  9613,  9614,  9615,  9616,  9617,  9618,  9619,  9620,  9621,
+  9622,  9623,  9624,  9625,  9626,  9627,  9628,  9629,  9630/
 data (IPO(L),L=631,720)/
+  9631,  9632,  9633,  9634,  9635,  9636,  9637,  9638,  9639,
+  9640,  9345,  9346,  9347,  9348,  9349,  9350,  9351,  9352,
+  9353,  9354,  9355,  9356,  9357,  9358,  9359,  9360,  9361,
+  9362,  9363,  9364,  9365,  9366,  9367,  9368,  9369,  9370,
+  9371,  9372,  9373,  9374,  9375,  9376,  9377,  9378,  9379,
+  9380,  9381,  9382,  9383,  9384,  9089,  9090,  9091,  9092,
+  9093,  9094,  9095,  9096,  9097,  9098,  9099,  9100,  9101,
+  9102,  9103,  9104,  9105,  9106,  9107,  9108,  9109,  9110,
+  9111,  9112,  9113,  9114,  9115,  9116,  9117,  9118,  9119,
+  9120,  9121,  9122,  9123,  9124,  9125,  9126,  9127,  9128/
 data (IPO(L),L=721,810)/
+  8833,  8834,  8835,  8836,  8837,  8838,  8839,  8840,  8841,
+  8842,  8843,  8844,  8845,  8846,  8847,  8848,  8849,  8850,
+  8851,  8852,  8853,  8854,  8855,  8856,  8857,  8858,  8859,
+  8860,  8861,  8862,  8863,  8864,  8865,  8866,  8867,  8868,
+  8869,  8870,  8871,  8872,  8577,  8578,  8579,  8580,  8581,
+  8582,  8583,  8584,  8585,  8586,  8587,  8588,  8589,  8590,
+  8591,  8592,  8593,  8594,  8595,  8596,  8597,  8598,  8599,
+  8600,  8601,  8602,  8603,  8604,  8605,  8606,  8607,  8608,
+  8609,  8610,  8611,  8612,  8613,  8614,  8615,  8616,  5186,
+  5187,  5188,  5189,  5190,  5191,  5192,  5193,  5194,  5195/
 data (IPO(L),L=811,900)/
+  5196,  5197,  5198,  5199,  5200,  5201,  5202,  5203,  5204,
+  5205,  5206,  5207,  5208,  5209,  5210,  5211,  5212,  5213,
+  5214,  5215,  5216,  5217,  5218,  5219,  5220,  5221,  5222,
+  5223,  5224,  5225,  5226,  5227,  5228,  5229,  5230,  5231,
+  5232,  5233,  5234,  5235,  5236,  5237,  5238,  5239,  5240,
```

```
+ 5241, 5242, 5243, 5244, 5245, 4930, 4931, 4932, 4933,
+ 4934, 4935, 4936, 4937, 4938, 4939, 4940, 4941, 4942,
+ 4943, 4944, 4945, 4946, 4947, 4948, 4949, 4950, 4951,
+ 4952, 4953, 4954, 4955, 4956, 4957, 4958, 4959, 4960,
+ 4961, 4962, 4963, 4964, 4965, 4966, 4967, 4968, 4969/
 data (IPO(L),L=901,990)/
+ 4970, 4971, 4972, 4973, 4974, 4975, 4976, 4977, 4978,
+ 4979, 4980, 4981, 4982, 4983, 4984, 4985, 4986, 4987,
+ 4988, 4989, 4674, 4675, 4676, 4677, 4678, 4679, 4680,
+ 4681, 4682, 4683, 4684, 4685, 4686, 4687, 4688, 4689,
+ 4690, 4691, 4692, 4693, 4694, 4695, 4696, 4697, 4698,
+ 4699, 4700, 4701, 4702, 4703, 4704, 4705, 4706, 4707,
+ 4708, 4709, 4710, 4711, 4712, 4713, 4714, 4715, 4716,
+ 4717, 4718, 4719, 4720, 4721, 4722, 4723, 4724, 4725,
+ 4726, 4727, 4728, 4729, 4730, 4731, 4732, 4733, 4418,
+ 4419, 4420, 4421, 4422, 4423, 4424, 4425, 4426, 4427/
 data (IPO(L),L=991,1080)/
+ 4468, 4469, 4470, 4471, 4472, 4473, 4474, 4475, 4476,
+ 4477, 5250, 5251, 5252, 5253, 5254, 5255, 5256, 5257,
+ 5258, 5259, 5260, 5261, 5262, 5263, 5264, 5265, 5266,
+ 5267, 5268, 5269, 5270, 5271, 5272, 5273, 5274, 5275,
+ 5276, 5277, 5278, 5279, 5280, 5281, 5282, 5283, 5284,
+ 5285, 5286, 5287, 5288, 5289, 5290, 5291, 5292, 5293,
+ 5294, 5295, 5296, 5297, 5298, 5299, 5300, 5301, 5302,
+ 5303, 5304, 5305, 5306, 5307, 5308, 5309, 4994, 4995,
+ 4996, 4997, 4998, 4999, 5000, 5001, 5002, 5003, 5004,
+ 5005, 5006, 5007, 5008, 5009, 5010, 5011, 5012, 5013/
 data (IPO(L),L=1081,1170)/
+ 5014, 5015, 5016, 5017, 5018, 5019, 5020, 5021, 5022,
+ 5023, 5024, 5025, 5026, 5027, 5028, 5029, 5030, 5031,
+ 5032, 5033, 5034, 5035, 5036, 5037, 5038, 5039, 5040,
+ 5041, 5042, 5043, 5044, 5045, 5046, 5047, 5048, 5049,
+ 5050, 5051, 5052, 5053, 4738, 4739, 4740, 4741, 4742,
+ 4743, 4744, 4745, 4746, 4747, 4748, 4749, 4750, 4751,
+ 4752, 4753, 4754, 4755, 4756, 4757, 4758, 4759, 4760,
+ 4761, 4762, 4763, 4764, 4765, 4766, 4767, 4768, 4769,
+ 4770, 4771, 4772, 4773, 4774, 4775, 4776, 4777, 4778,
+ 4779, 4780, 4781, 4782, 4783, 4784, 4785, 4786, 4787/
 data (IPO(L),L=1171,1260)/
+ 4788, 4789, 4790, 4791, 4792, 4793, 4794, 4795, 4796,
+ 4797, 4482, 4483, 4484, 4485, 4486, 4487, 4488, 4489,
+ 4490, 4491, 4532, 4533, 4534, 4535, 4536, 4537, 4538,
+ 4539, 4540, 4541, 4428, 4429, 4430, 4431, 4432, 4433,
+ 4434, 4435, 4436, 4437, 4438, 4439, 4440, 4441, 4442,
+ 4443, 4444, 4445, 4446, 4447, 4448, 4449, 4450, 4451,
+ 4452, 4453, 4454, 4455, 4456, 4457, 4458, 4459, 4460,
+ 4461, 4462, 4463, 4464, 4465, 4466, 4467, 4492, 4493,
+ 4494, 4495, 4496, 4497, 4498, 4499, 4500, 4501, 4502,
+ 4503, 4504, 4505, 4506, 4507, 4508, 4509, 4510, 4511/
 data (IPO(L),L=1261,1300)/
+ 4512, 4513, 4514, 4515, 4516, 4517, 4518, 4519, 4520,
+ 4521, 4522, 4523, 4524, 4525, 4526, 4527, 4528, 4529,
+ 4530, 4531, 4417, 4673, 4929, 5185, 4478, 4737, 4990,
+ 5249,    0,    0, 4481, 4734, 4993, 5246, 4542, 4798,
+ 5054, 5310,    0,    0/
 DATA ALIAS /'3021','3022','3023','3024','3061','3062',
+'3041','3042','3043','3044','3051','3052','3053','3054',
```

```
     +'3011','3012','3015','3016','3017','3014','3018',
     +'3031','3032','3033','3034','3091',
     +'3071','3072','3073','3074','3075','3076','3077','3078',
     +'3081','3082','3083','3084','3085','3086','3087','3088'/
      DATA DITAG /'TUBES/X-COORD',
     +            'TUBES/V-COORD',
     +            'TUBES/V-OFFSET',
     +            'TUBES/ORIENTATION',
     +            'TUBES/ANALOG-V-COORD',
     +            'TUBES/ANALOG-V-OFFSET',
     +            'BREMS/X-COORD',
     +            'BREMS/V-COORD',
     +            'BREMS/V-OFFSET',
     +            'BREMS/ORIENTATION',
     +            'DRIFT/X-COORD',
     +            'DRIFT/V-COORD',
     +            'DRIFT/V-OFFSET',
     +            'DRIFT/ORIENTATION',
     +            'CALOR/X-COORD',
     +            'CALOR/V-COORD',
     +            'CALOR/ELM-V-OFFSET',
     +            'CALOR/HA1-V-OFFSET',
     +            'CALOR/HA2-V-OFFSET',
     +            'CALOR/ORIENTATION',
     +            'CALOR/MASK',
     +            'FIBER/X-COORD',
     +            'FIBER/V-COORD',
     +            'FIBER/V-OFFSET',
     +            'FIBER/ORIENTATION',
     +            'DIAMO/X-COORD',
     +            'TRIGG/X-COORD-PLAN',
     +            'TRIGG/Y-COORD-PLAN',
     +            'TRIGG/Z-COORD-PLAN',
     +            'TRIGG/X-WIDTH-PLAN',
     +            'TRIGG/Y-WIDTH-PLAN',
     +            'TRIGG/Z-WIDTH-PLAN',
     +            'TRIGG/Z-ANGLE-PLAN',
     +            'TRIGG/POINTER-TO-BAR',
     +            'TRIGG/X-COORD-BAR',
     +            'TRIGG/Y-COORD-BAR',
     +            'TRIGG/Z-COORD-BAR',
     +            'TRIGG/X-WIDTH-BAR',
     +            'TRIGG/Y-WIDTH-BAR',
     +            'TRIGG/Z-WIDTH-BAR',
     +            'TRIGG/Z-ANGLE-BAR',
     +            'TRIGG/POINTER-TO-PLAN'/
C
C---      INITIALISATION
C
      CALL CDPAW(NPAW,NHBOOK,IDIV,'USR-DIV',5000,50000,'ZPHU',IRC)
      PRINT '('' IRC FROM CDPAW '',I5)',IRC
      LUNCD=1
      LUNFZ=2
      CALL CDPREF(10,'CH',CHTOP,CHFILE,IRC)
      PRINT '('' IRC FROM CDPREF '',I5)',IRC
      LRECL = 0
      CALL CDOPEN(LUNCD,LUNFZ,CHTOP,CHFILE,LRECL,IDIV,' ',IRC)
```

```
      PRINT '('' IRC FROM CDOPEN '',I5)',IRC
C
C---  STORE VECTORS
C
      DO IDX=1,NDX
          DNAME = '//CDCH/GEOMETRY/'//DITAG(IDX)
          NDAT = 1300
          CALL CDVECT(' ',IPO,NDAT,JADDR,'PI',IRC)
          PRINT '('' IRC FROM CDVECT '',I5)',IRC
          KEYDBS(11) = 1
          KEYDBS(12) = 999999
          IDIV = 0
          CALL CDSTOR(DNAME(1:26),JADDR,LDUMI,IDIV,KEYDBS,' ',IRC)
          PRINT '('' IRC FROM CDSTOR '',I5)',IRC
      END DO
C
C--- TERMINATION
C
      CALL CDEND(' ','A',IRC)
      END
```

The following example shows how objects may be copied from one database to another. The directory structures in the two databases is different in this case.

Copying objects from one database to another

```
      PROGRAM COPYHDB
C     ---------------
C     FILLS THE DIRECTORY STRUCTURE
C     FOR THE GEOMETRY DATABASE OF CHORUS
C
      DIMENSION KEYDBS(100),KEY(5),JP(5)
      DATA KEY /3023,3062,3043,3053,3071/
      DATA JP  /   3,   6,  10,  14,   7/
      PARAMETER (NWPAW=400000,NHBOOK=0,NDX=43)
      COMMON/PAWC/NWP,IXPAWC,IHDIV,IXHIGZ,IXKU,FENC(5),LMAIN,HCV(NWPAW)
      DIMENSION IQ(2),Q(2),LQ(8000)
      EQUIVALENCE (LQ(1),LMAIN),(IQ(1),LQ(9)),(Q(1),IQ(1))
      CHARACTER*7 CHPAT1
      DATA CHPAT1 /'//CDC2/'/
      CHARACTER*4 CHTOP
      CHARACTER*80 CHFILE,CHPATH
      CHARACTER*80 DNAME
      CHARACTER*40 DITAG(NDX)
      CHARACTER*4  ALIAS(NDX)
      DATA ALIAS /'3021','3022','3023','3024','3061','3062','3063',
     +'3041','3042','3043','3044','3051','3052','3053','3054',
     +'3011','3012','3015','3016','3017','3014','3018',
     +'3031','3032','3033','3034','3091',
     +'3071','3072','3073','3074','3075','3076','3077','3078',
     +'3081','3082','3083','3084','3085','3086','3087','3088'/
      DATA DITAG /'TUBES/X-COORD',
     +            'TUBES/V-COORD',
     +            'TUBES/V-OFFSET',
     +            'TUBES/ORIENTATION',
     +            'TUBES/ANALOG-V-COORD',
```

