The SNO Database: SNODB v3_07_01

Doug Cowen (Penn), Guy Jonkmans (Neuchatel), Rob Komar (UBC)
and Reda Tafirout (Laurentian)

August 26, 2001

ABSTRACT

The SNO database provides a repository for constants associated with the SNO dataset and for status logs with time-dependent information on components and processes. The repository of constants will be read from and written to principally by the DAQ and CMA systems, SNOMAN and calibrators. Database updates will be disseminated to the entire collaboration as frequently as necessary. The SNO database will have a web-based graphical user interface which will enable users to monitor the behavior of various database objects as a function of time. HEPDB, a CERN product, provides the fundamental building blocks for the database.

This document is also available on the web at `http://www.hep.upenn.edu/SNO/`.

Contents

1. Introduction

The HEPDB package comes in two distinct parts: the main part operates on the database objects themselves, and a subsidiary part communicates database changes to multiple copies of the database at various locations (see Fig. 1). Database objects (e.g., a ZEBRA bank) may be manipulated in a variety of ways. In normal operations, these objects are read, written and/or modified. The communication package allows one to implement a variety of schemes in which database modifications are propagated to collaborating institutions.

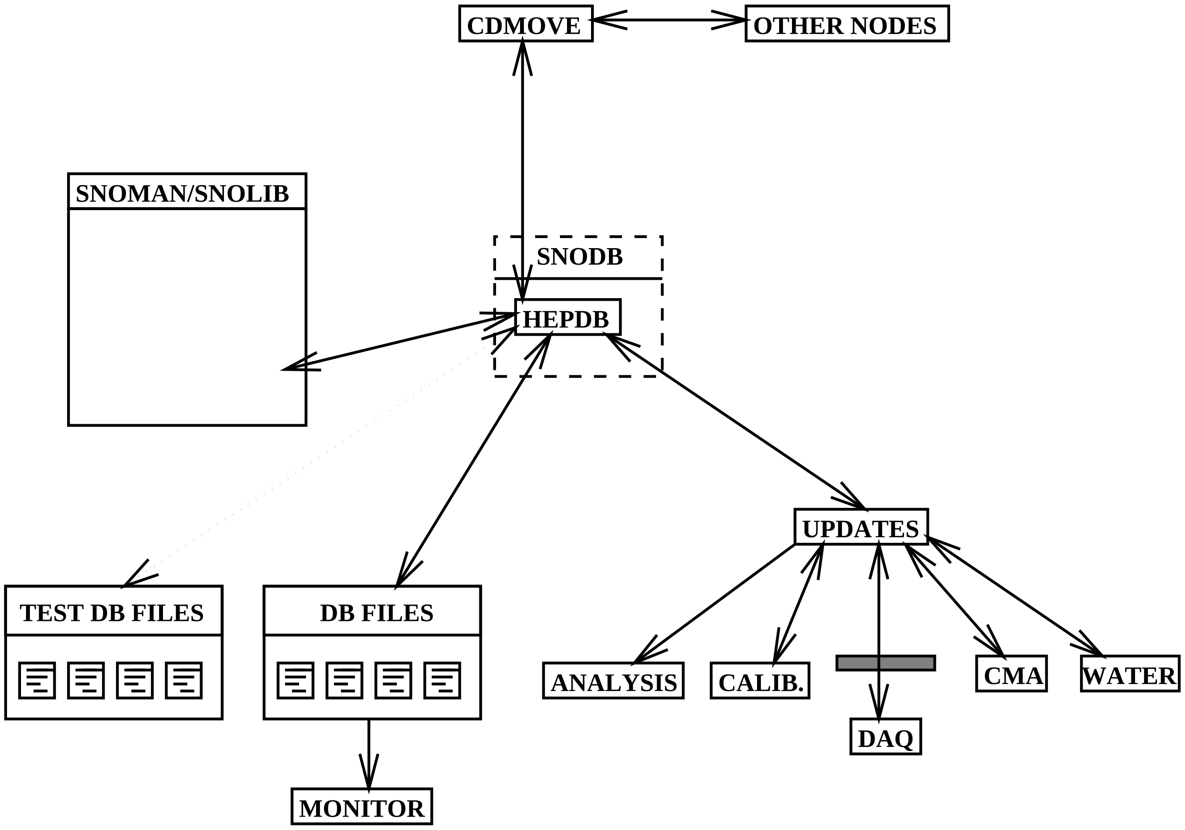

Fig. 1. Overview of the SNODB system. See text for details.

The fundamental ways database objects may be manipulated are well documented in the HEPDB manual so they will not be discussed further here. The same cannot be said for the communication package, so we are providing more detail in Section 3 for the interested reader.

This document is organized as follows. Appendix A describes in detail how to install and locally configure the SNO database. **Users are strongly urged to install the database prior to reading this manual.** Section 2 describes the basic internal and external structure of the database. Section 3 describes the networking aspects of SNODB and also includes a description of the protocol used to insure that updates are made in a controlled fashion. Section 4 describes how the database is maintained, concentrating on how we will insure that updates have been properly propagated to the entire collaboration and how we will keep the database from getting cluttered with outdated information. Section 5 gives a description of how we will verify that the data submitted to the database is correctly inserted in their entirety. Section 6 details the way in which backups of the database need to be made. Section 7 details how the database interacts with SNO systems such as DAQ and CMA, and gives the general procedure for writing to and reading from the database via automated external processes. This is in contrast to the manner in which individuals implement updates, make a local private version of database file(s) for testing purposes, or find out information about the status of various aspects of the database, described in Section 8.

2. Database Structure

The SNO database will be comprised of multiple files in multiple directories. In addition, each database file has an internal tree-like structure using a UNIX-based format for specifying directories and subdirectories. A multiple file database enables us to separate frequently-changed elements of the SNO database from those which are not, making the update procedure more robust and giving the database external as well as internal structure. Multiple files also give individual institutions more flexibility in the use of their available diskspace, since individual database files may reside on different disks, and make it possible to guard against erroneous data entries. The chief disadvantage of multiple files comes in the code which has to open all the database files at SNOMAN initialization time, and in the code which has to pick the particular file in which a particular object resides, but this is transparent to the average user and does not add significant processing time overhead.

Each database file will be internally organized into directories and subdirectories as specified by the creators of the data which reside therein. Database objects contained in these subdirectories are comprised of a "keys" bank and an attached data bank.* This is illustrated in Fig. 2.

The data contained in the keys banks which are pertinent here are:

1. a temporal validity range (GMT *must be used*),
2. an insertion time (automatically added by HEPDB when the object is written), and
3. several "user" keys.

For SNODB v3_07_01, these user keys include the following pieces of information:

- data type (a combination of detector type [SNO or MiniSNO] and data source [MC or real], as shown below).

data_type	=	10*(IDDT_REAL or IDDT_MC)
		+ 1*(IDDT_SNO or IDDT_MINISNO)

where

IDDT_SNO	=	1
IDDT_MINISNO	=	2
IDDT_REAL	=	1
IDDT_MC	=	2

- bank alias,
- bank number,

*Note that HEPDB allows one to use more complex data structures. However, we are restricting ourselves to the simplest possible structure so that we can easily interface the database to C. A C-callable fortran routine to fetch the object and place it in a structure in flat memory is much simpler to write if the objects themselves are flat. Note that it is not a limitation of C which impels us to do this, it is just a way to keep things simple.

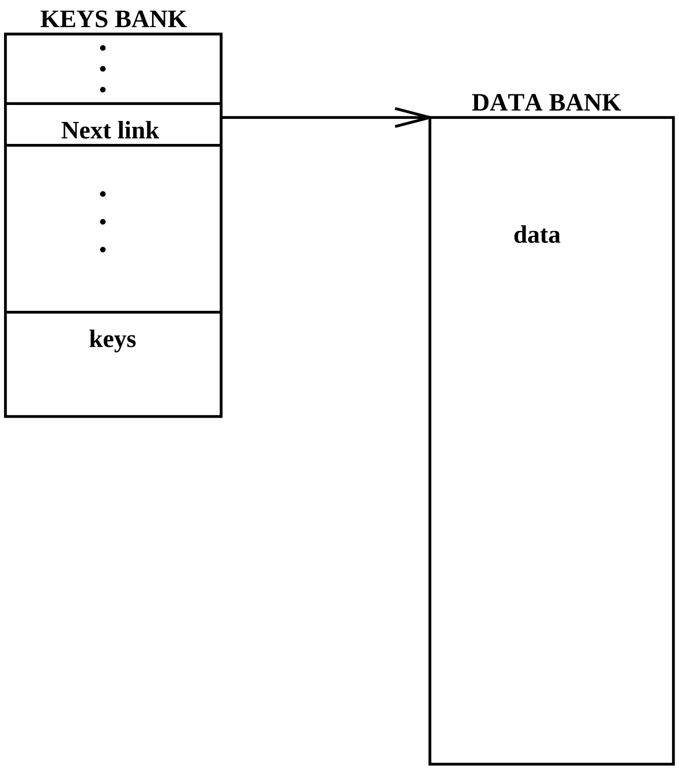

Fig. 2. A basic database object consisting of a keys bank which contains a pointer to its data bank.

- task type (e.g., are we looking at solar neutrino or upward-going muon events; we may wish to change the constants used by the fitter, say, depending on which type of event we are investigating),
- format number (initially 0 for all banks, changed whenever the format of the data in a bank is changed in a backward *in*compatible way, which hopefully occurs infrequently),
- source ID (identifies the source of the bank, e.g., DAQ or a particular calibrator).

This information is stored in several arrays in the SNODB code, as illustrated in Table 2. To read a bank from the database one must provide an instant of validity (most commonly, an event's time stamp) along with the bank alias and some number of additional keys. Using these items of information, HEPDB will then search for the most recently added database object–in this example a bank–which satisfies all the criteria. HEPDB can be forced to read an older version of a given object if one specifies the appropriate upper limit on the insertion time.

To write a bank, the same information is required except that a validity *range* must be provided instead of an instant of validity. The validity range provides the start and end times for which the information written to the database is considered valid. Greenwich Mean Time (GMT) is the standard time frame for SNO and SNODB.

There are several ways in which to read and write to the database using the tools described in Section 8. Information is passed to and from these routines by means of the `ipath` vector shown in Table 2.

The initial database structure is given in Appendix C. This structure reflects the fact

keydbs	
1	serial number*
2	pointer*
3	flags*
4	insertion time*
5	*reserved*
6	source identifier*
7	software ref. num.*
8	checksum 1
9	checksum 2
10	*reserved*
11	start validity (GMT)
12	end validity (GMT)
13	data type
14	bank alias
15	bank no.
16	task type
17	format no.
18	source ID

usrkeys	
1	data type
2	bank alias
3	bank no.
4	task type
5	format no.
6	source ID

ipath	
1	start date
2	start time
3	end date
4	end time
5	creation date
6	creation time
7	eff. start date†
8	eff. start time†
9	eff. end date†
10	eff. end time†
11	entry date
12	entry time
13	data type
14	bank alias
15	bank no.
16	task type
17	format no.
18	source ID

Table 1. The arrays containing information related to the SNODB keys. (*Indicates HEPDB filled quantities. †Indicates output values.)

that certain "constants" can change frequently with time, *e.g.,* PMT calibration constants, while other constants rarely if ever change, e.g., geometry constants. Such a structure is naturally implemented with separate files. The current database structure is given in the file `~/3_07_01/dbs_tools_f/sndirs.dat` which is reproduced the aforementioned Appendix. In this file, individual database files are given as `*.dbs`, followed by the internal HEPDB subdirectory and bank names.

3. Networking and Database Checking

The HEPDB database package comes with a distribution system for transmitting updates to and/or from a central repository. It is based on a master-slave protocol in which one node keeps the primary copy of the database, implements updates to that copy, and distributes updates to secondary copies at the slave nodes. Since it is critical to maintain a certain degree of control over the contents of the database, individual members of the collaboration will not in general be permitted to implement updates to the master database. Other large collaborations typically give control to one person (hereinafter referred to as the database "czar") who receives update submissions from individuals responsible for various segments of the database. After a reasonable number of these requests are accumulated and tested, they are incorporated into the master database, verified, and then sent out to the slave nodes and verified. Ideally, this should happen on roughly a monthly basis. Certainly more frequent updates will occur during detector commissioning.

The files and programs HEPDB uses to implement this master-slave protocol are described below:

- `cdserv` is the program which implements changes in local database files. It reads configuration information from the `hepdb.names` file in the directory pointed to by `CDSERV`.
- `hepdb.names` is an ASCII file that is used to configure each database management system. It is created and/or modified with the script `customize_hepdb_names.pl`, called by the main setup script, `snodb.perl`. Permissions for reading and writing to each database file are set and information about which nodes can modify each database file, where those files are, the paths to certain directories used for managing the database files, whether the current node is a master or slave server, etc., are stored in the names file.

  Some sample `hepdb.names` files are given in Appendix B.
- journal files are files that contain the information necessary for the server (`cdserv`) to make updates to the database files named in the file `hepdb.names`. They are produced by both master and slave servers.
- `CDSERV` is an environmental variable (UNIX) or a global symbol (VMS) that contains the name of a directory with a names file. It is accessed by each job that needs to find out from the `hepdb.names` file where the database files and directories are. Only one value for `CDSERV` is possible for a given server process.
- the `queue` directory is a particular directory (defined in the file `hepdb.names`) into which applications put journal files. If the application (e.g., a calibration) is running on a slave node, files in the slave's `queue` directory are moved to the master node by `cdmove` (see below).
- the `todo` directory is a particular directory (defined in the file `hepdb.names`) from which the local server reads journal files to be processed as updates to the local copy of the database. Updates are transferred to the `todo` directory on slave nodes from the master node by `cdmove`.
- the master server is the central repository for the overall database network system configuration, the place where updates received from slave nodes or created on the master node itself are applied to the official database and the place from which updates are propagated to all the slave nodes via `cdmove`. On the master server the `queue` and `todo` directories are the same. This means that, unlike slave nodes, updates which are made on the master node are processed by the master's `cdserv` without `cdmove` having to operate.
- a slave server is a server for which the `queue` and `todo` directories are different. A slave server can make an update to the official database only by submitting its update to the master node, which then propagates the update back out to the slave node (and all other slave nodes). A slave server cannot change its local official database without the update first going through the master server. (One can, however, make temporary local changes to the official database by using a "test database" as described in Section 8.2
- `cdmove` is a program which moves journal files (updates) between machines. The names of directories that are sources and targets for moved files are stored in the file `hepdb.names` on the machine on which the `cdmove` program is run (for SNO, this

machine will be the master server node). `cdmove` either moves journal files from a slave node to the master node when the slave node is submitting an update, or from the master node to the slave nodes once a suite of updates have been made and the database czar decrees the official database shall be updated.

`cdmove` makes use of the `.netrc` file residing on the master server. This file contains machine name, login username and login password information for each slave server. The typical username is `cdsno`. **Password changes in the `cdsno` (or equivalent) account must be propagated to the czar.**

Plans are underway to replace the `.netrc` file with a less obvious and encrypted file to enhance system security.

The master database server for SNO will probably be placed at a location with good access to the internet, with all other institutions acting as slave servers. The configuration for each server or application is determined by the `hepdb.names` file in the directory pointed to by the particular `CDSERV` variable which the job is given. When accessing information, a database application reads directly from the database file pointed to in the `hepdb.names` file. The `cdserv` program on the master server will only run when it is deemed appropriate by the database czar. Journal files resulting from applications requesting changes to the database will be automatically written to the `queue` directory by the database application through interaction with HEPDB and its `cdserv` program. Since the slave sites' `queue` and `todo` directories are not identical, the slave servers do not process the file and hence do not insert the changes into the slave servers' local database. Instead, the database czar, upon determining that a sufficient number of database updates are ready, starts the `cdmove` program, which in all likelihood will run on the master server. This program moves the journal files from the slaves' local `queue` directories to the `todo` directory of the master server.

Before the master server's `cdserv` program is allowed to process the journal files the database czar must check for the integrity of the submitted database changes. (It is for this reason that the master server's `cdserv` program does not run continuously; otherwise the journal files copied over by the `cdmove` program would be automatically inserted into the master server's official copy of the database by `cdserv`, and then propagated out to the entire collaboration by `cdmove`.) The way this check is done is as follows: a full "check" copy of the database is created, complete with a copy of all the requested changes in the form of the journal files copied from remote slave nodes by `cdmove` and copies of all the system interfaces' database write requests (see Section 7.2). A `cdserv` process, as well as all appropriate system interface routines which write to the database (e.g., DAQ, CMA, etc.), are started. These process the copied journal files and interface write requests, respectively, and thereby implement the submitted database changes to the database copy. Then, a comprehensive SNOMAN test job is run to check that the submitted updates are okay. Note that the SNOMAN test job has not yet been created, and probably will be only after we have a better idea what to look for.

Once the checking procedure is satisfactorily completed, the official `cdserv` process is started. The master server then processes the journal files, updating the master copy of the relevant database files. The master server also puts copies of the journal files in the appropriate directories for distribution to those slave sites that are identified in the master's `hepdb.names` file. The `cdmove` program looks in those directories, and moves the copies of the journal files to the `todo` directory at each slave site. An email message will be sent to local database contacts informing them that an update set has been propagated and that they can process the updates. Alternatively, a slave node can run the `cdserv` process continuously. By processing these files the official local copies of the database are updated.

The basic master-slave interaction for the case in which a slave server submits an update is shown in Fig. 3. This figure depicts the master-slave interaction for the illustrative case in which the slave server "slave1" submits an update. This update is processed by slave1 which puts the resulting journal file into its `queue` directory (step "1"). Some time later, the master node starts `cdmove` which moves the files from slave1 to its `todo/queue` directory (step "2"). Assuming the "check" copy checks out okay, the master node's server applies this update to the official database and places copies of the update in the form of journal files in special local directories, one for each slave node in the system (steps "3"). Some time later, `cdmove` places these files in the `todo` directories on the corresponding slave nodes (steps "4"). The `cdserv` processes running on the slave nodes then apply the update to the local copy of the official database (steps "5").

A process dedicated to serving the DAQ read requests will run continously on the "site node" (a slave server running on site) and as such will be monitored but will not be regularly stopped and started by the database czar. This is because the DAQ will be making read requests whenever it wants to load in new constants for the electronics (say), and it should not have to wait for the next time the database czar decides to update the database. Database reads use the program `sdb_output_titles` which does not generate journal files nor does it require the `cdserv` process to be running, so reads will have no impact on the regular operations of the other parts of the database.

For reasons of maintenance convenience, the master node resides at the same place where the czar resides (presently, this is at Penn). Under this scenario, updates made on site–of which there will be many, especially during the commissioning phase–will not actually appear in the site's official copy of the database until the minimum time it takes for updates to propagate from the site, to the master, and back again. For large update sets, this time lag will be too long.

To overcome this problem, a "mirror" database has been implemented on site. The SNODB mirror consists of a separate server on site, running as a master, along with a process which manages the placement of local updates in the relevant `todo` and `queue` directories. The mirror takes local updates and applies them to the site's mirror database immediately, as well as submitting the same updates for propagation to the master node. These updates will eventually get processed on the master node and propagated back to the site node, where they will be applied to the local official copy of the database. Since other nodes will also be submitting updates which we will want applied to both the site mirror and official databases, but we do not want updates which originated from the site node getting *re*-applied to the site mirror database, a special filter has been implemented. The mirror database is depicted diagrammatically in Fig. 3. See Sec. 4.4 for details on how to set up the mirror on the site node. (Users at other nodes should avail themselves of a test database if they wish to see locally-generated updates quasi-instantly.)