```
     +            'TUBES/ANALOG-V-OFFSET',
     +            'TUBES/MAGNET',
     +            'BREMS/X-COORD',
     +            'BREMS/V-COORD',
     +            'BREMS/V-OFFSET',
     +            'BREMS/ORIENTATION',
     +            'DRIFT/X-COORD',
     +            'DRIFT/V-COORD',
     +            'DRIFT/V-OFFSET',
     +            'DRIFT/ORIENTATION',
     +            'CALOR/X-COORD',
     +            'CALOR/V-COORD',
     +            'CALOR/ELM-V-OFFSET',
     +            'CALOR/HA1-V-OFFSET',
     +            'CALOR/HA2-V-OFFSET',
     +            'CALOR/ORIENTATION',
     +            'CALOR/MASK',
     +            'FIBER/X-COORD',
     +            'FIBER/V-COORD',
     +            'FIBER/V-OFFSET',
     +            'FIBER/ORIENTATION',
     +            'DIAMO/X-COORD',
     +            'TRIGG/X-COORD-PLAN',
     +            'TRIGG/Y-COORD-PLAN',
     +            'TRIGG/Z-COORD-PLAN',
     +            'TRIGG/X-WIDTH-PLAN',
     +            'TRIGG/Y-WIDTH-PLAN',
     +            'TRIGG/Z-WIDTH-PLAN',
     +            'TRIGG/Z-ANGLE-PLAN',
     +            'TRIGG/POINTER-TO-BAR',
     +            'TRIGG/X-COORD-BAR',
     +            'TRIGG/Y-COORD-BAR',
     +            'TRIGG/Z-COORD-BAR',
     +            'TRIGG/X-WIDTH-BAR',
     +            'TRIGG/Y-WIDTH-BAR',
     +            'TRIGG/Z-WIDTH-BAR',
     +            'TRIGG/Z-ANGLE-BAR',
     +            'TRIGG/POINTER-TO-PLAN'/
C
C---      INITIALISATION, OPEN 2 DATABASE FILES
C
      CALL CDPAW(NWPAW,NHBOOK,IDIV,'USR-DIV',5000,50000,'ZPHU',IRC)
      PRINT '('' IRC FROM CDPAW '',I5)',IRC
      LUNCD=1
      LUNFZ=2
      CALL CDPREF(10,'CH',CHTOP,CHFILE,IRC)
      PRINT '('' IRC FROM CDPREF1 '',I5)',IRC
      LRECL = 0
      CALL CDOPEN(LUNCD,LUNFZ,CHTOP,CHFILE,LRECL,IDIV,' ',IRC)
      PRINT '('' IRC FROM CDOPEN1 '',I5)',IRC
*
      LUNCD=3
      LUNFZ=4
      CALL CDPREF(10,'C2',CHTOP,CHFILE,IRC)
      PRINT '('' IRC FROM CDPREF2 '',I5)',IRC
      LRECL = 0
      CALL CDOPEN(LUNCD,LUNFZ,CHTOP,CHFILE,LRECL,IDIV,' ',IRC)
```

```
      PRINT '('' IRC FROM CDOPEN2 '',I5)',IRC
C
C---  COPY OBJECTS
C
C     DO IDX=1,NDX
      DO J=1,5
          WRITE(CHPATH,'(A7,I4)') CHPAT1,KEY(J)
          DNAME = '//CDCH/GEOMETRY/'//DITAG(JP(J))
          NRUN = 1
          CALL CDUSE(CHPATH,JKEY,NRUN,'N',IRC)
          JADDR = LQ(JKEY-1)
          PRINT '(A30)',CHPATH
          PRINT '(I8)',JADDR
          PRINT '('' IRC FROM CDUSE '',I5)',IRC
          KEYDBS(11) = 1
          KEYDBS(12) = 999999
          CALL CDSTOR(DNAME,JADDR,LDUMI,IDIV,KEYDBS,' ',IRC)
          PRINT '('' IRC FROM CDSTOR '',I5)',IRC
      END DO
C
C--- TERMINATION
C
      CALL CDEND(' ','A',IRC)
      END
```

Appendix B: HEPDB compression algorithms

Three methods of packing are used by the HEPDB when storing data. These are the **differences** method, the **delta** method, and the **packing** method.

B.1 The delta packing method

In the delta packing method, the user has to supply a floating point variable (delta). All numbers below this (in absolute value) are treated as zero and ignored. The output bank has 3 extra words.

1 The number of data words in the original bank
2 delta
3 the number of words stored after the zero-suppression.

B.2 The packing method

In the packing method, all numbers are first converted to integers by multiplying the original number with `10**IPREC`, a user specified quantity. They are then truncated at the decimal point. The minimum offset to make all numbers positive is then found and added to each value. The optimum packing factor is then determined. This factor is chosen so that the minimum number of words are used after packing. The data is then packed, except for those words which cannot be stored with the chosen packing factor, which are stored in 32 bit words. Here again 3 extra words are used to keep useful information, as follows:

1 The number of data words `*10**4 + original data type*1000 + (IPREC+100)` The data type is indicated as follows:

`2` Integer
`3` Floating point

2 The offset to make all the numbers positive
3 A packed word with bits
`1-26` = number of words stored as 32-bit words,
bits `27-31` = number of bits used in packing,
bit `32` set to distinguish from the other methods.

B.3 The differences method

In addition to the above two methods, one can also store the difference of the current object from a master. A comparison is made with the five nearest neighbours which are not themselves updates. The differences are then made against that neighbour for which the minimum number of words are required. The user may select that the user keys must also match during the search for nearest neighbours.

Appendix C: Extraction and Merging of database records

C.1 Copying database records from one to another database

There will probably be a need to extract only a subset of the contents of a database to form a smaller private database. For maintaining the main database, the manager will have to be able to merge new or updated records from privately created database files into the main database, and there will also be the need to create 'snapshot' records of the status of the current valid records in the database. Routine `HDBEXTR` serves these purposes.

`HDBEXTR` extracts records from the subdirectory (or subdirectories) specified by the path `PATHI` , and also (if `PATHI` only leads to a directory, not a subdirectory) the `NSDIR`-element `CHARACTER` vector `CHDIR`, which contains the subdirectory names (optionally, records from all subdirectories are extracted). The extracted records are valid for the period of validity specified in the integer vector `KEYS`. If the `W` option is not requested and there is more than one valid current record, appropriate for the given type, in the range then a series of summary records (each of which compresses the information from the chain of records which apply to a part of the period of validity) is created. The extracted record is written into the corresponding subdirectory in the file with top-directory name `PATHO`. There is also additional information written which is needed for book-keeping when databases are merged (see also option `E` in `HDBSTOR` and `HDBSTOM`). If the user only wants a copy of the database records he can choose option `M`, which suppresses the storing of merge informations. By default a brief summary of all extracted records is printed, unless option `N` is specified. If the user wishes to write the extracted records also on an FZ file he should specify the option `X`, but note that the appropriated FZ calls should have previously been issued before calling `HDBEXTR`.

```
CALL HDBEXTR  (PATHI,PATHO,KEYS,CHSDIR,NSECD,CHOPT,IRC*)
```

`PATHI`	Full pathname of directory to be extracted.
`PATHO`	Name of top directory of the output database. This database should not yet contain any of the subdirectories that are to be copied from the main database, unless option `O` is requested.
`ISEL`	Vector specifying the required validity range.
`CHDIR`	Character array of subdirectories to be extracted
`NSDIR`	Number elements in array `CHDIR`
`CHOPT`	Character variable containing a string of single letter options

`' '`	directory specified by `PATHI` will be extracted
`'A'`	All subdirectories of the specified path will be extracted
`'C'`	extract only records with key status bit copy set
`'I'`	extend the validity range beyond one experiment number
`'M'`	no merge information will be stored
`'N'`	no summary will be printed
`'S'`	a selected subset of subdetector subdirectories (specified in `CHDIR`) will be extracted
`'O'`	subdetector-subdirectories in `PATHO` may exist
`'W'`	the whole number of stored records are copied to the auxiliary database. However the key word containing the pointer to the logging will be changed. The input values of `ISEL` are ignored.

- 'X' write extracted records also to an FZ file, where the FZ logical unit number is given by the logical unit number for PATHO+10

IRC Return status

- 0 Normal completion

C.2 Merging databases or database records

The problem of merging records into the main database in a coherent manner is not a trivial one. In particular, care must be taken that conflicting updates are not entered into the main database. The routine HDBMERG described below performs checks to ensure that at record to be merged into the database does not conflict with any updates already in the database that have been made since information upon which the new record is based was extracted from the main database.

Routine HDBMERG will merge a database or records from a database with a pre-existing 'target' database. Depending on the options selected, all subdetector subdirectories or only a selected few may have their records merged. The subdetector-subdirectories to be merged are either specified by a full pathname in CHAUX, or else a path to the relevant subdetector-directory in CHAUX and a list of the selected subdetector-subdirectory names in the NSDIR elements of the CHARACTER vector CHSDIR. The "target" database is identified by the top-directory name given in CHTMAI .

The database which contains the records to be merged can be created with the HDBEXTR routine or with the routine HDBNEW. If the database has been created with the HDBEXTR routine only the records which have been stored in this database after calling HDBEXTR will be merged in. In the other case all records will be merged in. If the option X is specified the merged records are also written to an FZ file, but note that the appropriated FZ calls should have previously been issued before calling HDBMERG. By default a brief summary of all merged records is printed. This can be suppressed with option N. For expert use only there is the facility to change the range of applicability of the records on merging (option R).

```
CALL HDBMERG (PATHI,PATHO,KEYS,CHSDIR,NSECD,CHOPT,IRC*)
```

PATHI Full pathname of directory to be MERGacted.

PATHO Name of top directory of the output database. This database should not yet contain any of the subdirectories that are to be copied from the main database, unless option O is requested.

KEYS Vector specifying the required validity range.

CHDIR Character array of subdirectories to be MERGacted

NSDIR Number elements in array CHDIR

CHOPT Character variable containing a string of single letter options

- ' ' directory specified by PATHI will be MERGacted
- 'A' All subdirectories of the specified path will be MERGacted
- 'C' MERGact only records with key status bit copy set
- 'I' extend the validity range beyond one experiment number
- 'M' no merge information will be stored
- 'N' no summary will be printed
- 'S' a selected subset of subdetector subdirectories (specified in CHSDIR) will be MERGacted

- 'O' subdetector-subdirectories in PATHO may exist
- 'R' the new period of validity given by the integer vector KEYS is used. This may only be used for database records created by the HDBEXTR routine and with option W not requested.
- X write extracted records also to an FZ file, where the FZ logical unit number is given by the logical unit number for PATHO+10.

IRC Return status

- 0 Normal completion

C.3 Transferring datastructures to and from FZ files

RZ are not in themselves transportable between computers, unless they are written in exchange format (which is the default for HEPDB RZ files). They may also be translated into FZ file format for transportation. This process is possible for a whole database, a directory sub-tree or a single database record. It is possible to store the validity information in the record headers or in a special start-of-run record.

The records are written to an FZ file using the routine HDBTOFZ, and may be read back into a database using the routine HDBFRFZ. It is also possible to store the contents of some other FZ file in a specified subdirectory using the routine HDBFRFZ. The routine HDBFZM can only used for FZ files which have been produced by the routines HDBEXTR or HDBMERG with the option X requested, because of the special FZ file organisation.

C.4 Converting database records into an FZ file

```
CALL HDBTOFZ (PATH,LUN,ISEL,CHOPT,IRC*)
```

PATH Pathname of the directory from where the data are to be retrieved.

LUN Logical unit number for the FZ file.

CHOPT Character variable containing a string of single character options

- ' ' Write retrieved record to a FZ file.
- 'C' Write pathname into header vector (not valid for R option).
- 'P' Write period of validity and NSTYPE in header vector.
- 'R' Write start-of-run record.
- 'RP' Write start-of-run record with header containing the period of validity.
- 'W' Writes the directory and all subdirectories to a FZ file (using RZTOFZ). Other options are ignored.

IRC Return status

- 0 Normal completion

Information from a HEPDB database may be copied to a ZEBRA FZ sequential file using the routine HDBTOFZ. This routine can output a single datastructure, the contents of a subdirectory or directory tree with optional selection on perid of validity. The required OPEN and FZFILE statements must have previously been issued for the FZ file before calling this routine (see, for example, PROGRAM HDBTOFZ in the HEPDB PAM file).

C.5 Reading database records from FZ files

CALL **HDBFRFZ** (PATH,LUN,NSKIP,KEYS,CHOPT,IRC*)

PATH Pathname of the directory into which the records are to be copied.

LUN Logical unit number for the FZ file.

NSKIP Skip NSKIP records before reading. If option R is chosen the start-of-run record is not included in NSKIP.

KEYS Vector of KEYS containing the period of validity.

CHOPT Character variable containing a string of single character options

- ' ' No start-of-run record and take period of validity from input. Store as base record.
- 'A' Create the directory specified by PATH before reading the FZ file (cannot create a new file if a whole database is to be loaded).
- 'B' Store as base record.
- 'C' Header contains the pathname, i.e. first header contains length of the pathname (max. 40 words) and the following words contain the pathname. However, the pathname cannot be written into the start-of-run header.
- 'H' Take period of validity from the first 6 words of each header. If option P is chosen take sub-run type from the seventh word of the header.
- 'I' extend the validity range beyond one experiment number
- 'P' Store as partial record.
- 'R' FZ file begins with start-of-run record.
- 'RH' As option R and take period of validity from start-of-run header as is described for option H.
- 'T' Take time stamp from input KEYS vector
- 'W' The whole directory is loaded via call to RZFRFZ.

IRC Return status

- 0 Normal completion

An FZ file created using the routine HDBTOFZ or the routines HDBMERG or HDBEXTR with the X option may be read using the routine HDBFRRZ. In the special case that the FZ file contains the contents of a complete database, using the option W in HDBTOFZ, a new database file should be created using the routine HDBNEW before calling this routine. In all other cases, the data is entered into the subdirectory specified by the path PATH, or else the subdirectory tree stored is added at the level specified by the path PATH. This routine may also be used to process FZ files that are not written by one of the HEPDB routines. For example, an FZ file containing a single datastructure may be read in and entered into the database in the directory specified by the variable PATH with the period of validity as specified by ISEL.

Clearly, if the FZ file contains a database or directory tree, the validity of the recovered records is the same as those originally stored. However, when the FZ file contains a single record extracted from a database previously, or a completely new ZEBRA structure, then the validity may be specified. If the validity has been stored with the record, then this may be retrieved and applied. If the record contains a start-of-run record, the option R must be used.

C.6 Merging FZ files written by HDBEXTR or HDBMERG

FZ files written by HDBEXTR or HDBMERG with the option X may be processed with the routine HDBFZM. This routine decodes this FZ file structure and merges the corresponding information into the database. The user must issue the appropriate OPEN statement and call to the routine FZFILE before calling HDBFZM.

```
CALL HDBFZM (PATH,LUN,CHOPT,IRC*)
```

PATH Top directory name of the database into which the records are to be copied.

LUN Logical unit number for the FZ file.

CHOPT Character variable containing a string of single character options

- ' ' Copy records, skip directories which are not present, and print a summary.
- 'A' Add directories which are not present.
- 'N' No summary will be printed.

IRC Integer return code

- 0 Normal completion

Appendix D: Updating HEPDB databases

Several methods exist for updating HEPDB databases. Normally, updates are not applied directly to the database itself but queued to a dedicated server. The client-server communication also has several variants but the most important is when the communication is via files. Only this method will be described here as it is the only one enabled in the standard CERNLIB distribution of the package.

When a user accesses a HEPDB database a journal file is opened in which database modifications are written. This journal file is created in the user's directory and is in Zebra FZ alpha exchange format. This file is then moved to a queue directory upon request, when the user starts to modify a different database or when the database is closed. The queue directory is defined by a configuration file which is described in detail below.

The journal file name contains information on which database the modifications are for and the user and node name from which the update originated.

The above scheme works well in both localised and distributed environments and is designed with file systems such as NFS and DFS in mind. In the case of nodes which do not offer network file systems, journal files are transferred using the CERN Program Library package CSPACK [6].

Appendix E: Access control

By default, all users may access any database in read mode. Updates are also possible, but are not performed directly, but placed in a queue where they will be handled by a server.

One may use standard file permissions to control read and write access to a database (using write access to the queue directory in case of updates.) Alternatively, the **:read** and **:write** tags, described on page 100, may be used.

Appendix F: Creating a new database

The following example shows how to create a new database file.

Creating a new database file

```
      PROGRAM CDEXA1
*     ==============
*
*     Create a new, empty database
*
      PARAMETER   (NWPAW=100000)
      COMMON/PAWC/PAW(NWPAW)
*
*     Initialise Zebra, HBOOK and HEPDB
*
      CALL CDPAW(NWPAW,NHBOOK,IDIV,'USR-DIV',5000,50000,'ZPHU',IRC)
*
*     Unit for database access
*
      LUNCD  = 1
*
*     Database parameters
*
      NPAIR  = 1
      NREC   = 20000
      NTOP   = 1
*
*     Accept default record length (1024 words)
*
      LRECL  = 0
      CALL CDNEW(LUNCD,'HEPDB','HEPDB.DBS',IDIV,NPAIR,NREC,NTOP,
     +           LRECL,' ',IRC)
*
*     Set the log level
*
      CALL CDLOGL(' ',3,'A',IRC)
*
*     Terminate
*
      CALL CDEND(' ','A',IRC)

      END
```

The same result can be achieved by running the CDMAKE program, e.g. using the following script.

Script to run CDMAKE program

```
#
# Make a new database
#
# export CDFILE='name of the database file'
  export CDFILE='test.dbs'

# export CDPAIR='number of validity range pairs'
  export CDPAIR=1
```

```
# export CDPRE='number of records to be preformatted'
  export CDPRE=100

# export CDTOP='numeric ID for database'
  export CDTOP=1

# export CDQUO='number of records for database quota'
  export CDQUO=65000

# export CDRECL='record length of database file (words)'
  export CDRECL=1024

#
# now run the job
#

  /cern/pro/bin/cdmake
```

Appendix G: Managing the database servers

Once the database file has been created, the server must be configured for this file. This is done using a NAMES file, as follows.

Names file entries for a database file (hepdb.names)

```
:nick.config
            :list.ge au
            :log./hepdb/cplear/logs
            :queue./hepdb/cplear/queue
            :todo./hepdb/cplear/todo
            :save./hepdb/cplear/save
            :bad./hepdb/cplear/bad
            :loglevel.3
            :wakeup.60
            :servers.cernvm vxcpon hepdb

:nick.ge
            :file./hepdb/cplear/database/geo.dbs
            :servers.vxcpon hepdb cernvm
            :desc.Geometry database for the CPLEAR experiment

:nick.au
            :file./hepdb/cplear/database/aux.dbs
            :servers.vxcpon cernvm
            :desc.Auxiliary database for the CPLEAR experiment
            :read.*
            :write.phr cpb

:nick.hepdb
            :userid.cdcplear
            :node.hepdb
            :localq./hepdb/l3/todo

:nick.vxcpon
            :userid.cdcplear
            :node.vxcpon
            :queue.disk$db:[cdcplear.todo]
            :protocol.tcpip
            :localq./hepdb/cplear/tovxcpon

:nick.cernvm
            :userid.cdcplear
            :node.cernvm
            :queue./hepdb/cplear/tocernvm
            :protocol.tcpip
            :localq./hepdb/cplear/tocernvm
```

The various tags in the preceding names file have the following meanings.

CONFIG Configuration details for the server, as follows.

LIST A list of two character database prefixes

`LOG`	The directory where the server logs are written
`QUEUE`	The directory where new updates are placed by HEPDB clients
`TODO`	The directory scanned by the HEPDB servers for updates to process. In the case of `MASTER` servers, the `todo` and `queue` directories are the same. In the case of `SLAVE` servers, these queues are different.
`BAD`	The directory where the server places `bad` updates. Bad updates are files for which the corresponding database cannot be found, or updates which cannot be successfully processed by the database server.
`SAVE`	The directory where the server saves updates after processing
`LOGL`	The log level for the server
`WAKEUP`	The wakeup interval in seconds for the server
`SERVERS`	This is the *or* of the list of servers for the individual databases. The database servers are responsible for moving updates to the local queues for the remote servers. A separate process, CDMOVE, is responsible for moving the processes between different systems.

`prefix` The two character database prefix, e.g. `aa`.

`FILE`	The full name of the database file. For VM/CMS systems, the syntax is `<user.address>filename.filetype`
`DESC`	A comment string identifying the database and/or its purpose
`SERVERS`	The list of remote servers for this database. Each node in this list **must** also be in the list for the :nick.config entry.
`READ`	A list of users who may read the database. An asterisk grants read access to all users. If this tag is not present, read access control is not performed.
`WRITE`	A list of users who may update the database. Users with write access automatically gain read access. If this tag is not present, write access control is not performed.

`server` The nickname of the servers, e.g. `aa1`.

`USERID`	Userid under which the server runs on the remote node
`NODE`	Node on which the server runs
`QUEUE`	Input queue on the remote node
`PROTOCOL`	Method by which updates are transmitted
`LOCALQ`	The local directory where updates are written pending transmission to the remote node. This may, in fact, be the same as `QUEUE`, e.g. when the directory is accessible via NFS or AFS.

G.1 Master and slave database servers

Objects entered into a `HEPDB` database are assigned a unique key within the directory into which they are inserted (the key serial number) and are stamped with the insertion date and time. It is important that these values are the same in all copies of the database. This is achieved by assigning these values centrally. The node on which the so-called `master` server runs may be different for each experiment but will typically be at the laboratory where the experiment is being conducted. At CERN, a dedicated system has been set up to host the master database servers. This is node `hepdb`.

Master and slave servers operate identically. The only difference lies in the names file (hepdb.names) that drives them. Updates are always queued by the HEPDB client software into the directory pointed to by the `:queue` tag in the names file, as described above. The servers scan the directory pointed to by the `:todo` tag for outstanding updates. In the case of the master server, the `:queue` and `:todo` directories are the same. In all other cases it is a separate process that performs the automatic distribution of updates between servers. In the case of distributed file systems such as `afs`, this operation is trivial. In other cases `TCP/IP, Bitnet, DECnet` or other transport mechanism is used.

The updates are stamped with the user and node name of origin. This allows the servers to avoid forwarding updates back to their node of origin.

Appendix H: Managing HEPDB servers at CERN

The following sections describe the setup of HEPDB at CERN. In general, the descriptions are also valid for other sites.

H.1 Creating a new server on CERNVM

On CERNVM a dedicated account is used per experiment. Thus for `CPLEAR` we have the account `CDCPLEAR`, for `CHORUS` we have `CDCHORUS`. These accounts are created using the standard `USERREG` procedure. Each account has 3 mini-disks plus a link to the `191` disk of the `HEPDB` machine. The latter is used to store the various EXECs that are required for the servers, to avoid cluttering up the `CERNLIB` disks.

The various disks are used as follows:

`191` Disk for the database files and log files

`192` Link to `HEPDB 191`

`193` Disk for journal files

`194` Disk for `bad` files, i.e. those that cannot be successfully processed.

In addition there is a special server named `CDHEPDB`. This is used to exchange journal files between node `hepdb` and `CERNVM`. It is also used as a gateway to remote `Bitnet` sites. Thus, in the case of `CPLEAR`, updates from the `VM/CMS` systems in Lyon, Saclay and Rutherford are first sent via `CDHEPDB` to the master server on node `hepdb`. Once a unique key serial number and the insertion date and time has been allocated, the new journal files are then resent to the slave servers on those nodes and CERNVM.

The database servers are autologged by the machine `FATONE`, which also controls the `FATMEN` servers.

H.2 Transfer of updates between CERNVM and HEPDB

Updates are transferred between CERNVM and HEPDB by a dedicated service machine running under the account CDHEPDB. This machine keeps a TCP/IP connection open between the two nodes. Upon startup, it builds a list of HEPDB servers on the HEPDB node and transfers any pending updates. These updates are then sent to the appropriate server on CERNVM, or to a distribution list. The following example shows NAMES file (hepdb.names) entry that will cause updates to be distributed to multiple VM/CMS systems.

Sending updates to multiple VM/CMS systems

```
:nick.CDCPLEAR
             :list.cdcplear at frcpn11 cdcplear at cernvm
```

When an update is received by this service machine, it is immediately transferred to HEPDB and a scan for updates pending for CERNVM made.

Code for CDHEPDB service machine

```
      PROGRAM CDHEPDB
*CMZ :          21/02/91  16.24.17  by  Jamie Shiers
*-- Author :    Jamie Shiers   21/02/91
*     Program to move updates between CERNVM and HEPDB
*
*     Stolen from FATMEN.
*
      PARAMETER     (NDIR=100)
      CHARACTER*255 CHDIRS(NDIR)
      PARAMETER     (NMAX=500)
      CHARACTER*64  FILES(NMAX)
      CHARACTER*8   HEPUSR,HEPNOD,REMUSR,REMNOD,REMDBS
      CHARACTER*64  REMOTE,TARGET
      CHARACTER*12  CHTIME
      CHARACTER*8   CHUSER,CHPASS
      CHARACTER*8   CHNODE,CHTYPE,CHSYS,CHRAND
      CHARACTER*6   CHENT
      CHARACTER*80  CHMAIL,LINE,CHDIR
      CHARACTER*38  VALID
      CHARACTER*255 ERRMSG
      CHARACTER*2   CDPREF
      CHARACTER*255 CDFILE
      COMMON/PAWC/PAW(50000)
      PARAMETER     (IPRINT=6)
      PARAMETER     (IDEBUG=0)
      PARAMETER     (LUNI=1)
      PARAMETER     (LUNO=2)
+CDE,QUEST.
+CDE,SLATE.
      DATA          NENTRY/0/
      DATA          VALID/'ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890._'/
*
*     Initialise ZEBRA
*
      CALL HLIMIT(50000)
*
*     Initialise XZ
*
      CALL XZINIT(IPRINT,IDEBUG,LUNI,LUNO)
*
      CALL CDHOST(CHNODE,IRC)
      LNODE = LENOCC(CHNODE)
*
*     Open connection to HEPDB...
*
+SELF,IF=TCPSOCK.
      IDUMMY = CINIT(IDUMMY)
+SELF,IF=-TCPSOCK.
      CALL VMREXX('F','USER',CHUSER,IC)
      CALL VMREXX('F','PWD' ,CHPASS,IC)
      CALL CUTOL(CHUSER)
      CALL CUTOL(CHPASS)
      CALL VMSTAK(CHPASS,'L',IC)
      CALL VMSTAK(CHUSER,'L',IC)
+SELF.

      CALL CZOPEN('zserv','HEPDB',IRC)
```

```
*
*     First entry: look on hepdb before sleeping
*
      NDIRS = 0
      GOTO 20

   10 CALL VMCMS('EXEC HDBSERV',IRC)
      IF(IRC.EQ.99) GOTO 20
      IF(IRC.NE.0) THEN
         PRINT *,'CDHEPDB. error ',IRC,' from HDBSERV. Stopping...'
         GOTO 90
      ENDIF

      NENTRY = NENTRY + 1
*
*     Get the user and node name for this file...
*
      CALL VMCMS('GLOBALV SELECT *EXEC STACK HDBADDR',IC)
      CALL VMRTRM(LINE,IEND)
      ISTART = ICFNBL(LINE,1,IEND)
      CALL CDWORD(HEPUSR,0,' ',LINE(ISTART:IEND),IC)
      LHEP   = LENOCC(HEPUSR)
      CALL CDWORD(HEPNOD,1,' ',LINE(ISTART:IEND),IC)
      LNOD   = LENOCC(HEPNOD)
*
*     Get file name (for database prefix and name of remote server)
*
      CALL VMCMS('GLOBALV SELECT *EXEC STACK CDFILE',IC)
      CALL VMRTRM(CDFILE,LFILE)
      CDPREF = CDFILE(1:2)
      LBLANK = INDEX(CDFILE(1:LFILE),' ')
      JBLANK = INDEXB(CDFILE(1:LFILE),' ')
      REMDBS = CDFILE(LBLANK+1:JBLANK-1)
      LDBS   = JBLANK - LBLANK - 1

      IF(IDEBUG.GE.1)
     +PRINT *,'CDHEPDB. Update received for ',REMDBS(1:LDBS),' prefix ',
     +         CDPREF
*
*     Number of pending files
*
      CALL VMCMS('GLOBALV SELECT *EXEC STACK HDBFILES',IC)
      CALL VMRTRM(LINE,IEND)
      NFILES = ICDECI(LINE,1,IEND)

      CALL DATIME(ID,IT)
      WRITE(CHTIME,'(I6.6,I4.4,I2.2)') ID,IT,IS(6)
      WRITE(CHENT ,'(I6.6)') NENTRY
      CALL CDRAND(CHRAND,IRC)
*
*    Now put this file...
*    This assumes the HEPDB naming convention: /hepdb/cdgroup,
*                                             e.g. /hepdb/cdchorus
      CHDIR  = '/hepdb/'//REMDBS(1:LDBS)//
     +         '/todo'
      LDIR   = LENOCC(CHDIR)
*
```