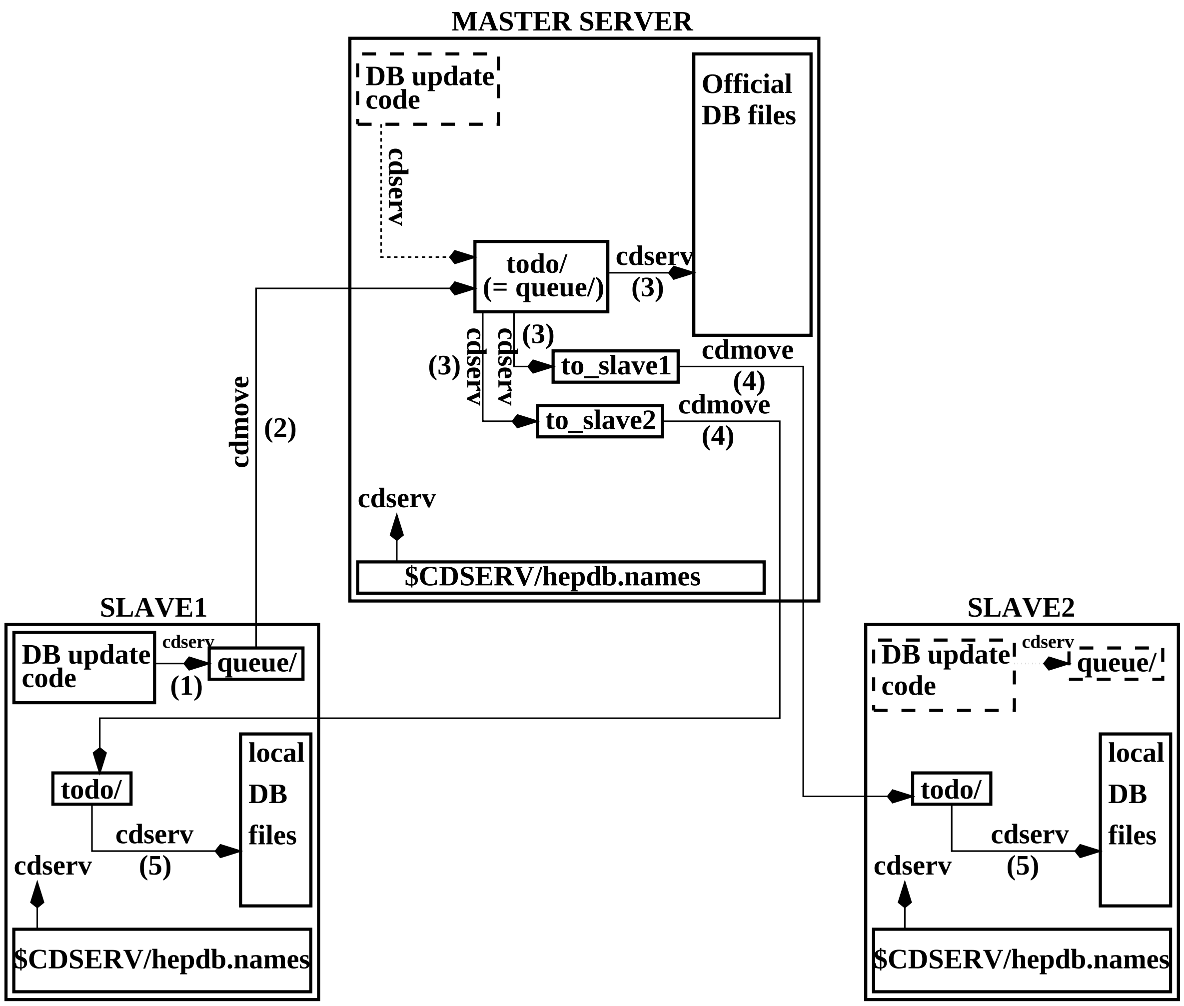

Fig. 3. The master-slave interaction for the illustrative case in which the slave server "slave1" submits an update. The subsequent train of events, labelled numerically, is described in the text. Updates may also be initiated from the master server or another slave server, as indicated by the dashed lines in the figure. The `hepdb.names` file on each server is used by the local `cdserv` process to determine the locations of all relevant directories, etc. Note that all of the lines except those internal to slave1 and slave2 may be connected or disconnected under the control of the database czar. Those lines internal to slave1 or slave2 are under the control of the local database maintainer at each slave node.

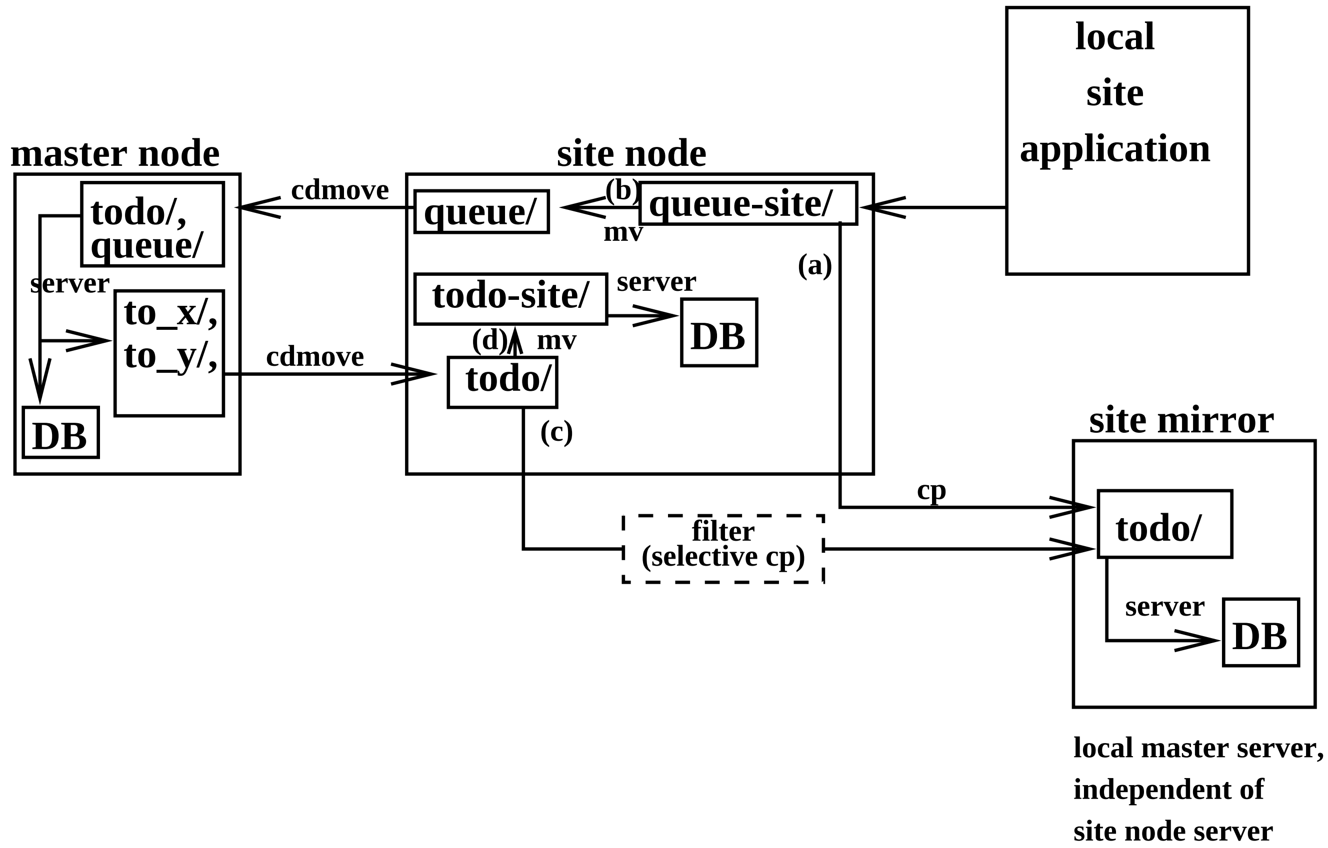

Fig. 4. Diagram of the mirror database implementation.

4. Database Maintenance

The information in this section is intended for maintainers of the database on the master and slave nodes. For standard user interfaces to the database, please see Secs. 7 and 8.

4.1. Slave Node Database Checking

To insure that updates have been propagated to all the nodes, one needs to get a summary of the current state of each local database. Since automating this procedure would be difficult, a manual procedure using code which provides a "snapshot" of the database has been written. The code is run via `~/3_07_01/snodb.perl -snapshot` and produces a file which contains a formatted summary of the contents of the local database. This summary may then be compared to a similar summary of the master copy of the database.

A snapshot is created and mailed to the database czar automatically after each time a slave node runs `~/3_07_01/snodb.perl -process_updates`. The czar compares this with the master database's snapshot.

4.2. Database Purging

Every so often, outdated data needs to cleared out of the database. HEPDB has a procedure which does this (`HDBEXTR`), but on closer inspection it turned out to be... vaporware! It is possible to write custom procedures to do this, but this is not needed until further down the road when the database has been used for a while.

A simple way to do this would be to create the purged version of the database file in question on the master node and then have slave nodes pick it up via downloads from the master node's web page at `http://www.hep.upenn.edu/SNO/snodb/main.html`.

4.3. Adding and Removing Slave Nodes

To add or remove a slave node from the system requires that the database czar stops the master node's `cdserv` process. Then, the master's `hepdb.names` file needs to be changed accordingly via the `init.pl` script. Specifically, to add a node:

- Have the new slave node's database contact person install the current version of the database.
- Edit `~/$SNODB_VER/scripts/snodb_lib/init.pl` on the master node, adding the new node to the node list.
- Run `~/$SNODB_VER/scripts/snodb.perl -dir` on the master node to add the new `to_node (localq)` subdirectory.
- Add the new node to the `.netrc` file with the appropriate information.
- Run `~/$SNODB_VER/scripts/snodb.perl -names` on the master node to create a new version of the `hepdb.names` file.
- For the case in which the master node and slave node platforms are running the same OS (or if another slave node is running the same OS), one can simply copy the database files. For different OS's, we had hoped to use the HEPDB routines `HDBTOFZ` and `HDBFRFZ`. These subroutines convert HEPDB native format files to and from FZ exchange mode format files. Regrettably, these documented routines do not exist, so one must rebuild the database using all the journal file backups made on the master node, putting them into the correct `localq` directory and propagating them out to the slave node being added.