```
      REMOTE = ' '
      REMOTE = 'zz'//CHTIME//CHRAND//CHENT
     +         //'.'//HEPUSR(1:LHEP)//'_'//HEPNOD(1:LNOD)
      LREM   = LENOCC(REMOTE)
      TARGET = REMOTE(1:LREM)
*
*     Change remote directory
*
      CALL CUTOL(CHDIR(1:LDIR))
      IF(IDEBUG.GE.1) PRINT *,'CDHEPDB. Changing remote directory to ',
     +   CHDIR(1:LDIR)
      CALL XZCD(CHDIR(1:LDIR),IRC)

      IF(IDEBUG.GE.1) PRINT *,'CDHEPDB. Sending file as ',
     +   REMOTE(1:LREM)
      CALL XZPUTA(CDFILE(1:LFILE),REMOTE(1:LREM),' ',IC)
      IF(IC.NE.0) THEN
         WRITE(ERRMSG,9001) IC,HEPUSR,HEPNOD
 9001    FORMAT(' CDHEPDB. error ',I6,' sending update from ',
     +            A,' at ',A,' to HEPDB')
         LMSG = LENOCC(ERRMSG)
         GOTO 100
      ENDIF
*
*     Rename the remote update file
*
      LSTA = INDEXB(TARGET(1:LREM),'/') + 1
      TARGET(LSTA:LSTA+1) = CDPREF
      IF(IDEBUG.GE.1) PRINT *,'CDHEPDB. Renaming file to ',
     +   TARGET(1:LREM)
      CALL XZMV(REMOTE(1:LREM),TARGET(1:LREM),' ',IRC)
*
*     Delete this update...
*
      CALL VMCMS('ERASE '//CDFILE(1:LFILE),IC)
*
*     Try to clear out RDR
*
      IF(NFILES.GT.10) GOTO 10
*
*     Are there any files for us to get?
*
   20 CONTINUE
*
*     Get list of remote directories
*
      JCONT  = 0
      IF(NDIRS.EQ.0) THEN
         IF(IDEBUG.GE.1) PRINT *, 'HEPDB. Retrieving list of remote '
     +   //'directories...'
         CALL XZLS('/hepdb/cd*/tovm',CHDIRS,NDIR,NDIRS,JCONT,'D',IC)
         NDIRS = MIN(NDIR,NDIRS)
         IF(JCONT.NE.0) THEN
            IC = 0
            PRINT *,'CDHEPDB. too many directories - excess names ',
     +      'will be flushed'
*
```

```
   30       CONTINUE
            CALL CZGETA(CHMAIL,ISTAT)
            LCH = LENOCC(CHMAIL)
            IF(CHMAIL(1:1).EQ.'0') THEN
*
*        Nop
*
            ELSEIF(CHMAIL(1:1).EQ.'1') THEN
            ELSEIF(CHMAIL(1:1).EQ.'2') THEN
               GOTO 30
            ELSEIF(CHMAIL(1:1).EQ.'3') THEN
               IQUEST(1) = 1
               IRC = 1
            ELSEIF(CHMAIL(1:1).EQ.'E') THEN
               IQUEST(1) = -1
               IRC = -1
            ELSEIF(CHMAIL(1:1).EQ.'V') THEN
               GOTO 30
            ELSE
               IQUEST(1) = 1
               IRC = 1
            ENDIF
*
         ENDIF
      ENDIF

      IF(NDIRS.EQ.1.AND.INDEX(CHDIRS(1),'not found').NE.0) THEN
         ERRMSG = 'CDHEPDB. there are no remote directories!'
         LMSG   = LENOCC(ERRMSG)
         GOTO 100
      ENDIF

      DO 80 J=1,NDIRS

         LDIR = LENOCC(CHDIRS(J))
         IF(LDIR.EQ.0) GOTO 80
         CALL CLTOU(CHDIRS(J)(1:LDIR))
*
*     Get the name of the server for whom these updates are intended...
*
         JSTART = INDEX(CHDIRS(J)(1:LDIR),'/CD')
         IF(JSTART.EQ.0) THEN
            IF(IDEBUG.GE.-3)
     +      PRINT *,'CDHEPDB. unrecognised directory - skipped ',
     +      '(',CHDIRS(J)(1:LDIR),')'
            GOTO 80
         ELSE
            JSTART = JSTART + 1
         ENDIF

         JEND = INDEX(CHDIRS(J)(JSTART:LDIR),'/')

         IF(JEND.EQ.0) THEN
            PRINT *,'CDHEPDB. unrecognised file name - skipped ',
     +      '(',CHDIRS(J)(1:LDIR),')'
            GOTO 80
         ENDIF
```

```
      REMUSR = CHDIRS(J)(JSTART:JSTART+JEND-2)
      LREM = LENOCC(REMUSR)

      IF(LREM.EQ.0) THEN
         IF(IDEBUG.GE.-3)
     +   PRINT *,'CDHEPDB. unrecognised file name - skipped ',
     +   '(',CHDIRS(J)(1:LDIR),')'
         GOTO 80
      ENDIF

      IF(IDEBUG.GE.1)
     +PRINT *,'CDHEPDB. processing updates for ',REMUSR(1:LREM)
      CALL XZCD(CHDIRS(J)(1:LDIR),IRC)
      IF(IRC.NE.0) THEN
         IF(IDEBUG.GE.-3)
     +   PRINT *,'CDHEPDB. cannot set directory to ',
     +      CHDIRS(J)(1:LDIR)
         GOTO 80
      ENDIF

      ICONT  = 0
      NFILES = 0
      IF(IDEBUG.GE.1) PRINT *, 'HEPDB. Retrieving list '
     +//'of remote files in ', CHDIRS(J)(1:LDIR)
      CALL XZLS(' ',FILES,NMAX,NFILES,ICONT,' ',IC)
      NFILES = MIN(NFILES,NMAX)
      IF(IDEBUG.GE.2)
     +PRINT *,'CDHEPDB. ',NFILES,' files found in ',CHDIRS(J)(1:LDIR)
      IF(ICONT.NE.0) THEN
         IC = 0
         IF(IDEBUG.GE.0)
     +   PRINT *,'CDHEPDB. too many files - excess names will be '
     +   //'flushed'
*
   40    CONTINUE
         CALL CZGETA(CHMAIL,ISTAT)
         LCH = LENOCC(CHMAIL)
         IF(CHMAIL(1:1).EQ.'0') THEN
*
*     Nop
*
         ELSEIF(CHMAIL(1:1).EQ.'1') THEN
         ELSEIF(CHMAIL(1:1).EQ.'2') THEN
            GOTO 40
         ELSEIF(CHMAIL(1:1).EQ.'3') THEN
            IQUEST(1) = 1
            IRC = 1
         ELSEIF(CHMAIL(1:1).EQ.'E') THEN
            IQUEST(1) = -1
            IRC = -1
         ELSEIF(CHMAIL(1:1).EQ.'V') THEN
            GOTO 40
         ELSE
            IQUEST(1) = 1
            IRC = 1
         ENDIF
```

```
*
      ENDIF


      DO 70 I=1,NFILES
         LF = LENOCC(FILES(I))
         IF(LF.EQ.0) GOTO 70
         CALL CLTOU(FILES(I))
*
*     Fix for the case when there are no files...
*
         IF(NFILES.EQ.1) THEN

            IF(INDEX(FILES(I)(1:LF),'DOES NOT EXIST').NE.0.OR.
     +         INDEX(FILES (I)(1:LF),'NOT FOUND').NE.0) GOTO 10

            IF(INDEX(FILES(I)(1:LF),'ARG LIST TOO LONG').NE.0) THEN
               IF(IDEBUG.GE.-3) THEN
                  PRINT *,'CDHEPDB. Stopping due to the following '
     +                  //'error...'
                  PRINT *,FILES(I)(1:LF)
                  PRINT *,'(Intervention required on HEPDB)'
               ENDIF
               CALL VMCMS('EXEC TELL JAMIE '//FILES(I)(1:LF),IC)
               CALL VMCMS('EXEC TELL JAMIE Logging off...',IC)
               CALL VMCMS('EXEC TELL FATONE Logging off due to'//
     +         FILES(I)(1:LF),IC)
               CALL VMSTAK('LOGOFF','L',IC)
               STOP
            ENDIF

         ENDIF
*
*     Check that file name is valid
*
         DO 50 L=1,LF
            IF(INDEX(VALID,FILES(I)(L:L)).EQ.0) THEN
               IF(IDEBUG.GE.-3) THEN
                  PRINT *,'CDHEPDB. invalid character ',
     +                     FILES(I)(L:L),
     +                     ' at ',L,' in ',FILES(I)(1:LF)
                  PRINT *,'CDHEPDB. skipping update...'
               ENDIF
               GOTO 70
            ENDIF
   50    CONTINUE

         IF(INDEX(FILES(I)(1:LF),CHNODE(1:LNODE)).NE.0) THEN
            IF(IDEBUG.GE.1)
     +      PRINT *,'CDHEPDB. skipping update for ',CHNODE(1:LNODE),
     +      '(',FILES(I)(1:LF),')'
            GOTO 70
         ENDIF
         LSLASH = INDEXB(FILES(I)(1:LF),'/')
         IF(FILES(I)(LSLASH+1:LSLASH+2).EQ.'ZZ') THEN
            IF(IDEBUG.GE.1)
     +      PRINT *,'CDHEPDB. active file - skipped ', '(',FILES(I)
```

```
     +         (1:LF),')'
               GOTO 70
            ENDIF

            IF(IDEBUG.GE.2)
     +      PRINT *,'CDHEPDB. update found for ',REMUSR(1:LREM), '(',
     +      FILES(I)(1:LF),')'

            IF(IDEBUG.GE.1) PRINT *,'CDHEPDB. retrieving update ',
     +      FILES(I)(1:LF)

            CDPREF = FILES(I)(1:LF)
            CALL CDRAND(CHRAND,IRC)
            CDFILE = CDPREF // CHRAND(3:) // '.HEPDB.B'
            LFILE  = 16

            CALL XZGETA(CDFILE(1:LFILE),FILES(I)(1:LF),'S',IC)
            IF(IC.NE.0) THEN
               WRITE(ERRMSG,9002) IC,REMUSR(1:LREM)
 9002   FORMAT(' CDHEPDB. error ',I6,' retrieving update for ',A)
               LMSG = LENOCC(ERRMSG)
               GOTO 100
            ENDIF
*
*      Protection against zero length files
*
            IF(IQUEST(11).EQ.0) GOTO 60

            LDOT = INDEX(CDFILE(1:LFILE),'.')
            CDFILE(LDOT:LDOT) = ' '
            LDOT = INDEX(CDFILE(1:LFILE),'.')
            CDFILE(LDOT:LDOT) = ' '

            CALL VMCMS('EXEC SENDFILE '//CDFILE(1:LFILE)//' TO '
     +                //REMUSR(1:LREM),IC)
            IF(IC.NE.0) THEN
               WRITE(ERRMSG,9003) IC,REMUSR(1:LREM)
 9003   FORMAT(' CDHEPDB. error ',I6,' sending update to ',A)
               LMSG = LENOCC(ERRMSG)
               GOTO 100
            ENDIF
*
*     Now delete local file
*
            CALL VMCMS('ERASE '//CDFILE(1:LFILE),IC)
*
*     and the remote one
*
   60       continue
            CALL XZRM(FILES(I)(1:LF),IC)
            IF(IC.NE.0) THEN
               WRITE(ERRMSG,9004) IC,FILES(I)(1:LF)
 9004   FORMAT(' CDHEPDB. error ',I6,' deleting file ',A)
               LMSG = LENOCC(ERRMSG)
               GOTO 100
            ENDIF
```

```
   70    CONTINUE

   80 CONTINUE
*
*     Wait for some action...
*
      GOTO 10

   90 CALL CZCLOS(ISTAT)
      STOP
  100 CONTINUE
*
*     Error exit
*
      IF(IDEBUG.GE.-3) PRINT *,ERRMSG(1:LMSG)
      CALL VMCMS('EXEC TELL JAMIE '//ERRMSG(1:LMSG),IC)
      CALL VMCMS('EXEC TELL JAMIE Logging off...',IC)
      CALL VMCMS('EXEC TELL FATONE Logging off due to'//ERRMSG(1:LMSG),
     +           IC)
      CALL VMSTAK('LOGOFF','L',IC)
      GOTO 90
      END
```

H.3 Transfer of updates between HEPDB and other nodes

Updates are transferred between HEPDB and non-VM nodes using a generalisation of the above procedure, known as **CDMOVE**. Rather than maintain a permanent network connection with each of the remote nodes, the method is as follows:

- Upon startup, the program obtains a list of experiments for which updates are to be processed. This list is defined by the environmental variable **CDGROUPS**.

Defining a list of groups

```
CDGROUPS="CDCHORUS,CDCPLEAR,CDNOMAD,CDCMS,CDNA48";export CDGROUPS
```

- Each group is then processed in turn. The directory containing the NAMES file (hepdb.names) for a given group is obtained from the environmental variable **CDgroup**, e.g. **CDCPLEAR**.
- The list of remote nodes is identified by the **:servers** tag of the **:nick.config** entry. Thus, for CPLEAR, the list of remote nodes is vxcrna uxcp05 axcp01 (see below). **N.B. Each database may have a different list of remote servers. The :servers tags on the individual database entries are processed by CDSERV, not CDMOVE. CDMOVE only looks at the :servers tag on the :nick.config entry.**
- The remote nodes are processed for each database. Any files in the local queue for the remote node will be sent. The local queue is defined by the tag **:localq**. If this queue is missing, the directory tonode, e.g. tovxcrna, is used.

- The updates are transferred to the directory defined by the **:queue** tag. If this tag is missing, the subdirectory **TODO** is used.
- One may define a different list of remote nodes for each database. However, this is the concern of **CDSERV**, the database server, and not **CDMOVE**. **CDSERV** makes a copy of all update files in a local queue directory for each remote server defined by the **:servers** tag for the current database. **CDMOVE** looks at all of the local queues - the database to which an update file corresponds is determined by the two character prefix of the update file.
- If the tag **:receive.yes** is specified, any pending updates will be retrieved from the remote system.
- If the tag **:poll.yes** is specified, the remote node will be contacted and any pending updates will be retrieved from the remote system, regardless of whether there are any updates pending for the remote node in question.
- A connection is only made to a remote system if there are updates pending for that system. This makes sense in an environment where database updates are typically produced centrally.
- The files are received from the directory identified by the **:remoteq** tag. If this tag is not specified, the subdirectory **queue** is used.
- In all cases, the updates are received into the **todo** subdirectory of the local server. It is assumed that **CDMOVE** will be run on the same machine as the master server.

Example script to run CDMOVE

```
#!/bin/ksh
#
# Define the list of groups for whom updates are to be processed
#
export CDGROUPS="CDCPLEAR,CDCHORUS"
#
# For each group, define the pathname where the hepdb.names file resides
#
export CDCPLEAR=/hepdb/cdcplear
export CDCHORUS=/hepdb/cdchorus
#
# Now start the server
#
/cern/pro/bin/cdmove
```

NAMES file for CDCPLEAR (hepdb.names)

```
:nick.config
   :list.au ge ca aa
   :log./hepdb/cdcplear/log
   :queue./hepdb/cdcplear/todo
   :todo./hepdb/cdcplear/todo
   :save./hepdb/cdcplear/save
```

```
   :bad./hepdb/cdcplear/bad
   :loglevel.3
   :wakeup.60
   :servers.vxcrna uxcp05 axcp01

:nick.au
   :file./hepdb/cdcplear/aux.dbs
   :desc.auxil database
   :servers.cernvm vxcrna uxcp05

:nick.ca
   :file./hepdb/cdcplear/cal.dbs
   :desc.calibration database
   :servers.cernvm vxcrna uxcp05

:nick.ge
   :file./hepdb/cdcplear/geo.dbs
   :desc.geometry database
   :servers.cernvm vxcrna uxcp05

:nick.aa
   :file./hepdb/cdcplear/aa.dbs
   :desc.test database
   :servers.cernvm

:nick.cernvm
   :userid.cdcplear
   :node.cernvm
   :localq./hepdb/cdcplear/tovm

:nick.vxcrna
   :userid.cdcplear
   :node.axcrnb
   :localq./hepdb/cdcplear/tovxcrna
   :queue.disk$mf:[cdcplear.todo]

:nick.uxcp05
   :userid.cdcplear
   :node.uxcp05
   :localq./hepdb/cdcplear/touxcp05
   :queue.disk$mf:[cdcplear.todo]
   :poll.yes

:nick.axcp01
   :userid.cdcplear
   :node.axcp01
   :localq./hepdb/cdcplear/toaxcp01
   :queue./hepdb/cdcplear/todo
```

NAMES file for CDCHORUS (hepdb.names)

```
:nick.config
   :list.c2 ch
   :log./hepdb/cdchorus/log
```

```
   :queue./hepdb/cdchorus/todo
   :todo./hepdb/cdchorus/todo
   :save./hepdb/cdchorus/save
   :bad./hepdb/cdchorus/bad
   :loglevel.3
   :wakeup.60
   :servers.vxcrna xantia

:nick.c2
   :file./hepdb/cdchorus/charm2.dbs
   :desc.Charm2 database converted to HEPDB format
   :servers.cernvm vxcrna xantia

:nick.ch
   :file./hepdb/cdchorus/chorus.dbs
   :desc.CHORUS (geometry) database
   :servers.cernvm vxcrna xantia

:nick.cernvm
   :userid.cdhepdb
   :node.cernvm
   :localq./hepdb/cdchorus/tovm

:nick.vxcrna
   :userid.cdhepdb
   :node.vxcrna
   :localq./hepdb/cdchorus/tovxcrna

:nick.xantia
   :userid.cdhepdb
   :node.xantia.caspur.it
   :localq./hepdb/cdchorus/toxantia
   :receive:yes
   :queue./cern/hepdb/cdchorus/todo
```

The actual file transfer is performed by `cspack` [6] routines. Thus, the remote node must be set up correctly, i.e. **ZSERV** must be installed. The user name and password for the remote connection are defined in the **.netrc** file (or ftplogin. for VMS systems) in the home directory of the account under which the **CDMOVE** job is run.

H.4 Example of configuration on two VAX systems

Let us take the example of two VAX systems: **VXLNFB** and **AXPALS**. In this example, the *master* server will run on the node **VXLNFB**. To achieve this, we must configue the file **hepdb.names** so that the **:queue** and **:todo** tags point to the same directory, as shown below.

Example names file for master server

```
:nick.config
:list.ct rt gt
:log.DISK$MF:[KLOEDB.LOG]
:queue.DISK$MF:[KLOEDB.TODO]
:todo.DISK$MF:[KLOEDB.TODO]
```

```
:save.DISK$MF:[KLOEDB.SAVE]
:bad.DISK$MF:[KLOEDB.BAD]
:loglevel.3
:wakeup.60
:servers.axpals

:nick.ct
:file.DISK$MF:[KLOEDB.DATABASE]CALRT.DBS
:desc.Calibration Database for Test Beam 1994
:servers.axpals

:nick.rt
:file.DISK$MF:[KLOEDB.DATABASE]RUNRT.DBS
:desc.RUN CONDITION Database for Test Beam 1994
:servers.axpals

:nick.gt
:file.DISK$MF:[KLOEDB.DATABASE]GEORT.DBS
:desc.Geometry Database for Test Beam 1994
:servers.axpals

:nick.axpals
:userid.kloedb
:node.axpals
:poll.yes
```

The names file for the slave server is similar, as shown below.

Example names file for slave server

```
:nick.config
:list.ct rt gt
:log.DISK$MF:[KLOEDB.LOG]
:queue.DISK$MF:[KLOEDB.TOVXLNFB]
:todo.DISK$MF:[KLOEDB.TODO]
:save.DISK$MF:[KLOEDB.SAVE]
:bad.DISK$MF:[KLOEDB.BAD]
:loglevel.3
:wakeup.60
:servers.

:nick.ct
:file.DISK$MF:[KLOEDB.DATABASE]CALRT.DBS
:desc.Calibration Database for Test Beam 1994
:servers.vxlnfb

:nick.rt
:file.DISK$MF:[KLOEDB.DATABASE]RUNRT.DBS
:desc.RUN CONDITION Database for Test Beam 1994
:servers.vxlnfb

:nick.gt
:file.DISK$MF:[KLOEDB.DATABASE]GEORT.DBS
:desc.Geometry Database for Test Beam 1994
:servers.vxlnfb
```

In the above examples, it is assumed that **CDMOVE** will run on the same node as the *master* server, e.g. **VXLNFB**.

H.4.1 Making an update on a node with a MASTER server

- If an update is made on the node where the master server is running, it is written into the directory **DISK$MF:[KLOEDB.TODO]**.
- The master server will process this update and make a copy in the directory **DISK$MF:[KLOEDB.TOAXPALS]**.
- A seperate process, **CDMOVE** takes the files in **DISK$MF:[KLOEDB.TOAXPALS]**, connects to the node **AXPALS** and transfers the files to the directory **DISK$MF:[KLOEDB.TODO]**.
- It will then be processed by the **CDSERV** process on this node and the local copy of the database updated.

H.4.2 Making an update on a node with a SLAVE server

- If an update is made on the node where the slave server is running, it will be written into the directory **DISK$MF:[KLOEDB.TOVXLNFB]**.
- The **CDMOVE** process running on VXLNFB will connect at periodic intervals and transfer any pending files, as we have specified the tag **:poll.yes**, the **CDMOVE** Alternatively, one may specify **:receive.yes**, in which case updates will only be transferred from **AXPALS** if the **CDMOVE** server has made a connection to transfer updates from **VXLNFB** to **AXPALS**. If neither tag is specified, updates will never be transferred from **AXPALS** to **VXLNFB**.

H.4.3 Running CDMOVE on the slave node

Normally, one would run the **CDMOVE** process on the same node as the master server. As stated above, this requires that **ZSERV** has been correctly configured on all slave nodes.

If this is not possible for some reason, e.g. if UCX is running on the slave node, then **CDMOVE** can be run on the slave node. This, in turn, requires that **ZSERV** is correctly configured on the master node.

In this case the names file on the slave node must be modified as shown below. The changes are indented for clarity, although this is purely cosmetic.

Example names file for slave server with changes for CDMOVE

```
:nick.config
:list.ct rt gt
:log.DISK$MF:[KLOEDB.LOG]
:queue.DISK$MF:[KLOEDB.TOVXLNFB]
:todo.DISK$MF:[KLOEDB.TODO]
:save.DISK$MF:[KLOEDB.SAVE]
:bad.DISK$MF:[KLOEDB.BAD]
:loglevel.3
:wakeup.60
   :servers.VXLNFB
```

```
:nick.ct
:file.DISK$MF:[KLOEDB.DATABASE]CALRT.DBS
:desc.Calibration Database for Test Beam 1994
:servers.vxlnfb

:nick.rt
:file.DISK$MF:[KLOEDB.DATABASE]RUNRT.DBS
:desc.RUN CONDITION Database for Test Beam 1994
:servers.vxlnfb

:nick.gt
:file.DISK$MF:[KLOEDB.DATABASE]GEORT.DBS
:desc.Geometry Database for Test Beam 1994
:servers.vxlnfb

   :nick.VXLNFB
   :userid.kloedb
   :node.VXLNFB
   :poll.yes
```

H.5 Creating a new server on VXCRNA

Although it is possible to access remote database files from VAX/VMS systems using NFS, there are cases, such as on an online VAXcluster, when it is desirable to have a database server on the VAX itself.

As above, an account is first created using USERREG. This account is then configured using the following command file, included in the standard distribution:

CDNEW.COM

```
$!DECK ID>, CDNEW.COM
$ !
$ ! Setup the directory and file structure for a new
$ ! server
$ !
$   procedure = f$parse(f$environment("PROCEDURE"),,,"NAME")
$   say       = "write sys$output"
$   if p1 .eqs. ""
$      then
$         write sys$output "''procedure': usage ''procedure' group"
$         exit
$   endif
$ !
$ ! Does the directory already exist?
$ !
$   home = f$search("DISK$MF:[000000]''p1'.dir")
$   if home .eqs. ""
$      then
$         say "''procedure': home directory for ''p1' does not exist."
$         say "''procedure': please create an account using USERREG"
$         exit
$   endif
$ !
```

```
$ ! Create subdirectories
$ !
$   create/directory DISK$MF:['p1'.BAD]
$   create/directory DISK$MF:['p1'.LOG]
$   create/directory DISK$MF:['p1'.QUEUE]
$   set file/protection=w:rwe DISK$MF:['p1']QUEUE.DIR
$   create/directory DISK$MF:['p1'.TODO]
$   create/directory DISK$MF:['p1'.SAVE]
$   directory DISK$MF:['p1'] /security
$ !
$ ! Create names file
$ !
$   open/write out DISK$MF:['p1']HEPDB.NAMES
$   write out ":nick.config"
$   write out ":list.aa"
$   write out ":bad.DISK$MF:[''p1'.BAD]"
$   write out ":log.DISK$MF:[''p1'.LOG]"
$   write out ":queue.DISK$MF:[''p1'.QUEUE]"
$   write out ":todo.DISK$MF:[''p1'.TODO]"
$   write out ":save.DISK$MF:[''p1'.SAVE]"
$   write out ":wakeup.120"
$   write out ":loglevel.3"
$   close out
$   type DISK$MF:['p1']HEPDB.NAMES
```

Should the disk in question have disk quotas enabled, one should ensure that the `queue` directory is owned by an identifier and has an ACL as in the following example:

Queue directory on VAX/VMS systems

```
(IDENTIFIER=CDCHORUS,ACCESS=READ+WRITE+EXECUTE+DELETE+CONTROL)
(IDENTIFIER=ID$_CHORUS,ACCESS=READ+WRITE+EXECUTE)
(IDENTIFIER=ID$_CHORUS,OPTIONS=DEFAULT,ACCESS=READ+WRITE+EXECUTE+DELETE+CONTROL)
(IDENTIFIER=CDF_EXPERIMENT,OPTIONS=DEFAULT,ACCESS=READ+WRITE+EXECUTE)
```

The identifier must be granted to all users who should be permitted to update the database with the `RESOURCE` attribute.

As for `CERNVM`, an extra account exists which is used to exchange updates between `VXCRNA` and `hepdb`.

The files created on `hepdb` must have the correct ownership (in this case UID 102 and GID 3) which must be mapped to the UIC under which the command file is executed on the VAX.

This is performed as follows:

Mapping a Unix UID/GID pair to a VMS username

```
$ !    MULTINET CONFIGURE /NFS
$ !    NFS-CONFIG>add cdhepdb 102 3
$ !    NFS-CONFIG>ctrl-z
```

This is done using the following command file:

Moving updates between VXCRNA and hepdb

```
$!DECK ID>, CDSEND.COM
$ !
$ ! Command file to move updates between 'slave' and 'master'
$ !
$ ! Invoked by CDSERV.COM from the account CDHEPDB on VXCRNA
$ !
$ ! Assumes correct UID & GID mapping for directories on 'master'
$ !    MULTINET CONFIGURE /NFS
$ !    NFS-CONFIG>add cdhepdb 102 3
$ !    NFS-CONFIG>ctrl-z
$ !
$ set noon
$ !
$ ! List of servers
$ !
$   cdservers = "CDCPLEAR,CDCHORUS,CDNOMAD"
$ !
$ ! Master & slave definitions
$ !
$   slave     = "VXCRNA"
$   master    = "HEPDB"
$ !
$ main_loop:
$   nserver   = 0
$ !
$ ! Loop over all servers
$ !
$ loop_servers:
$   server    = f$element(nserver,",",cdservers)
$   nserver   = nserver + 1
$   if server .eqs. "," then goto sleep
$ !
$ ! Look for files waiting to be sent to 'master'
$ !
$ to_hepdb:
$    journal_file = f$search("DISK$MF:[''server'.TO''master']*.*")
$ !
$    if journal_file .eqs. "" then goto from_hepdb
$ !
$ ! Skip 'active' files
$ !
$    if f$extract(0,2,journal_file) .eqs. "ZZ" then goto to_hepdb
$ !
$ ! Build remote file name
$ !
$   istart = f$locate("]",journal_file) + 1
$   remote_file = "''master':[''server'.TODO]ZZ" + -
      f$extract(istart+2,f$length(journal_file),journal_file)
$ !
$ ! Copy the file over
$ !
$   copy 'journal_file' 'remote_file' /log /noconfirm
$ !
$ ! Rename remote file and delete local file if it was ok
$ !
$   if $severity .eq. 1
$      then
```