The next time cdserv is started, it will read this new hepdb.names file and the new slave node will be treated like the other nodes. If you wish to make the change take effect immediately, you must stop and start cdserv via ~/$SNODB_VER/scripts/snodb.perl -stop_server and ~/$SNODB_VER/scripts/snodb.perl -server, respectively.

To remove a node,

- Edit ~/$SNODB_VER/scripts/snodb_lib/init.pl on the master node, removing the node from the node list.
- From the command line, delete the to_node subdirectory.
- Run ~/$SNODB_VER/scripts/snodb.perl -names on the master node to create a new version of the hepdb.names file.

If you wish to make the change take effect immediately, follow the instructions above for stopping and starting cdserv.

Changing the characteristics of a particular node involves an obvious combination of the two procedures outlined above.

4.4. Setting Up the Mirror Database

To set up the mirror for the first time, set the variable database_mode to "mirror" in snodb.config. Run snodb.perl -dir and snodb.perl -names to create the directory structure and hepdb.names file used by the mirror. Then, copy the existing database files over from the official database directory to to the mirror database directory, typically the same pathname as the official database files but with "dbsm" replacing "dbs" (this is set in snodb.config).

Start the mirror database server by issuing the command snodb.perl -server. Unlike with the slave nodes, which should normally run snodb.perl -process_updates, which starts and then stops the server, the mirror node should have the server running continuously.

Set database_mode back to "official" in snodb.config.

Finally, run snodb.perl -mirror to start the process which monitors the new todo_site and queue_site directories for updates and handles them appropriately.

4.5. Adding a New Database File

A new database file is created using the sncreate program. Assume that the file we wish to create is called foobar.dbs and is referred to by the two letter abbreviation fb. (To add a new bank to this new database file, refer to Sec. 8.1.)

1. Stop the server if it is running by issuing the command
   snodb.perl -stop_server
2. Edit ~/$SNODB_VER/scripts/snodb_lib/customize_hepdb_names.pl to include a new line for the fb database file
3. Run snodb.perl -names to create a modified version of the hepdb.names file which includes reference to foobar.dbs
4. Run snodb.perl -server to restart the server.

Note that it may take a significant length of time before a new database file gets installed on all slave nodes. This is because there is no mechanism for automatically propagating file creation directives using standard HEPDB journal files. Hence, the process depends on individual database czars either copying the newly created database files from the web site or running `sncreate` at each slave node. Following that, automatic updates creating an internal directory structure and bank aliases would follow. Before updates containing references to banks contained in the new database file can be propagated, all these steps must be executed at all the slave nodes.

5. Database Verification

It is important that the integrity of the data in the database be frequently checked. With SNODB, this can be done automatically whenever a bank is retrieved. At each local site, a (yet to be written) utility will regularly check each bank in the database for corruption. User applications can also verify each bank as it is retrieved, to ensure that corrupted data is not used.

Verification is disabled by default in the SDB code. To enable it in SNOMAN, set the appropriate word to "1" in the TSDB bank in the `job_control.dat` file, or use the `$sdb_enable_verification` mnemonic in a command file. To enable it in non-SNOMAN code, the following call must be made prior to retrieving the bank:

```
      include 'sdb_option.inc'
      call sdb_util_put_option(ksdb_idx_verification,
     +     ksdb_opt_verify,iretc)
```

The data is verified using the following procedure. Before each Zebra bank is stored in the database, a checksum of its contents is computed and written to the database header within the bank. Each word in the bank is converted to CERN's exchange format when computing the checksum to ensure that the results are independent of the computer platform used to store the bank. When the data is read back using the `SDB_READ` routine, if data verification is enabled, then the checksum is re-computed and compared to the one stored in the header. A difference in the two checksums indicates that the data has been corrupted. In such a case, a message is printed and the bank is not returned to the user for use. The user should promptly notify the local database administrator and the SNO database czar so that the database can be repaired as quickly as possible.

The platform-independent checksum computation uses the Zebra bank I/O descriptor to determine the word type prior to converting it to exchange format. A similar operation takes place when the data is stored in exchange format so that it can be correctly distributed among the various supported computer platforms (with their various native data formats and byte-orderings). Those people that are responsible for inputting data to the database (calibrators, for example) must ensure that the I/O descriptor is correct, or the checksum will be incorrect and the data may be corrupted when transferred to other platforms.

6. Database Backups

HEPDB has its own built-in backup mechanism which consists of writing all journal files which are processed by `cdserv` to the `save` subdirectory. Any slave node which for whatever reason misses part or all of an update can be brought up to speed by copying the relevant files from the `save` directory to the corresponding `to_nodename` directory on the master node and then running `cdmove`.

This procedure is useful but can quickly get cumbersome when there are many files in the `save` directory. To make the recovery process easier a SNODB procedure makes a time-stamped backup file consisting of the most recent set of files written to the `save` directory. During this procedure it is crucial that the `cdserv` process on the master node is not running to avoid the possibility that the database gets changed during the backup. This is taken care of automatically in the script which runs the backup. Once the backup is complete, the files in the `save` directory are deleted. In this way, each backup file will represent the full set of individual updates associated with each official database update set. These backups are an integral part of the procedure which the database czar uses to perform the official database update.

In addition to the above procedure, all files associated with the database need to be backed up on a regular basis. It is imperative that disk-resident, several-versions-deep backup copies are available on the master node, and that a backup is made *prior* to any procedure which necessitates a SNODB version number change. The version number is incremented by changes in the SNODB scripts, an update implementation, or a database cleaning, or other things we have not yet thought of. It is also critical that regularly scheduled backups are made on (say) a weekly basis during commissioning, and thereafter on a monthly basis. The regularly scheduled backups are done as part of standard system procedure on the master node.

7. SNO System Interfaces

SNODB will interact with a number of SNO subsystems, primarily through reading and writing operations, so although this section describes the SNODB interfaces to CMA and DAQ, the interfaces are general enough to apply to other systems as well.

7.1. CMA Interface

CMA only needs to write information into the database. The CMA writing process run on the site server. The database czar starts up the CMA write process by typing

`~/$SNODB_VER/scripts/snodb.perl -cma_write`.

This process then runs continuously, looking for new files in standard "titles" format named `*.dat` to appear in the directory defined in `scripts/snodb_lib/init.pl`. The default is `~/write_to_database/cma/todo`. (Note that this directory structure is loosely modelled on the standard SNODB/HEPDB structure.) The files in the `todo` subdirectory are processed in time-ordered FIFO sequence by the `~/$SNODB_VER/exes/sdb_input_titles` program which reads in the files and submits the associated updates to the database.

To stop the CMA write process, type `snodb.perl -cma_stop` which creates a file named `signal.stop` in the appropriate directory and eventually causes the CMA write process to stop. The `signal.stop` file is then deleted.

7.2. DAQ Interface

DAQ needs to write to *and* (possibly) read from the database. The write process is controlled by a cron job which runs the script `~/3_07_01/scripts/copy_titles.pl` on a regular basis. This script only checks for so-called DQxx titles files which contain banks describing the current state of the detector. The script performs a number of checks on the titles files, synchronizes validity ranges such that the N^{th} file's infinite end validity timestamp is altered to be the $(N+1)^{th}$ file's start time validity timestamp, processes the files so that the mirror processor on site places them into the fast version of the database, and finally emails the czar to have the updates propagated to the collaboration as a whole.

The read process is started by typing `snodb.perl -daq_read`, which initiates a process which runs continuously, looking for for new files named `*.keys` to appear in the directory defined in `init.pl`. The default is `~/read_from_database/keys_to_dbase`. Once a file appears in this directory, it is copied to the `~/read_from_database/keys_to_dbase/todo` directory. These "keys" files contain information which pinpoints a bank in the database. Here is an example of such a file:

file contents	meaning
cst0	bank name
1	bank number
19980527 150705	instant of validity
n	(reserved word)

The information in the keys file is read by the `~/$SNODB_VER/exes/sdb_output_titles` program which goes to the database, searches for the bank which matches all the keys, and then prints out the bank in titles format onto the file `xxxx_n.dat` where `xxxx` is the bank name and `n` is the bank number. The keys file is renamed to mark it as having been processed and subsequent keys files, if they exist, are then processed.

To stop the DAQ read process, one types `snodb.perl -stop_daq_read` which then creates a file named `signal.stop` in the appropriate directory and eventually causes the DAQ read process to stop.

7.3. Standard Update Propagation

Propagation of update sets is typically handled by the rather involved script invoked by `snodb.perl -update_database`. This script performs a variety of actions and checks to insure that a given update set is properly processed: updates are gathered up from all the slave nodes, checked on the master node, inserted into the official copy of the database on the master node, and finally propagated out to each slave node. This last step includes an email message sent to the local database maintainers, informing them that an update set is ready to be processed (which can be done via `process_updates`, see Sec. 8). The code is well documented and interested individuals are welcome to look at `update_database.pl` for more details.

8. SNODB User Interfaces

This is a list of tools for use by local database maintainers, calibrators, shifters, and the average user. While low level HEPDB routines are of course available (see the HEPDB User Manual[2] for details), we strongly recommend against using them without great care. Generally speaking, we would prefer to make a new tool at users' requests rather than have people crashing around in the database with their own HEPDB-based code.

Here are the main tools:

1. `sdb_compare`: Allows one to compare the contents of two banks or a bank and a titles file. **Fuller documentation in preparation.**

2. `process_updates` (called via `snodb.perl -process_updates`): Processes updates waiting in the local todo/ directory by starting the server, waiting for the updates to be placed in the local official database, and finally stopping the server. Also automatically creates a snapshot of the local copy of the database and emails it to the czar, enabling the czar to insure that the local database is in step with the master database.