```
$         remote_update = "''master':[''server'.TODO]" + -
             f$extract(istart,f$length(journal_file),journal_file)
$         rename 'remote_file' 'remote_update' /nolog /noconfirm
$         if $severity .eq. 1 then delete /nolog /noconfirm 'journal_file'
$   endif
$ !
$   goto to_hepdb
$ !
$ ! Look for files to be pulled over from 'master'
$ !
$ from_hepdb:
$    journal_file = f$search("HEPDB:[''server'.TO''slave']*.*")
$ !
$    if journal_file .eqs. "" then goto loop_servers
$ !
$ ! Skip 'active' files
$ !
$    if f$extract(0,2,journal_file) .eqs. "ZZ" then goto from_hepdb
$ !
$ ! Build local file name
$ !
$   istart = f$locate("]",journal_file) + 1
$   local_file = "DISK$MF:[''server'.TODO]ZZ" + -
       f$extract(istart+2,f$length(journal_file),journal_file)
$ !
$ ! Copy the file over
$ !
$   copy 'journal_file' 'local_file' /log /noconfirm
$ !
$ ! Rename local file and delete remote file if it was ok
$ !
$   if $severity .eq. 1
$      then
$         local_update = "DISK$MF:[''server'.TODO]" + -
             f$extract(istart,f$length(journal_file),journal_file)
$         rename 'local_file' 'local_update' /log /noconfirm
$         if $severity .eq. 1 then delete /log /noconfirm 'journal_file'
$   endif
$ !
$   goto from_hepdb
$ !
$ sleep:
$   wait 00:30:00
$   goto main_loop
```

The servers are controlled by the following job, which runs in the `SYS$FATMEN` queue:

Command file to control HEPDB servers

```
$!DECK ID>, CDMAST.COM
$SET NOON
$ !
$ !   Master HEPDB command file
$ !
$     save_mess = f$environment("MESSAGE")
$     set message/nofacility/noseverity/noid/notext
```

```
$     write sys$output "CDMAST starting at ''f$time()'"
$ !
$ !   define list of servers
$ !
$     servers  = "CDHEPDB,CDCHORUS,CDCPLEAR" ! Separate by commas
$     wakeup :== 00:30:00                    ! Every 30 minutes
$ !
$ !   define symbols - this is VXCRNA specific
$ !
$     n = 0
$ loop:
$     server    = f$element(n,",",servers)
$     if server .eqs. "," then goto again
$     'server' == "DISK$MF:[''server']"
$     n         = n + 1
$     goto loop
$ again:
$ !
$ !   Run the command files that expect a complete list as argument
$ !
$     write sys$output ">>> CDPURGE..."
$     @CERN_ROOT:[EXE]CDPURGE 'servers'  ! Purge old journal files
$ !
$     write sys$output ">>> CDCHECK..."
$     @CERN_ROOT:[EXE]CDCHECK 'servers'  ! Check that servers are started
$ !
$     write sys$output ">>> Time is ''f$time()'. Waiting ''wakeup'..."
$     wait 'wakeup'
$     write sys$output ">>> Wakeup at ''f$time()'."
$     goto again
$     set message 'save_mess'
$     exit
```

The job CDPURGE purges old journal and log files and is as follows:

Job to purge old journal and log files

```
$!DECK ID>, CDPURGE.COM
$SET NOON
$ !
$ ! Purge journalled HEPDB updates that are over a day old
$ !
$ if p1 .eqs. "" then exit
$ hepdb =  p1
$ count  = 0
$ save_mess = f$environment("MESSAGE")
$ set message/nofacility/noseverity/noid/notext
$loop:
$ server = f$element(count,",",hepdb)
$ if server .eqs. "," then goto end
$ count  = count + 1
$ write sys$output "Processing ''server'..."
$ ON WARNING THEN GOTO UNDEFINED
$ cddir = &server
$ purge 'cddir' ! Purge old log files
$ cdfil = f$extract(0,f$length(cddir)-1,cddir) + ".SAVE]*.*;*"
```

```
$ ON WARNING THEN CONTINUE
$ delete/before=-0-23:59 'cdfil'
$ goto loop
$ undefined:
$ write sys$output "Warning: symbol ''server' is not defined"
$ goto loop
$ end:
$ set message 'save_mess'
$ exit
```

The job to check and restart the servers is as follows:

CDCHECK command file

```
$!DECK ID>, CDCHECK.COM
$SET NOON
$ !
$ ! Check that HEPDB servers are started
$ !
$ if p1 .eqs. "" then exit
$ servers = p1
$ count   = 0
$ save_mess = f$environment("MESSAGE")
$ set message/nofacility/noseverity/noid/notext
$ !
$ ! Check that the queue is started
$ !
$ if f$getqui("DISPLAY_QUEUE","QUEUE_STOPPED","SYS$FATMEN") .eqs. "FALSE" then -
     start/queue sys$fatmen
$loop:
$ server = f$element(count,",",servers)
$ if server .eqs. "," then goto end
$ count  = count + 1
$ write sys$output "Processing ''server'..."
$ show user/nooutput 'server'
$ if $severity .ne. 1
$    then
$ !
$ !  Check that server has not been stopped
$ !
$    ON WARNING THEN GOTO UNDEFINED
$    cddir = &server
$    ON WARNING THEN CONTINUE
$    cddir = f$extract(0,f$length(cddir)-1,cddir) + ".TODO]SIGNAL.STOP"
$    if f$search(cddir) .nes. ""
$       then write sys$output "Signal.Stop file found - will not restart"
$       goto loop
$    endif
$    write sys$output "Restarting server ..."
$    cdserv = &server + "CDSERV.COM"
$    submit/queue=sys$fatmen/user='server' /id 'cdserv'
$    endif
$ goto loop
$ undefined:$ write sys$output "Warning: symbol ''server' is not defined"
$ goto loop
$ end:
$ exit
```

H.6 Accessing remote database files over NFS

One may avoid running a local database server on a given node by accessing the database files over the network. This is the recommended procedure for Unix systems at CERN. To enable this, one should first mount the `/hepdb` file system as shown below.

Mounting the /hepdb file system on a machine running Unix

```
mount hepdb:/hepdb /hepdb
```

Should your experiment require access to **HEPDB** from one of the CORE/SHIFT/CSF systems, please contact your CORE representative and ask them to perform the above action on the nodes in question.

On a VAX/VMS system that has the NFS client software installed, as is the case on VXCERN, the following commands are issued at system startup time.

Mounting the /hepdb file system on a machine running VMS

```
$ !
$ ! Mount the file system if not already done
$ !
$ if f$trnlm("HEPDB").eqs."" then NFSMOUNT/soft HEPDB::"/hepdb" HEPDB
```

The HEPDB software automatically uses C I/O to access remote database files on VMS systems. This is because VAX Fortran does not recognise the file structure of the remote Unix database file but is in any case completely transparent to the user.

It is currently recommended that the update directory reside on the local VMS system. This is because Multinet NFS requires that VMS UICs are mapped to Unix UID and GID pairs on the remote node, even if the remote directory is `world` writable (or writable by `others` in Unix parlence). On VXCERN only a single UIC is mapped to a valid UID/GID pair on node hepdb. A job runs under this UIC to move the update files between the local and remote file systems.

H.7 VMS systems running UCX

N.B. The use of UCX is not recommended. The following section remains for historical reasons only.

At the time of writing, DEC's UCX product still does not provide an NFS client. In this case one can mount a VMS directory on node `hepdb`. This is done today for `CPLEAR`.

As is the case for Multinet NFS, one must map a Unix UID/GID pair to a VMS username. In addition, a `binding` must be made been a VMS directory and a Unix style file name.

This can be done as follows:

Binding a VMS directory to a Unix name

```
$ UCX
UCX> BIND UXCP05$DKA300: /vxcplear
UCX> show bind

Logical filesystem                    Pathname

UXCP05$DKA300:                        /vxcplear
UCX>
```

Mapping a UID/GID pair to a VMS username

```
$ UCX
UCX> ADD PROXY CDCPLEAR /UID=102 /GID=1 /HOST=hepdb.cern.ch
```

Note that UCX treats hostnames as case sensitive.

Finally, one must start the UCX NFS server. This involves

- Modifying (correcting) the UCX startup command file `SYS$MANAGER:UCX$NFS_STARTUP.COM)`
- Invoking the command file at system startup.

Modifying the UCX NFS startup command file

```
$ ! ...
$ !
$ ! Set the following UID and GIDs
$ !
$ DEFINE/SYSTEM/EXE/NOLOG UCX$NFS00000000_GID 1
$ DEFINE/SYSTEM/EXE/NOLOG UCX$NFS00000000_UID 0
$ !
$ ! ...
$ !
$ ! Comment out the following line
$ ! RUN SYS$SYSTEM:UCX$SERVER_NFS.EXE
$ !
$ ! The following section contains NFS process quota that is required by
$ ! manual startup.  Please uncomment the following lines and comment out
$ ! the "RUN" command above, if you choose to manually start NFS.
$ !
$ RUN SYS$SYSTEM:UCX$SERVER_NFS.EXE/DETACH -
        /OUTPUT=NLA0: -
        /ERROR='P1' -
        /AST_LIMIT=512 -
        /BUFFER_LIMIT=200000 -
        /EXTENT=20000 -
        /FILE_LIMIT=1024 -
        /IO_BUFFERED=400 -
        /IO_DIRECT=200 -
        /QUEUE_LIMIT=64 -
```

```
        /ENQUEUE_LIMIT=3000 -
        /MAXIMUM_WORKING_SET=20000 -
        /PAGE_FILE=20000 -
        /PRIORITY=8 -
        /PRIVILEGES=(BYPASS,SYSPRV) -
        /UIC=[1,4] -
        /NORESOURCE
$ !
$EXIT:
$ EXIT
```

The file system is now ready for mounting on `hepdb`.

Extract from /etc/filesystems for /vxcplear

```
/vxcplear:
        dev             = /vxcplear/cdcplear
        vfs             = nfs
        nodename        = uxcp05
        mount           = true
        options         = bg,hard,intr
```

H.8 Setting up a new server on `hepdb`

The `hepdb` system is a dedicated IBM RS6000 that only runs `HEPDB` servers and associated jobs. The database files are maintained in the `/hepdb` file system. This is nfs exported and should be mounted on other Unix systems, such as `CSF`, as follows:

Mounting the /hepdb file system

```
mount hepdb:/hepdb /hepdb
```

Before creating a new server, the account must be registered for service **AFS** using **USERREG**. The account should be the letters **cd** followed by the name of the experiment, e.g. **cdatlas, cdnomad, cdna49**.

Once the account has been centrally registered for **AFS**, one should create an account on the **HEPDB** machine, using the UID and GID allocated by **USERREG** and visible through **XWHO**.

Finally, the following script is run to create the appropriate directories and dummy configuration files.

New servers can be setup using the following script, which creates the necessary directory structure and configuration files.

Creating the files and directories for a new server

```
#
# Setup the directory and file structure for a new
# server
#
iam=`whoami`
#
# Are we root?
#
if [ "$iam" != "root" ]
then
   echo $0: This script must be run from root
   exit
fi
#
# Did we get any arguments?
#
if [ $# != 1 ]
then
   echo $0: usage $0 group
   exit
fi
#
# Does this directory exist?
#
if [ -d /hepdb/$1 ]
then
   echo $0: Directory /hepdb/$1 already exists
   exit
fi
#
# No, so make it
#
mkdir /hepdb/$1
#
# and the subdirectories...
#
mkdir /hepdb/$1/bad
mkdir /hepdb/$1/log
mkdir /hepdb/$1/queue
chmod o+w /hepdb/$1/queue
mkdir /hepdb/$1/todo
mkdir /hepdb/$1/save
ls -F /hepdb/$1
#
# now create the names file
#
echo :nick.config > /hepdb/$1/hepdb.names
echo :list.aa     >> /hepdb/$1/hepdb.names
echo :log./hepdb/$1/log >> /hepdb/$1/hepdb.names
echo :queue./hepdb/$1/queue >> /hepdb/$1/hepdb.names
echo :todo./hepdb/$1/todo >> /hepdb/$1/hepdb.names
echo :save./hepdb/$1/save >> /hepdb/$1/hepdb.names
echo :bad./hepdb/$1/bad >> /hepdb/$1/hepdb.names
echo :loglevel.3 >> /hepdb/$1/hepdb.names
echo :wakeup.60 >> /hepdb/$1/hepdb.names
echo :nick.aa >> /hepdb/$1/hepdb.names
echo :file./hepdb/$1/aa.dbs >> /hepdb/$1/hepdb.names
```

```
echo :desc.Description of the database >> /hepdb/$1/hepdb.names
echo :servers. >> /hepdb/$1/hepdb.names
cat /hepdb/$1/hepdb.names
#
# Link the server script
#
ln -s /cern/new/bin/cdserv.sh /hepdb/$1/cdserv
#
# and the server module
#
ln -s /cern/new/bin/cdserv /hepdb/$1/cdsrv
```

The servers are started at boot time by adding the file /etc/inittab as follows:

Extract from /etc/inittab

```
rcnfs:2:wait:/etc/rc.nfs > /dev/console 2>&1 # Start NFS Daemons
hepdb:2:wait:/etc/rc.hepdb > /dev/console 2>&1 # Start HEPDB
cons:0123456789:respawn:/etc/getty /dev/console
```

This invokes the following script:

rc.hepdb

```
#!/bin/sh
#
#               Start HEPDB servers
#
#
if [ -x /cern/pro/bin/cdstart ]
then
        echo Start HEPDB servers ...
        su - hepdb /cern/pro/bin/cdstart 2>&1
fi
```

One may execute `cdstart` at any time, as it will only restart servers that are not already running.

cdstart script