3. `sdb_input_titles` (called via `snodb.perl -input_titles`): Prompts user for either the name of a titles file or the name of a file containing a list of titles files. A check is performed to see what databases the user is allowed to write to. Then the information is fed to `~/$SNODB_VER/exes/sdb_input_titles`.

   A description of the CERN titles format is given in Appendix D.

   You may also use `~/$SNODB_VER/exes/sdb_input_titles` directly to input titles files, but that will not do checking of write permissions. The following input switches may be provided on the command line:

   - `-f --format /format /FORMAT`
     A titles format bank is also input into the database. This is the default action.
   - `-n --no-format /noformat /NOFORMAT`
     No titles format bank is input into the database.
   - `-i --input /input /INPUT`
     The next entry in the command line is the name of the titles file to be input.
   - `-l --list /list /LIST`
     The next entry in the command line is the name of the file containing the list of titles files to be input.
   - `-d --directory /directory /DIRECTORY`
     The next entry in the command line is the full pathname of the directory containing the titles file(s). If this option is omitted, then the current working directory is used.

4. `sdb_output_titles` (called via `~/$SNODB_VER/exes/sdb_output_titles`): Accepts as input the alias of a bank and an instant of validity and outputs a titles bank containing the requested bank.

   - `-d --data /data /DATA`
     The output file only contains the data stored in the bank. Thus, all comments and titles control lines are stripped out.
   - `-b --bank /bank /BANK`
     The next two entries in the command line are the name and number of the bank to be output.
   - `-v --validity-inst /validity_inst /VALIDITY_INST`
     The next two entries are the date (YYYYMMDD) and time (HHMMSS) for which the output bank is valid.
   - `-n --no-query /noquery /NOQUERY`
     Turns off the query for more output titles files at the end (the default is for the query to take place).

5. `snapshot` (called via `snodb.perl -snapshot`): Creates a snapshot of the database pointed to by `$CDSERV/hepdb.names`. The snapshot consists of two snapshot files containing "full" and "brief" outputs. The "full" output contains, for each insertion of each bank in the database, the bank name, number, serial number, insertion time and validity range. The "brief" output omits the insertion time (useful because the insertion time will change from node to node, so by omitting it the resulting file can be compared directly node-to-node).

6. `sncreate` (called via `snodb.perl -sncreate`): Creates database files based on the contents of the `hepdb.names` file. If a filename found in `hepdb.names` already exists in the official database directory, it is not created.

7. `sndirs` (called via `snodb.perl -sndirs`): Creates the database files' internal structure one internal directory at a time based on command line arguments, or for mulitiple directories residing in multiple files according to the contents of a file, e.g., `~/$SNODB_VER/dbs_tools_f/sndirs.dat` by using `-l filename`, `--list filename`, `/list filename` or `/LIST filename`. A sample `sndirs.dat` file, containing comments which explain what information is needed to create internal directories, is provided in Appendix C.

8. `snalia` (called via `snodb.perl -snalia`): Creates the aliases for database banks on bank at a time based on command line arguments, or for mulitiple banks residing in multiple files according to the contents of a file, e.g., `~/$SNODB_VER/dbs_tools_f/sndirs.dat`, in the same manner as specified for `sndirs` above. A sample `sndirs.dat` file, containing comments which explain what information is needed to create bank aliases, is provided in Appendix C.

Here are some additional tools which will probably find less frequent usage. These tools are under development.

1. obtaining a history of a given bank, i.e., insertion time, validity range, effective validity range, etc. Maybe just a nice printout of all the keys does the trick,
2. obtaining a history of a given class of banks, i.e., calibration banks,
3. obtaining deltas between two versions of a given bank,
4. seeing pictorially the overall structure of the database, or for one file in the database,
5. a subroutine which could be called repeatedly and which, on each call, returned the keys of the next bank in the database. This subroutine could also accept a bank name or perhaps a 2-character database abbreviation with which to limit the looping.

8.1. Adding a New Bank to the Database

If one wishes to add a new bank to an existing database file, the steps to follow are below. (If one wishes to add a new bank to an entirely new database file, see Sec. 4.5.)

1. Create a file called (say) `my_sndirs.dat` and enter one line for each directory and/or bank you wish to insert. For example, the lines might look like:
   ```
   //CDFB/bnk1 bnk1 F
   //CDFB/bnk2 bnk2 F
   //CDFB/stuff
   //CDFB/stuff/bnk3 F
   ```
   where `FB` is the two-character database file prefix of the file to which you are adding the new bank. This anticipates three banks, `bnk1` and `bnk2` in the top directory and `bnk3` in the `stuff` subdirectory. The `F` qualifier directs `sndirs` to make an `fmt` subdirectory to store that bank's format information. This format information is used to write out the bank as a titles file in the same format it had when it was input.

2. Run `~/$SNODB_VER/exes/sndirs -l my_sndirs.dat`
   to create the internal directory structure.

3. Once the journal files for the internal directory structure have been processed, run
   `~/$SNODB_VER/exes/snalia -l my_sndirs.dat`
   to create the aliases for the banks. These aliases are `bnk1`, `bnk3` and `bnk3` which refer to the banks by the same name and point to their locations within the internal directory structure of the database file in which they are placed.

8.2. Making a Local Test Database

Individuals who submit database updates are required to check them locally before submission. In some cases, several iterations of this procedure at the local site may be needed. SNODB has anticipated this requirement/need by allowing users to set up easily a local test version of all or part of the database. A test database will normally be configured such that one of its files–the one with the data under test–is copied from its official location to the test location and placed under local user control. The remaining database files are accessed from their official location in readonly mode.

The following series of steps should be followed in order to set up a test database.

1. Copy over all the files in the official `~/3_07_01` directory to your own private area, i.e., `my_area/cdsno/3_07_01`. All the following instructions assume that you are editing the files in this area. (It is assumed that the platform on which the official database is running is the same as the platform on which you wish to run the test database. If not, you need to set up the entire database from scratch.)

2. Set the following environment variables:
   - `$SNODB_ROOT` to `my_area/cdsno`
   - `$SNODB_VER` to `3_07_01`
   - `$SNODB_CODE` to the official SNOMAN code area

3. Edit the file $SNODB_ROOT/3_07_01/scripts/snodb.config and make the following changes:
   - Set "`official or test?`" to "`test`"
   - If you are not in the "snodbw" group, which you can check by typing the unix command `groups`, change `snodbw` to `sno`. (You should be in the `sno` group!)
   - Set `parent_dir` to `my_area`

4. Insure that the disk area chosen for the test database files exists and is appropriate. This area is specified in `$SNODB_ROOT/3_07_01/scripts/snodb.config`. The area should be one which is writeable by you, so chances are you will need to create it. To do this, edit `snodb.config`, changing the line

   ```
   "test dbase file dir" , "dbst_dir"
   ```

   with the new directory instead of `dbst_dir` and then run `$SNODB_ROOT/3_07_01/scripts/snodb.perl -dir`.

5. Select the particular database files to place under local control by modifying `$SNODB_ROOT/3_07_01/scripts/snodb.config`. Place a list of 2 character database abbreviations, e.g., `xx` and `yy`, separated from one another by a single space, at the appropriate line as shown:

```
"dbase files to test" , "xx yy"
```

The actual database filenames corresponding to the abbreviations like `xx` and `yy` can be found in `$CDSERV/hepdb.names`, `$SNODB_ROOT//3_07_01/dbs_tools_f/sndirs-full.dat`, or `$SNODB_ROOT//3_07_01/scripts/snodb_lib/init.pl`.

6. Set the `$CDSERV` environment variable to point to the directory where you wish to place the new `hepdb.names` file which you will create in the next few steps. This can be any directory where you have write permission. A recommended location is `my_area/cdsno`.

7. Set the default directory to `$CDSERV` and create a new `hepdb.names` file by running `$SNODB_ROOT/3_07_01/scripts/snodb.perl -names`

   If you are adding a completely new file (as opposed to updating a bank in an existing database file, or adding a new bank or directory to an existing database file) you must edit the file `my_area/cdsno/3_07_01/scripts/snodb_lib/customize_hepdb_names.pl`. In this file, you must:

   - Add a new two character abbreviation for your new database file to the list which is prefaced by `:list.`
   - Add a new database description similar to those existing already. These descriptions are prefaced by `:nick.xx` where `xx` is the two character database abbreviation.

8. Start the server by issuing the command `$SNODB_ROOT/3_07_01/scripts/snodb.perl -server`

9. Create `$SNODB_ROOT//3_07_01/dbs_tools_f/sndirs.dat` as described in in Section 8.1 if you have any new banks and/or directories. The `sndirs.dat` file should reside in your personal area.

10. Run `$SNODB_ROOT/3_07_01/scripts/snodb.perl -build_test` which will copy over the relevant database files from the official directory and then read in the `sndirs.dat` file to create new directory structures and bank aliases in the test database files if requested.

11. When `$CDSERV` is set as above, updates you wish to test (e.g., via sdb_input_titles) will be applied to your own copy of the appropriate `.dbs` file(s). You may then run your test SNOMAN job (again with `$CDSERV` set as above) to analyze the update(s).

A. Installation

The current version of SNODB is supported on UNIX and on VMS platforms. One local copy of the database can serve both platforms as long as the files can be accessed via NFS.

Before you can use the SNO database you must obtain and install version 96a or higher of PACKLIB (part of CERNlib). For Linux platforms, version 97a is required. Version 96a currently has some bugs (`cdfrst`, `cdfsnd` and `cdsopn`) but a set of `cd*.f` routines is supplied with SNODB to fix these known problems. Note that certain operating systems are no longer supported under 96a, such as SunOS (hence, OS users need to switch to Solaris).