```
#!/bin/ksh
start=" "
stop=" "
run=" "
nolog=" "
noscr=" "
b="."
#
#   Ensure that variables are defined...
#

for i in /hepdb/cd*
   do

echo
```

```
typeset -u cdgrp
cdpath=$i
cdgrp=`basename $i`
echo Setting $cdgrp to $cdpath ...
eval $cdgrp=$cdpath;export $cdgrp
#
# and start the servers
#
if [ -x $i/cdserv ]
   then
#
# does a log file exist?
#
   if [ -f /hepdb/$cdgrp.log ]
      then
      echo '>>> log file exists - looking for existing process'
      log=$log$b$cdgrp
      pid=`cat /hepdb/$cdgrp.log | awk 'printf "%s
n",$13'`
      if (test $pid)
         then
         echo Looking for server process for $cdgrp
         if(ps -ae  | grep -s $pid )
            then
            echo CDSRV running PID = $pid
            run=$run$b$cdgrp
            else
            echo No existing server found for $cdgrp - starting server
            if [ -f $i/todo/signal.stop ]
               then echo but signal.stop file found!
               else echo Starting server for $cdgrp
               nohup $i/cdserv $cdgrp > $i/cdserv.log &
               start=$start$b$cdgrp
            fi
         fi

         else
         echo No existing server found for $cdgrp - starting server
         if [ -f $i/todo/signal.stop ]
            then echo but signal.stop file found!
            stop=$stop$b$cdgrp
            else echo Starting server for $cdgrp
            nohup $i/cdserv $cdgrp > $i/cdserv.log &
            start=$start$b$cdgrp
         fi
      fi
      else
      echo No server log found in $i
      if [ -f $i/todo/signal.stop ]
         then echo but signal.stop file found!
         stop=$stop$b$cdgrp
         else echo Starting server for $cdgrp
         nohup $i/cdserv $cdgrp > $i/cdserv.log &
         start=$start$b$cdgrp
      fi
   fi
   else
```

```
   echo No cdserv script found in $i - cannot start server
   scr=$scr$b$cdgrp
fi

done

echo
echo Log files found for $log | tr '.' ' '
echo Started servers for $start | tr '.' ' '
echo Servers already running for $run | tr '.' ' '
echo Servers stopped $stop | tr '.' ' '
echo No scripts found for $scr | tr '.' ' '
```

The servers can be checked by running the following script:

Looking for running servers

```
echo 'HEPDB server                                        Elapsed     CPU time   %CPU'
echo '==============================================================================='
ps -aef -F "args,etime,time,pcpu" | grep "/cdsrv" | sort +2 -r
```

Output from the above script

```
HEPDB server                                        Elapsed     CPU time   %CPU
===============================================================================
/hepdb/cdnomad/cdsrv                              7-02:19:29    00:04:44   0.0
/hepdb/cdchorus/cdsrv                             7-02:19:29    00:04:43   0.0
/hepdb/cdcplear/cdsrv                             7-02:19:29    00:04:41   0.0
```

Appendix I: Examples of of the flow of journal files

I.1 Updating a database residing on node `hepdb` from a Unix system at CERN

In this simple example, the journal file is written directly to the directory pointed to by the `:queue` tag in the `hepdb.names` file. As the server is the database master, the `:queue` and `:todo` tags point to the same directory. Whilst the journal file is being written, the reserved prefix `zz` is used. As soon as it is complete, the file is renamed to have the correct database prefix so that the server, which can of course handle several databases for the same experiment, can identify which database file is to be updated.

After the update has been processed, the master server sends the new journal file to all slave servers.

I.2 Updating a database residing on node `hepdb` from CERNVM

In this case the update is first sent to the service machine `CDHEPDB`. This machine receives the file and transfers it to the `todo` directory of the appropriate server on `hepdb`. Once the update has been processed, the new journal file will be written to a `tovm` directory and transferred back. This file will then be sent to the local slave server on CERNVM and any remote servers on Bitnet nodes.

I.3 Updating a database residing on node `hepdb` from a VMS system at CERN

This is similar to the above, except that the journal file is written to a special `tohepdb` directory on the local VMS system. A batch job periodically scans this directory and copies any files found over NFS to `hepdb`. Once again, once the update has been processed the new journal file is copied back and placed in the `todo` directory of the local VMS slave server.

I.4 Updating a database residing on node `hepdb` from a remote VM system

This is the same as for the CERNVM case, except that the names file on the remote system points to `CDHEPDB at CERNVM`, rather than simply `CDHEPDB`.

Appendix J: Hardware configuration of node HEPDB

The central HEPDB server at CERN is an RS6000 model 320.

It has an internal disk, used only for the system, and two external disks, used for the /hepdb and /backdb filesystems.

HEPDB disk configuration

```
name   status    location    description

hdisk0 Available 00-01-00-00 320 MB SCSI Disk Drive
hdisk1 Available 00-01-00-30 Other SCSI Disk Drive
hdisk2 Available 00-01-00-40 Other SCSI Disk Drive
```

Appendix K: Return codes

Error Code	Meaning	Routine Name
-1	Invalid top directory name	CDINIT
-2	The file is already open with correct LUNRZ and TOPNM	CDINIT
-3	The file is already open with wrong LUNRZ or TOPNM	CDINIT
-5	Invalid process name in Online context	CDINIT
-6	Error in IC_BOOK for booking the CACHE	CDINIT
-7	Error in CC_SETUP for reserving the CLUSCOM	CDINIT
-8	Error in opening journal file in server mode	CDFOPN
-9	Unable to open FZ communication channel	CDINIT
-10	Host unable to open RZ file	CDINIT
1	Illegal character option	CDUSEDB/CDUSEM
2	Illegal path name	CDGETDB/CDUSE/CDUSEM
3	Data base structure in memory clobbered	CDUSE/CDUSEDB/CDUSEM
4	Illegal key option	CDUSE/CDUSEDB/CDUSEM
5	Error in CDCHLD in P3 communication	CDUSP3
12	Illegal pathname	CDNODE
13	Not enough structural link to support a new Node	CDNODE
15	Cannot define IO descriptor for Key bank	CDNODE
21	Too many keys with option M	CDKMUL
22	Illegal key option	CDKMUL
24	No Key bank created satisfying key options for option S	CDBKKS
25	Illegal Path Name	CDBKKS
31	Illegal path name or path name in node bank is wrong	CDCHCK/CDKXIN/CDPRIN
32	No keys/data in this directory	CDCHCK/CDGETDB CDPRIN
33	No valid data for the given range of insertion time or for the given set of keys and program version number	CDKXIN
34	RZIN fails to read the data	CDRZIN
35	Wrong reference to data objects in update mode	CDKXIN
36	Data bank address zero on return from CDKXIN	CDCHCK
37	Insufficient space in USER store array	CDCHCK
38	Read error in getting the RZ date and time	CDPRDT
39	Illegal data type in the key descriptor	CDPRKY
43	Too many key elements	CDMDIR
44	Cannot find the top directory name (wrong initialization)	CDMDIR
45	Illegal Path name	CDMDIR
47	The Directory already exists	CDMKDI
48	Error in directory search sequence	CDMKDI

49	FZOUT fails to write on the sequential file	CDSDIR
51	Illegal character option	CDFRDB
52	No access to the Key banks	CDFRDB
54	Pathname not matched to that found in bank NODB	CDFRDB
57	Illegal pathname	CDFRDB
58	Database structure in memory clobbered	CDFRDB
59	Some of the expected key banks not found	CDFRDB
61	Too many keys	CDENTB/CDREPL
62	Illegal character option	CDREPL/CDSTOM
63	Data base structure in memory clobbered	CDREPL/CDSTOR
64	Error in MZCOPY while copying Data bank	CDREPL/CDSTOR
65	Illegal number of data objects	CDSTOM
66	Illegal logical unit number	CDATOI/CDRHLP
67	File too long; no space in buffer	CDATOI
68	Input directory is partitioned	CDPART
69	Input directory is not partitioned	CDPURP
70	Error in deleting a partition through RZDELT	CDPURP
71	Illegal path name	CDDONT/CDENFZ/ CDENTB/CDFZUP/ CDKOUT/CDPART/ CDPURP/CDRTFZ
72	Read error on the FZ file (journal file)	CDENFZ/CDFZUP
73	RZOUT fails to write on disk	CDDONT/CDENFZ/ CDENTB/CDKOUT/ CDPART/CDPURP
74	Error in RZRENK in updating key values for partitioned data set	CDENFZ/CDENTB/ CDKOUT/CDPART/ CDPURP
76	Cannot form the IO descriptor for the FZ header	CDDONT/CDENTB/ CDFZUP/CDFZWR/ CDKOUT/CDPART
77	FZOUT fails to write on the sequential journal file	CDDONT/CDENFZ/ CDENTB/CDFZWR/ CDKOUT/CDPART/ CDPURP
78	Illegal number of keys on data base/journal file	CDFZUP/CDKOUT
79	Top directory name illegal in the FZ file	CDFZUP
81	Precision is not correctly given	CDUCMP
82	Illegal Data Type	CDUCMZ
83	Data update but uncompreseed	CDUNCP
84	The update structure has different number of data words	CDUNCP
85	No data in the structure	CDUNCP
86	The update structure has different data type	CDUNCP
91	Illegal Character Option	CDOPTS
92	Nonstandard IO descriptor	CDFRUS
93	Illegal time	CDPKTM/CDUPTM
94	Nonmatching NPAR's in different UPCD banks	CDVALID
95	Description not found in the dictionary	CDLDIC
96	RZCDIR fails to set to the current directory	CDLDUP
97	No matching UPCD bank found	CDLDUP/CDVALID
98	Invalid path name in Node bank	CDSTAT

99	No space in memory for creating the bank	CDBANK/CDRZIN
111	Illegal path name	CDPURG/CDPURK
112	No key or data for the path name	CDPURG/CDPURK
113	Illegal character option	CDPURK
114	Valid data object(s) in the Node/Key structure	CDPURK
115	Cannot form the IO descriptor for the FZ header	CDSPUR
116	FZOUT fails to write on the sequential file	CDSPUR
131	Illegal pathname (in key bank for CDLAST)	CDLAST/CDLKEY/ CDLMOD
132	Illegal number of keys in the directory	CDLAST/CDLKEY CDLMOD
135	Illgeal Top directory name	CDFZOP/CDILDU
136	Illegal logical unit number	CDILDF/CDILDU/ CDJOUR
140	Illegal top directory name	CDUDIC
141	Error in creating the DICTIONARY/HELP directory	CDUDIC
142	Error in RZ in writing the dictionary object	CDCDIC/CDUDIC
143	Error in RZ in purging the dictionary directory	CDCDIC/CDUDIC
144	Dictionary directory cannot be loaded	CDCDIC
145	Pathname already exists in the dictionary	CDCDIC
146	Illegal path name	CDDINF/CDEALI CDEHLP/CDENAM/ CDGNAM/CDRHLP/ CDRNAM
147	Dictionary directory not found in memory	CDEALI/CDGNAM/ CDRNAM
148	FZOUT fails to write on the sequential file	CDEALI/CDSNAM
149	Error in RZ for writing to the R.A. file	CDEALI/CDSNAM
150	Illegal number of data words	CDENAM
151	No description of data elements for the given path name exists in the data base	CDGNAM/CDRNAM
152	Illegal flag (IFLAG)	CDSNAM
153	FZIN error for reading the data structure	CDSNAM
154	Illegal alias name for a directory	CDRALI
155	No HELP directory inside the data base	CDRHLP
156	No help information for this path stored yet	CDRHLP
171	Illegal Path name	CDDDIR
172	Cannot find the top directory for the path name	CDDDIR
173	Error in RZ for reading the dictionary object	CDDDIR
174	Error in FZOUT for saving the journal file	CDDDIR
175	Error in RZ in writing the dictionary object	CDDDIR
176	Error in RZ in purging the dictionary directory	CDDDIR
177	Error in RZ in deleting the tree	CDDDIR
178	Error in RZ in deleting Name/Help information	CDDDIR
191	Illegal path name	CDRENK
192	Specified key elements do not match with any of the existing set of keys	CDRENK
194	Cannot form the IO descriptor for the FZ header	CDRENK
195	FZOUT fails to write on the sequential journal file	CDRENK
196	Error in RZRENK in updating key values partitioned data set	CDRENK

211	Illegal number of paths	CDKEEP
212	Illegal path name	CDFPAT/CDKEEP
213	Conflicting top directory names	CDKEEP
221	Error in CC_WRITELOCK for locking CLUSCOM (VAX);	CDWLOK
222	Error in CC_RELEASE for releasing CLUSCOM (VAX)	CDCWSV
223	Error in IC_SIGNAL for signalling the VAX Server	CDCWSV
225	Error in sending spool file to the server (IBM or APOLLO)	CDSTSV

Appendix L: Format for FZ output

HEPDB can create a journal file and can also update a data base from the corresponding journal file. The journal file format is defined as an FZ record consisting of a header and the data part. The format is general enough and can also be used for the communication betwen the server and a process which wants to update the data base.

The data part of the FZ record is relevant only for data to be entered. It is exactly the same data structure as input to `DBENTR`. For efficiency reason, HEPDB for its own journal file stores the data structure as input to the `RZOUT` call. This difference can be easily recognised from the value of `KEY(1)`, which is zero for outside source and nonzero for HEPDB's own journal file.

The header part has very similar structure for the eight actions foreseen so far, e.g., entering data, creating new directories, deleting data objects, deleting a directory tree, renaming the keys, entering names of data elements or help information for a directory, entering alias name to a directory, deleting a few partitions in a partitioned directory. However, they differ in details and the eight different types of FZ headers are listed below.

Header for entering data :

Word Count	Mnemonic	Type	Content
1	IACT	I	Action code (=1)
2	NWKEY	I	Number of key elements
3	NWDOP	I	Number of words used to store CHOPT
4	NDOP	I	Number of words used to to store the path name
5	IPREC	I	Precision chosen for packing (see DBENTR)
6	KEY(1)	I	Key element 1
..	...	..	....
NWKEY+5	KEY(NWKEY)	..	Key element NWKEY
NWKEY+6	CHOPT	H	Character option
..	..	H	
NWKEY+6 +NWDOP	PATHN	H	Path name
..	..	H	

Header for creating directories :

Word Count	Mnemonic	Type	Content
1	IACT	I	Action code (=2)
2	NWKEY	I	Number of key elements
3	NWDOP	I	Number of words used to store CHOPT
4	NDOP	I	Number of words used to to store the path name
5	MXKP	I	Maximum number of objects inside one partition (see DBMDIP)
6	INSTM	I	Insertion time packed up to minutes (see DBPKTM)

7	NRECD	I	Unused at this moment
8	CHOPT	H	Character option (e.g., 'P' for a
..	...	..	partitioned directory)
NDOP+8	CHFOR	H	Description of key element type. This
..	..	..	information is stored in NCFO = (NWKEY
..	..	..	+3)/4 words
NDOP+8	CHTAG	H	Tags for each key element. This info.
+NCFO	..	..	is stored in NTAG = 2*NWKEY words.
NDOP+NCFO	PATHN	H	Path name
+NTAG+8			
..	..	H	

Header for deleting objects :

Word Count	Mnemonic	Type	Content
1	IACT	I	Action code (=3)
2	NWKEY	I	Number of key elements
3	NWDOP	I	Number of words used to store CHOPT
4	NDOP	I	Number of words used to to store the
			path name
5	NPARS	I	Number of pairs of validity range (set
			for CDPURK) or -1 for CDPURG
6	INSTM	I	Deletion time packed up to minutes
			(see DBPKTM)
7	ISEL(1)	I	The objects to be selected using the
..	...		validity criteria in CDPURK
NPARS+6	ISEL(n)	I	
NPARS+7	KEY(1)	I	Key element 1 for CDURK
	...	..	....
NENDK	KEY(n)	..	Key element NWKEY for CDPURK
7	KYDAT	I	To be used for CDPURG
8	KYTIM	I	To be used for CDPURG
..	...	..	
NENDK			NWKEYth word following KYDAT for CDPURG
NENDK+1	CHOPT	H	Character option
..	...	..	
NENDK+1	PATHN	H	Path name
+NWDOP			
..	..	H	

Header for deleting directories :

Word Count	Mnemonic	Type	Content
1	IACT	I	Action code (=4)
2	---	I	Unused (set to 0)
3	NWDOP	I	Number of words used to store CHOPT
4	NDOP	I	Number of words used to to store the
			path name
5	---	I	Unused (set to 0)
6	INSTM	I	Deletion time packed up to minutes
			(see DBPKTM)

7	CHOPT	H	Character option
NWDOP+7	PATHN	H	Path name
..	..	H	

Header for renaming keys :

Word Count	Mnemonic	Type	Content
1	IACT	I	Action code (=5)
2	NWKEY	I	Number of key elements
3	NWDOP	I	Number of words for CHOPT (= 0)
4	NDOP	I	Number of words used to to store the path name
5	Unused	I	Set to zero
6	KYO(1)	I	Old key element 1
..	...	..	....
NWKEY+5	KYO(NWKEY)	..	Old key element NWKEY
NWKEY+6	KYN(1)	I	New key element 1
..	..	..	....
2*NWKEY+5	KYO(NWKEY)	..	New key element NWKEY
2*NWKEY+6	PATHN	H	Path name
..	..	H	

Header for entering/deleting names or help information :

Word Count	Mnemonic	Type	Content
1	IACT	I	Action code (=6)
2	NWKEY	I	Number of key elements
3	NWDOP	I	Number of words used to store CHOPT
4	NDOP	I	Number of words used to to store the path name (DICTIONARY or HELP)
5	IFLAG	I	Flag (1 for help information; 2 for names of the data elements)
6	KEY(1)	I	Key element 1 (= Identifier of path)
..	...	..	....
NWKEY+5	KEY(NWKEY)	..	Key element NWKEY
NWKEY+6	CHOPT	H	Character option
..	...	..	....
NWKEY+ NWDOP+6	PATHN	H	Path name (DICTIONARY or HELP)
..	..	H	

Header for entering the alias name :

Word Count	Mnemonic	Type	Content
1	IACT	I	Action code (=7)
2	---	I	Unused (set to 0)
3	NWDOP	I	Number of words used to store CHOPT(=0)
4	NDOP	I	Number of words used to to store the

			path name of the dictitionary
5	IFLAG	I	Flag (0 means temporary; 1 permanent)
6	NWDP	I	Number of words used to store the path name
7	PATHD	H	Path name of the dictionary
..	..	H	
NDOP+7	ALIAS	H	Alias name
..	..	H	
NDOP+9	PATHN	H	Path name of the directory
..	..	H	

Header for deleting a few partitions in a partitioned directory :

Word Count	Mnemonic	Type	Content
1	IACT	I	Action code (=8)
2	---	I	Unused (set to 0)
3	NWDOP	I	Number of words used to store CHOPT
4	NDOP	I	Number of words used to to store the path name
5	INSTM	I	Deletion time packed up to minutes (see CDPKTM)
6	NKEEP	I	Number of partitions to be kept
7	CHOPT	H	Character option
NWDOP+7	PATHN	H	Path name of the directory
..	..	H	

The bank structure created in memory by HEPDB is show below.