The main engine for installing SNODB is a Perl script. Perl is a language for easily manipulating text, files and processes. Many UNIX system managers use Perl to control complex system-related tasks. Perl is freeware and widely available, and may already be present on your site. You must have Perl version 5 (*aka* Perl5) on your system to install SNODB (Perl4 won't work). To find out if you have it, type `perl -v`. Hopefully, you will get an answer like:

```
This is perl, version 5.005_03 built for sun4-solaris

Copyright 1987-1999, Larry Wall

Perl may be copied only under the terms of either the Artistic License or the
GNU General Public License, which may be found in the Perl 5.0 source kit.

Complete documentation for Perl, including FAQ lists, should be found on
this system using `man perl' or `perldoc perl'.  If you have access to the
Internet, point your browser at http://www.perl.com/, the Perl Home Page.
```

If not, Perl is either not in your path or it is not present on your machine. The next step is therefore to ask your system manager if Perl is on your system. If it isn't, or you only have Perl4, then have Perl5 installed by following the instructions in subsection 1.1. If Perl is present, but the command above didn't work, then

UNIX: add the name of the directory in which the Perl executable is kept to your path. If it isn't in /usr/bin/perl, please have your system manager set up a softlink so that /usr/bin/perl points to wherever your Perl is located. (The scripts distributed in the SDB package assume that Perl is located in the /usr/bin directory, but you can change this by editing the first line of the `snodb.perl` file accordingly.)

VMS: either get your system administrator to make system-wide global symbols pointing to the Perl images, or set them up for yourself in your login.com file. For example, if the Perl root directory is `sys$share:[perl5_003_04]`, then define:

```
$ define/translation=concealed Perl_Root sys$share:[perl5_003_04.]
$ define PerlShr Perl_Root:[000000]perlshr.exe
$ Perl:==$Perl_Root:[000000]perl.exe
$ Perldoc == "''Perl' Perl_Root:[lib.pod]Perldoc -t"
```

If Perl is present you can skip to section 1.2.

1.1. Getting Perl

Perl is freeware. You get the best results if you get the Perl source code and build it on your local machine. This may sound like a lot of work, but the configuration scripts that come with the source code make it easy to build and install. You can get the source code via anonymous ftp as follows:

UNIX: look at http://www.perl.com/CPAN/CPAN.html to find a Perl Archive site near you and download the source code from there.

VMS: download the Perl5 source code from ftp://genetics.upenn.edu/perl5. It is especially tailored for VMS, and overcomes many limitations the generic CPAN version has on that platform. VMS-specific information on Perl, and what UNIX-like tools are required to build it can be found at
http://w4.lns.cornell.edu/~pvhp/perl/vms/VMS.html.

Each of the machines providing the source code keeps things in a slightly different way, so the best strategy is to look for a README file and follow the local instructions. At the time of this writing, the most recent version of Perl was >5.0. We are currently running v5.004_04 on surf.

For those of you who wish to know more about Perl, there are two excellent books,[1] web sites containing on-line manuals (http://www.eleves.ens.fr:8080/cgi-bin/perl-man and http://www.cs.cmu.edu:8001/htbin/perl-man), and at least one user group (comp.lang.perl).

1.2. Getting Started

Before shipping and installing the code on your computer you will need to create one account with name "cdsno". This account will host the SNODB code and will run the database server to process updates. We strongly recommend that you adhere to the following steps in creating this account. If you don't, it may be very difficult to include your institution in the networked version of the database. Many of the items below must be done by the system manager.

- Create an account with name "cdsno". This is the account under which the various database management servers will run, and will be used for database maintenance purposes.
- Create a "sno" group (or some other name of your choosing) which will be used to control who gets read access to the SNO database. Every account owned by someone affiliated with SNO should probably be in this group. "cdsno" should also be a member of this group.
- On UNIX systems, create a "snodbw" group which will be used to control who gets write privileges to the SNO database (on VMS systems, this is done using ACLs). "cdsno" should be a member of this group. Only people officially responsible for putting data into the database should be members of this group. Others, in general, should not be allowed to write to the SNO database in order to protect the integrity of your local database.
- Create a "hepdb" directory (owned by "cdsno" with permission 0750) somewhere with at least a few hundred MB of space (0750 corresponds to o:rwe,g:re,w: permissions in VMS). This is the directory which will contain all of the files and directories required by the HEPDB database servers. This directory is referred to as the parent directory.

You are now ready to ship and install the programs.

1.3. Shipping and Installing the Programs

In the following sections, the directory pathnames are given using UNIX syntax. Although the syntax is different on VMS machines, the name of each directory is the same. In the examples below, the user `cdsno`'s root directory is `/users/cdsno`, and the hepdb parent directory is `/hepdb`. In VMS, these might be `dkb$200:[cdsno]` and `dkb$200:[hepdb]`, respectively.

From your "cdsno" account, pick up `snodb_3_07_01.tar.gz` via anonymous ftp from the distribution site:

Site:	upenn5.hep.upenn.edu
Directory:	pub/cowen/snodb

From this directory copy the tar file

snodb_n_nn.tar.gz   where n_nn is the version number (*i.e.* 3_07_01)

into the cdsno root directory (*e.g.*, /users/cdsno). Run "gunzip" and "tar xvf snodb_n_nn.tar" to expand it (these utilities are available for VMS machines. Contact Rob Komar if you can't find them). You should now have the following directory structure in "cdsno":

```
./cdsno
        /3_07_01
                /cern_mods_f   ! Debugged CERN packlib routines
                /dbs_tools_f   ! The database tools
                /doc           ! Documentation
                /exes          ! dir for SNODB binaries
                /includes      ! include files common to /dbs_tools_f and /snoman_f
                /scripts       ! The main perl script plus other scripts
                /snoman_f      ! SNOMAN subroutines, alter at your own peril
```

Go to the /*_f directories and remove any "*.o" files, and then remove any files from the /exes directory.

1.3.1. Defining Your Local SNODB Configuration

The database setup proceeds through a single perl script:

```
~/3_07_01/scripts/snodb.perl
```

NOTE: If this does not work please edit the file and replace the top line with the location of your local node's perl executable. Prior to actually running snodb.perl you need to edit the file:

```
~/3_07_01/scripts/snodb.config
```

to specify the location of various directories (see footnote below for VMS-specific information†) and various snodb initializations.

One of the fields (labelled "official" or "test") is used to define whether an official SNO database is set up for general use or if a test database is set up for personal use (for testing data before it is input into the official database). Leave the field as "official."

You should also change the pointers to the locations of CERNLIBs, edit in the correct command for doing backups (mainly relevant for non-UNIX platforms), and modify any other fields in the `snodb.config` file to tailor SNODB to your system.
The final step is to set up some environmental variables:

UNIX: set the environmental variable `CDSERV` which tells HEPDB where your server resides (i.e., under the parent directory with the name you have chosen for your server). The default is `/hepdb/cdsno` and it is set by the snodb.perl script as needed. However, other users who wish to run their own personal server (for example, with test database files) will need to edit their `.login` file and define the variable CDSERV to point to their personal server directory. For example, in csh:

```
setenv CDSERV /users/robk/hepdb/cdtest
```

The environmental variable `SNODB_VER` needs to be set to the name of the SNODB version, the variable `SNODB_ROOT` needs to be set to the name of the directory in which the "3_07_01/scripts..." directory structure resides, and the variable `SNO_CODE` needs to be set to the directory containing the latest development release of SNOMAN. For example, in csh:

```
setenv SNODB_ROOT /users/cdsno
setenv SNODB_VER 3_07_01
setenv SNO_CODE /users/snoman/dev
```

VMS: both a global symbol *and* a logical name called `CDSERV` have to be defined as described in the UNIX paragraph above. Logical names SNODB_ROOT, SNODB_VER and SNO_CODE also have to be defined as described above. For example:

$	define	CDSERV dkb$200:[hepdb.cdsno]
$	cdserv	== "dkb$200:[hepdb.cdsno]"
$	define	SNODB_ROOT dkb$200:[cdsno]
$	define	SNODB_VER 3_07_01
$	define	SNO_CODE dkb$200:[snoman]

It is important that the global symbol `CDSERV` and the logical names `SNODB_ROOT` and `SNODB_VER` be accessible to batch processes, so be careful that you don't define them only under a `IF F$MODE() .EQS. "INTERACTIVE" THEN...` -type statement in your `LOGIN.COM` file.

A global symbol pointing to the `EXTRACT.EXE` image needs to be defined. For example:

```
$extract:==$dkb$200:[snoman.tools.exe]extract.exe
```

The name of the batch queue in which the HEPDB database server will run needs to be defined using the logical name `CDSERV$BATCH_QUEUE`. The queue should allow

†Use VMS syntax when specifying pathnames in `snodb.config`, and *escape* all dollar symbols in the pathnames with a backslash. For example,
`''official dbase file dir'',''dkb$200:[cdsno.dbs]'', #directory...`
Also, the fields "read group name," "write group name" and "link libraries" only apply to UNIX systems, and are ignored on VMS systems.

infinite maximum CPU time, and the owner of the server job should have permission to start and stop the queue. A high priority for the queue is not necessary.

Most of the variables defined above are only required by the account from which the database is maintained (i.e. the `cdsno` account). Others who simply want read or write access to database files only need to set up the CDSERV environmental variable on UNIX systems and the CDSERV global symbol on VMS systems as defined above.

The setup script automatically loads in the compiling and linking flags defined for your platform in the SNOMAN file $SNO_TOOLS/get_platform_variables.scr. For this to work you must have edited the contents of that file as needed and defined SNO_TOOLS and SNO_PLATFORM (see the SNOMAN installation procedure).

You are now ready to run snodb.perl. To see a list of the snodb.perl command options, type the command:

UNIX: `~/3_07_01/scripts/snodb.perl -help`

VMS: `perl [cdsno.3_07_01.scripts]snodb.perl -help`

Some tedious typing can be avoided if the following is done:

UNIX: put the `/3_07_01/scripts` directory in your path, then simply type:

```
snodb.perl -help
```

VMS: define a symbol to run the script:

```
snodb_cmd=="''perl' dkb$200:[cdsno.3_07_01.scripts]snodb.perl"
snodb_cmd -help
```

Note that in what follows command options are given one at a time, but that they can be entered simultaneously if you wish, keeping in mind that the order may matter. Also note that, on UNIX systems, if `/usr/bin/perl` does not exist (either as an executable or as a link to one) then you will either have to edit the first line in snodb.perl to point to the location of the executable, or type `perl` before the script name like on VMS systems.

1.3.2. Creating the Directory Structure for SNODB

The first step is to create the directory structure. Type:

```
~/3_07_01/scripts/snodb.perl -dir
```

This will create the required directory structure and set the proper ownerships[‡] for each of them. You should now have 2 directory structures:

1. The official or test database directory (depending on which was chosen in the `snodb.config` file) with permission 740.
2. The server directory. If undefined, it is set by default to be `/hepdb/cdsno`. It should have the following minimum structure:

[‡] In some institutions, where the system administrator does not allow the parent directory to belong to the "cdsno" account, you will need to reset the group assignment (and some permissions) of all the created directories and files.

```
/hepdb
        /cdsno
                /bad     ! dir where the server places bad updates
                /log     ! dir where the server logs are written
                /queue   ! dir where HEPDB clients place new updates
                /save    ! dir where server saves old updates
                /todo    ! dir which server scans for new updates
```

The following top directories will appear if your node is the "site" node. (Addition subdirectories are also made but not listed below.) If you wish to test reading and writing to the database using the facilities described in Section 7, set the variable `$site_node` to your nodename in the file `3_07_01/scripts/snodb_lib/init.pl`.

```
/hepdb
        /cdsno
                /write_to_dbase    ! Top dir for ascii titles files writes
                /read_from_dbase   ! Top dir for ascii titles files reads
```

On VMS systems, the ACL for the `queue` directory should be set to allow write access only for the privileged few locals who supply official data to the SNO database.

1.3.3. Running the Source Files Through EXTRACT

The fortran source code in the SNODB distribution must be run through EXTRACT prior to building the executables on any platform. This is done by issuing the command:

```
~/3_07_01/scripts/snodb.perl -extract
```

This only has to be done once after the code has been unpacked.

1.3.4. Making the Binaries

Next compile and link the various SNODB tools and the server program by issuing the command:

```
~/3_07_01/scripts/snodb.perl -make_tools
```

You should now have the following files under the `~/3_07_01/exes` directory.

```
cdserv              ! The server program from CERN plus bug fixes
sncreate            ! Will create the database files
sndirs              ! Will create the directory structure in the database file
snalia              ! Will create alias definition for the diretories
sdb_input_titles    ! Loads the SNOMAN titles files in the database
sdb_output_titles   ! Extracts titles files from the database
```

There will also be at least one log file in the `exes` directory in which the results of the building process are recorded.

1.3.5. Creating the `hepdb.names` *File*

Next create the `hepdb.names` file via

```
~/3_07_01/scripts/snodb.perl -names
```

This file is created in `$CDSERV/` and it contains the server configuration. It contains such information as the location of the database file(s), the location of the server's directories, the write and read authorizations, the wake-up interval for the server, node locations, etc.

For the typical case where you are following this installation procedure in order to install the database from scratch, you must edit the `hepdb.names` file in order to temporarily fool `cdserv` into thinking that your node is a master server and not a slave server. This way, the updates resulting from the execution of `sndirs` and `snalia` later on in this installation procedure will be processed immediately on your local node rather than being queued up for processing by the remote master node. The edit to `hepdb.names` is simply to change the line

```
:queue./hepdb/cdsno/queue
```

to

```
:queue./hepdb/cdsno/todo
```

You will be instructed to remove this change later in the installation procedure.

1.3.6. Starting the Server

You then need to start the database server before building the database:

```
~/3_07_01/scripts/snodb.perl -server
```

You may verify that the server is running:

UNIX: use the `ps` command.

VMS: type `show queue cdserv$batch_queue`

You are now ready to build the database.

1.4. Building the Database

1.4.1. Building the Database Files and Structure

Now that the server is running and ready to receive updates type the command:

```
~/3_07_01/scripts/snodb.perl -build_official
```

This builds the database files. When run, the `sncreate` program creates the database files listed in the `sncreate.dat` file. The program `sndirs` creates internal structure as requested by `~/3_07_01/dbs_tools_f/sndirs.dat` in each of these files. `snalia` is then run to create HEPDB aliases as requested in `sndirs.dat`.

The script will wait until `sndirs` is finished running before running `snalia`. It does this to allow the server to process all the directory changes before it tries to process all the aliases for those directories. Otherwise, `snalia` would report errors. The script also waits for the updates generated by `snalia` to be processed before proceeding. Finally, the script checks the log files from the `sndirs` and `snalia` operations for errors and reports any it finds. If you wish, you may run these programs independently by typing:
`~/3_07_01/scripts/snodb.perl -sncreate`

~/3_07_01/scripts/snodb.perl -sndirs
~/3_07_01/scripts/snodb.perl -snalia
(but you are definitely better off running -build_official). To be sure that the server is finished creating the directories, check that the `$CDSERV/todo` directory is empty before running `snalia`.

Note that `sncreate` can be run without the server running.

1.4.2. Loading the Database with the SNOMAN Titles Files

Finally you can load the database with the SNOMAN titles files. First, you must start the server running via

```
~/3_07_01/scripts/snodb.perl -server
```

Then, to start the (long) process by which the database is loaded with the initial database files–mainly SNOMAN-related titles files containing geometry and various physics constants–type

```
~/3_07_01/scripts/snodb.perl -load_tz
```

This loads the SNOMAN titles files in the database file `sm.dbs` in the official database directory. The loading is performed by running the program `sdb_input_titles`.

After the loading is finished, the server is stopped, the copies of the journal files used to implement the loading of the titles files in the `hepdb/cdsno/save` directory are backed up. The journal files in the resulting backup can be used to re-input the data into the database files again (just by copying them into the `hepdb/cdsno/todo` directory), and are stored in case they are needed to repair the database after it is corrupted, or the disk fails, or some other similar catastrophe.

If you edited the `hepdb.names` file in Section 1.3.5 you must undo the edit now. This entails changing the line

```
:queue./hepdb/cdsno/todo
```
to
```
:queue./hepdb/cdsno/queue
```

If the server is already running, you must stop and restart it in order to make this change have effect. To stop the server, issue the command:

```
~/3_07_01/scripts/snodb.perl -stop_server
```

and then restart it by issuing the command:

```
~/3_07_01/scripts/snodb.perl -server
```

1.5. Setting up for a Networked Database

The following changes must be made by your system manager in order to enable SNODB to update the database on your node, and accept database updates from your node, via ftp. (Note that SNODB will not be running in networked mode in this release but that it will be in the near future.)

UNIX:

1. Add the following line to /etc/services:
   `zserv 346/tcp # server for zftp`
2. Add the following line to /etc/inetd.conf:
   `zserv stream tcp nowait root <path to zserv>/zserv zserv`
3. Reset inetd with one of the following commands:

```
refresh -s inetd                 ; AIX
inetd -l                         ; HP/UX
/sbin/init.d/inetd stop--start   ; Alpha/OSF
kill -HUP <inetd PID>            ; others
```

4. On AIX systems, CERN's CSPACK manual recommends that the following line be added to the file `/etc/environment`, setting the shell variable `xrf_messages` to `no` to prevent FORTRAN run-time messages from being printed:
   `xrf_messages=no`
   These messages disturb the protocol used between ZSERV and PAWSERV servers and the ZFTP or PAW clients.
5. Those of you who use yellow pages should see Chapter 13 of CERN's CSPACK manual for a couple of more details on setting up these services.

VMS: the steps outlined below apply to those machines running MULTINET TCP/IP. See Chapter 13 of CERN's CSPACK Manual for information on other systems.

1. Add the following line to MULTINET:HOSTS.LOCAL
   `SERVICE : TCP : 346 : ZSERV`
2. Compile and activate this change by typing:

```
MULTINET HOST_TABLE COMPILE
@MULTINET:INSTALL_DATABASES
@MULTINET:START_SERVER
```

3. Define the server to MULTINET:
   `MULTINET CONFIGURE /SERVERS`
   A dialog similar to the following should follow:

```
SERVER-CONFIG>add zserv
Protocol: [TCP] tcp
TCP port number: 346
Program to run: CERN:[PRO.EXE]ZSERV.EXE
SERVER-CONFIG>RESTART
```

1.6. Running SNOMAN with the "Static Database"

Here are the relevent excerpts from the SNOMAN v3.0086 user manual.

> As from 2.09, SNOMAN has an interface to SNO Database which is implimented using the CERN package HEPDB. SNOMAN 3.00 is incompatible with the Database available when 2.09 was released. In 2.09, the database used lower case aliases whereas in 3.00 they are upper case. Quite apart from this certain

titles banks are not backward compatible and so conflicts will arise if you try to run SNOMAN 3.00 with a 2.09 titles file set or database and vice versa.

You must rebuild the database before using it with SNOMAN 3.00!

To allow people time to get an up to date CERNlib and set up a database, SNOMAN 3.00 does not require a database to run. Indeed, by default, it will not even initialise the database software unless the symbolic command:

```
$enable_database
```

is issued. Even that is insufficient to get it to read from the database as titles banks coming from titles files take priority. As SNOMAN comes with a complete set of title files, there is no need to search the database for them. To force loading from the database it is necessary to remove TITLES commands from the .cmd files and principally the file `read_universal_titles.cmd`. To test out the database you could try commenting out lines in this file (assuming that the files have been loaded into the database!). However, do NOT comment out the following files:

`job_control.dat` This contains the TMTT titles bank which controls enabling of the database. As such, it could hardly be useful if the database had to be accessed to get it! It also contains TSDB which controls database initialisation which is there for the same reason.

`ntuple.dat` SNOMAN does not demand NTPR (ntuple request) banks (see section 5.1; instead it just uses those that the user has provided via titles files.

A simple way to test the database is to use the command file:

```
@run_snodb.cmd
```

You will see that this command file also contains the line:

```
$sdb_enable_verification
```

which is recommended as it does checksum validation of the data as it reads it from the database. The above switch changes the SDB titles bank TSDB. This bank contains other switches but they should, for the most part, be left at their default settings. The only possible exception is e.g.:

```
$sdb_max_ins_date 19970912 123200
```

which will mask off any updates placed in the database after that date time and would be used when reproducing an old analysis, to ensure that only those banks that existed at the time of the original analysis are used.