```
           (3)   +-------\
       +---------|  FZDB  >   List of directories to be updated
       |         +-------/
       |
 +--------\      +--------\
 |  UPDB   >-----|  UPDB   >  Support for all top directories opened
 +--------/      +--------/
   |   |
   |   |   (2)   +--------\
   |   +---------|  DICT   >  Dictionary information
   |             +--------/
   |
   |       (1)   +--------\
   +-------------|  NODB   >  Node bank for the top directory
                 +--------/
                  |..|..||
                     |
                     |
                 +--------\
                 |  NODB   >
```

```
+--------/
 |....|||
      |      Node bank of subdirectory for which data
+--------\   +--------\   +--------\   is retrieved
|  NODB   >--|  KYDB   >--|  KYDB   >
+--------/   +--------/   +--------/   Key banks
                 | (1)        | (1)
             +--------+   +--------+
             |  DATA  |   |  DATA  |
             +--------+   +--------+
```

Bank description

```
=========================================================================
|Bank:  UPDB                                              Top level bank |
|NL_/NS_ =  2/2                                            IO_ = '8I -H' |
|NW_     =  12                                                           |
+-----------------------------------------------------------------------+
|LINKS:                                                                  |
|link   type   bank                                       offset         |
|----   ----   ----                                       ------         |
| -3     Ref   FZDB                                       KLFZDB ( 3)    |
| -2     Str   DICT                                       KLDICT ( 2)    |
| -1     Str   NODB                                                      |
|  0     nxt   UPDB of the next data base file                           |
+-----------------------------------------------------------------------+
|DATA WORDS:                                                             |
|word  type  contents                                     offset         |
|----  ----  --------                                     ------         |
|   1    I   Logical unit number of RZ file               MUPLUN ( 1)    |
|   2    I   Flag if database to be updated (0 if not)    MUPFLG ( 2)    |
|   3    I   Logical unit number of standard journal file MUPJFL ( 3)    |
|   4    I   Logical unit number of special backup file   MUPBAK ( 4)    |
|   5    I   Identifier of the top directrory             MUPDIC ( 5)    |
|   6    I   Number of characters in the top directory    MUPNCH ( 6)    |
|            name                                                        |
|   7    I   Shared/server flag (IOPS*10 + IOPP)          MUPSRV ( 7)    |
|            (IOPS = 1 if S option in DBINIT;                            |
|             IOPP = 1 if P option in DBINIT)                            |
|   8    I   Maximum insertion time for subsequent        MUPKY7 ( 8)    |
|            object retrieval                                            |
|9-12    H   Name of the top directory                    MUPNAM ( 9)    |
+-----------------------------------------------------------------------+
```

```
=========================================================================
|Bank:  DICT                                             Dictionary bank |
|NL_/NS_ =  0/0                                      IO_ = '1I /3I 22H' |
|NW_     =  1 + 25*n                                                     |
+-----------------------------------------------------------------------+
|DATA WORDS:                                                             |
|word  type  contents                                     offset         |
```

```
|----  ----  --------                                     ------     |
|   1    I   Number of nodes in the dictionary            MDCNTM ( 1) |
|      For each node (Node number n)                                  |
|IOFF+      (= (n-1)*NWITDB + 1)  (NWITDB = 25)                       |
|   1    I   Unique identifier of the node                MDCITM ( 1) |
|   2    I   Number of characters for describing the path  MDCNCH ( 2) |
|            to the node                                              |
|   3    I   Last update to the node (not avaialable yet)  MDCLUP ( 3) |
| 4-5    H   Alias name                                   MDCALI ( 4) |
|6-25    H   Name of the path to the node (excluding the   MDCNAM ( 6) |
|            top directory part)                                      |
+--------------------------------------------------------------------+


======================================================================
|Bank:  NODB                                               Node bank |
|NL_/NS_ =  NS_/(number of down nodes)            IO_ = '4I 16B -H' |
|NW_     =  20 + words needed for path name                           |
+--------------------------------------------------------------------+
|LINKS:                                                               |
|link   type   bank                                       offset      |
|----   ----   ----                                       ------      |
| -n     Str   NODB (next level node)                                 |
|  0     nxt   KYDB of the first key bank to the node     KLDYDB ( 0) |
+--------------------------------------------------------------------+
|DATA WORDS:                                                          |
|word  type  contents                                     offset      |
|----  ----  --------                                     ------      |
|   1    I   Number of key elements for this node         MNDNWK ( 1) |
|   2    I   Total number of data words in the Key bank   MNDNWD ( 2) |
|   3    I   Number of characters describing the path to  MNDNCH ( 3) |
|            the node                                                 |
|   4    I   Unique identifier of this node               MNDDIC ( 4) |
|5-20    B   IO descriptor of the Key bank                MNDIOF ( 5) |
|21-..   H   Name of the path to the node                 MNDNAM (21) |
+--------------------------------------------------------------------+


======================================================================
|Bank:  KYDB                                                Key bank |
|NL_/NS_ =  3/1                                      IO_ = Dynamic |
|NW_     =  NWKEY + NWFXM(=6)                                         |
+--------------------------------------------------------------------+
|LINKS:                                                               |
|link   type   bank                                       offset      |
|----   ----   ----                                       ------      |
| -2     Ref   UPDB (Top level bank)                      KLUPDB ( 3) |
| -2     Ref   NODB (parent node bank)                    KLNODB ( 2) |
| -1     Str   Data bank                                  KLDADB ( 1) |
|  0     nxt   KYDB of the next key bank                              |
+--------------------------------------------------------------------+
|DATA WORDS:                                                          |
|word  type  contents                                     offset      |
|----  ----  --------                                     ------      |
|   1    I   Serial number of the object                              |
|   2    I   Refernce to the master object (for update)               |
|   3    I   Start validity time (upto seconds)                       |
```

```
|   4    I   End   validity time (upto seconds)                          |
|   5    I   Source identifier                                           |
|   6    I   Flag for storing the object (internal to HEPDB)             |
|             Bit JRZUDB (=1) : Full RZ option                           |
|                 JIGNDB (=2) : Ignore the object                        |
|                 JPRTDB (=3) : Directory is partitioned                 |
|                 JASFDB (=4) : Specially encoded ASCII                  |
|   7    I   Insertion time (upto minutes)                               |
|8-NWKEY     User keys                                                   |
|NWKEY+1 I   Logical end validity time (upto seconds)                    |
|NWKYDB+                                                                 |
|  -4    I   Number of physical reads to disk for this key MKYRID (-4) |
|  -3    I   Number of calls to DBUSE in the same event    MKYCEV (-3) |
|  -2    I   Number of calls to DBUSE in the entire run    MKYCRU (-2) |
|  -1    I   Precision used for storing the object         MKYPRE (-1) |
|   0    I   Free flag (set by DBFREE call)                MKYFRI ( 0) |
+-----------------------------------------------------------------------+


=========================================================================
|Bank:  FZDB                        List of directories to be updated |
|NL_/NS_ =  0/0                                           IO_ = '-H' |
|NW_     =  4 + 20*n                                                   |
+-----------------------------------------------------------------------+
|LINKS:                                                                 |
|link   type   bank                                        offset      |
|----   ----   ----                                        ------      |
|  0     nxt   FZDB of the next data base file                         |
+-----------------------------------------------------------------------+
|DATA WORDS:                                                            |
|word  type  contents                                      offset      |
|----  ----  --------                                      ------      |
| 1-4    H   Top directory name                            MFZTOP ( 1) |
|      For each directory (number n)                                    |
|IOFF+      (= (n-1)*(MXLWDB+1) + MFZDIR)  (MXLWDB = 20; MFZDIR = 5)   |
|  1     I   Number of characters in the path                          |
|2-21    H   Complete pathname of the directory or the root            |
+-----------------------------------------------------------------------+
```

Index

Bibliography

[1] L. Lamport. *LaTeX A Document Preparation System (2nd Edition)*. Addison-Wesley, 1994.

[2] CN/ASD Group and J. Zoll/ECP. *ZEBRA Users Guide, nProgram Library Q100*. CERN, 1993.

[3] CN/ASD Group. *HBOOK Users Guide (Version 4.21), nProgram Library Y250*. CERN, January 1994.

[4] CN/ASD Group. *PAW users guide, nProgram Library Q121*. CERN, October 1993.

[5] M. Brun, R. Brun, and F. Rademakers. *CMZ - A Source Code Management System*. CodeME S.A.R.L., 1991.

[6] CERN. *CSPACK – Client Server Computing Package, nProgram Library Q124*, 1991.

[7] J. D. Shiers. *FATMEN - Distributed File and Tape Management, nProgram Library Q123*. CERN, 1992.

[8] L3 Collaboration. The L3 database system. *Nuclear Instruments and Methods in Physics Research*, A309:318–330, 1991.

[9] R. Cranfield, B. Holl, R. W. L. Jones, and N. Watson. *OPCAL User Guide – OC504*. OPAL/OFFL/36/0003, 1991.

[10] R. Brun. *ZEBRA - Reference Manual - RZ Random Access Package, nProgram Library Q100*. CERN, 1991.

[11] J. Zoll. *ZEBRA - Reference Manual - MZ Memory Management, nProgram Library Q100*. CERN, 1991.

[12] M. Goossens. *ZEBRA - Overview of the System and DZ Reference Manual, nProgram Library Q100*. CERN, 1991.

[13] J. Zoll. *ZEBRA - Reference Manual - FZ Sequential I/O, nProgram Library Q100*. CERN, 1991.

[14] CN/ASD Group. *GEANT Detector Description and Simulation Tool, Version 3.16*. CERN, January 1994.

[15] CN/ASD Group. *KUIP – Kit for a User Interface Package, nProgram library I202*. CERN, January 1994.