When you run with the database you may see a warning of the type:

```
RZOPEN. Error - input file ... sn.dbs does not exist
```

Pay no attention to it; it refers to an obsolete file and will be fixed with the next release of SNODB.

In 2.09 you were warned not to load the `pmt_positions_minisno_*.dat` files into the database as the GEDP banks they contain would overwrite the SNO ones and you would end up with rather few PMTs! This is no longer a problem; miniSNO data from Mini-SNO has a different data type to SNO and the access to the database is data type sensitive. So you should load the 90 PMT MINISNO file (`pmt_positions_minisno_90.dat`) into the database. If you want the 128 PMT set, input them via a titles file, which will override the default 90 PMT version.

Once the SNO database has been established throughtout the collaboration, SNOMAN will become dependent on it and titles files will, for the most part, disappear. However titles file support will remain part of SNOMAN because, as its banks take priority over the database, it provides a mechanism for overriding existing titles and adding new ones without having to update the database.

In short you can run SNOMAN with the database by moving all `.dat` files from the `prod/` area except

`ntuple.dat`, `job_control.dat`, `pegs4_10.dat` and `symbol_table.dat`.

Here is a (simple) sample command file which uses the database:

```
!file member=RUN_SNODB library=SNOMAN language=cmd
!file DATE=30:Jun:1995
echo
*     Standard Command File: run_snodb.cmd                  22 May 1996
noecho
*     Contact:  G. Jonkmans,  CRL.

*     Perform standard run.

$enable_database

titles ntuple.dat
titles job_control.dat
*     Terminate command processing
end
!endfile member=RUN_SNODB
```

B. Sample `hepdb.names` Files

Here are a few sample `hepdb.names` files. The file below is similar to one which is used on the master node. To save space, only a few database files and only a few nodes are shown. Please see the HEPDB manual[2] for an explanation of the keywords used in these files.

```
:nick.config
   :list.as av ch
   :log./hepdb/cdsno/log
```

```
   :queue./hepdb/cdsno/todo
   :todo./hepdb/cdsno/todo
   :save./hepdb/cdsno/save
   :bad./hepdb/cdsno/bad
   :loglevel.3
   :wakeup.30
   :servers.laurentian guelph lanl

:nick.as
   :file./usr/local/hepdb/official/dbs/assay.dbs
   :desc.assay info
   :servers.
   :read.*
   :write.cokada edh cdsno

:nick.av
   :file./usr/local/hepdb/official/dbs/av.dbs
   :desc.acrylic vessel
   :servers.
   :read.*
   :write.cdsno

:nick.ch
   :file./usr/local/hepdb/official/dbs/cal_history.dbs
   :desc.calibration sources history
   :servers.
   :read.*
   :write.hallin ford mcdonald cdsno

:nick.guelph
   :userid.cdsno
   :node.eta.physics.uoguelph.ca
   :localq./hepdb/cdsno/to_guelph
   :protocol.tcpip
   :queue./hepdb/cdsno/todo
   :poll.yes

:nick.lanl
   :userid.cdsno
   :node.snohp.lanl.gov
   :localq./hepdb/cdsno/to_lanl
   :protocol.tcpip
   :queue./home/cdsno/hepdb/cdsno/todo
   :poll.yes

:nick.laurentian
```

```
   :userid.cdsno
   :node.neutrino.phys.laurentian.ca
   :localq./hepdb/cdsno/to_laurentian
   :protocol.tcpip
   :queue./hepdb/cdsno/todo
   :poll.yes
```

This file is similar to one used on a slave node. Again, only a few database files are shown.

```
:nick.config
   :list.as av ch
   :log./hepdb/cdsno/log
   :queue./hepdb/cdsno/queue
   :todo./hepdb/cdsno/todo
   :save./hepdb/cdsno/save
   :bad./hepdb/cdsno/bad
   :loglevel.3
   :wakeup.30
   :servers.

:nick.as
   :file./usr/local/hepdb/3_06_02/dbs/assay.dbs
   :desc.assay info
   :servers.
   :read.*
   :write.cdsno

:nick.av
   :file./usr/local/hepdb/3_06_02/dbs/av.dbs
   :desc.acrylic vessel
   :servers.
   :read.*
   :write.cdsno

:nick.ch
   :file./usr/local/hepdb/3_06_02/dbs/cal_history.dbs
   :desc.calibration sources history
   :servers.
   :read.*
   :write.cdsno
```

C. The Initial sndirs.dat File

```
# This file is read by 'sndirs' when creating directories internal to
# the HEPDB datbase files, and by 'snalia' when assigning aliases to
# those directories.
# Each line has up to three fields, separated by spaces or tabs:
# 1) full pathname of the directory in the RZ structure of the
#    database file.  The '//CDxx' tells it to create the
#    directory in the 'xx.dbs' database file.
# 2) the alias for the directory.  If the directory contains
#    copies of a ZEBRA bank, then the 4 character bank name
#    is to be used as the alias.  The alias is case-insensitive.
# 3) the options to be used when creating the directory.
#    These are: ' ' default (non-partitioned)
#       'P' partitioned (to 1000 objects each)
#       'C' objects are compressed by default.
#       'F' create a second directory inside the first to
#           hold the titles format banks, and alias it
#           by adding 'FMT' to the first alias.
#
# The directory pathname is mandatory, the alias is mandatory for
# directories that will be accessed by the name of the banks that
# they contain, and the option is the default if left out.
#
# *** The maximum number of characters allowed in the full pathname
# to the directory is 50 characters, while the total number of
# characters in each line below must be less than 100. ***
#
# (in this file, the '#' symbol only acts to comment out a line if placed
# in the first column.)
#
#  MODS:
#     1. added //CDSM/PT70/HIST hist F.  DFC 6Feb97.
#     2. added//CDSM/MCDSDATA/MCEC mcec F. From Dave via Nick.  DFC 12Feb97.
#     3. removed extra //CDSM/GEOM/GEDS geds F. RK 8Mar97.
#     4. added //CDSM/PMRS/*VAR *var F.  RK 8Mar97
#     5. added //CDCO/TOCA toca F. RK 8Mar97.
#     6. added //CDCE/TECA teca F. RK 8Mar97.
#     7. completely revamped for close-to-final DB structure.  DFC 25Jul97.
#     8. added banks: walk wqwd gain gqwd to //CDPM.  DFC 16Apr98.
#
#
#=====================================================================
#
########
# calibration: electronics
```

```
#
#    -charge slopes: elec_charge_slopes.dbs
#
//CDEC
#
# This bank is (I think) going to be superceded by the ones following.
#
//CDEC/cst0 cst0 f
#
//CDEC/q1ch q1ch f
//CDEC/q1si q1si f
//CDEC/q1sp q1sp f
//CDEC/q1sl q1sl f
//CDEC/q1sq q1sq f
//CDEC/q1qc q1qc f
//CDEC/q2ch q2ch f
//CDEC/q2si q2si f
//CDEC/q2sp q2sp f
//CDEC/q2sl q2sl f
//CDEC/q2sq q2sq f
//CDEC/q2qc q2qc f
//CDEC/q3ch q3ch f
//CDEC/q3si q3si f
//CDEC/q3sp q3sp f
//CDEC/q3sl q3sl f
//CDEC/q3sq q3sq f
//CDEC/q3qc q3qc f
#
#    -time slopes: elec_time_slopes.dbs
#
//CDET
//CDET/tmdm tmdm f
//CDET/tmch tmch f
//CDET/tmsi tmsi f
//CDET/tmsp tmsp f
//CDET/tmsl tmsl f
//CDET/tmsq tmsq f
//CDET/tmqf tmqf f
//CDET/tmqc tmqc f
//CDET/tmcc tmcc f
#
#    -pedestals: elec_pedestals.dbs
#
//CDEP
//CDEP/q1av q1av f
//CDEP/q2av q2av f
```

```
//CDEP/q3av q3av f
//CDEP/tmav tmav f
//CDEP/q1fl q1fl f
//CDEP/q2fl q2fl f
//CDEP/q3fl q3fl f
//CDEP/tmfl tmfl f
//CDEP/q1ne q1ne f
//CDEP/q2ne q2ne f
//CDEP/q3ne q3ne f
//CDEP/tmne tmne f
//CDEP/q1rm q1rm f
//CDEP/q2rm q2rm f
//CDEP/q3rm q3rm f
//CDEP/tmrm tmrm F
#
########
# calibration: sources
#
#    -history: cal_history.dbs
#
//CDCH
//CDCH/dates/trial datr F
#
#    -sources: cal_sources.dbs
#
//CDCS
#
//CDCS/optical/laser/N2_dye olnd F
//CDCS/optical/laser/short  olsh F
//CDCS/optical/laser/double oldb F
//CDCS/optical/flash/N2     ofn2 F
//CDCS/optical/flash/Xe     ofxe F
//CDCS/optical/sono         oson F
//CDCS/optical/led          oled F
//CDCS/optical/scint        osci F
//CDCS/rotating             rota F
//CDCS/gas/N16              gn16 F
//CDCS/gas/N17              gn17 F
//CDCS/gas/Li8              gli8 F
//CDCS/accel/Li7            ali7 F
//CDCS/accel/pt             apt  F
//CDCS/prop/Th              pth  F
//CDCS/prop/Pb212           ppb  F
//CDCS/neutron/Cf/hi        ncfh F
//CDCS/neutron/Cf/lo        ncfl F
//CDCS/NaI                  nai  F
```

```
//CDCS/hot_rock/U_Th        hrut F
#
#    -rundata: cal_rundata.dbs
#
//CDCR
//CDCR/optical/laser/N2_dye/stat  olns F
//CDCR/optical/laser/N2_dye/nhit  olnn F
//CDCR/optical/laser/N2_dye/Q     olnq F
//CDCR/optical/laser/short/stat   olss F
//CDCR/optical/laser/short/nhit   olsn F
//CDCR/optical/laser/double/stat  olds F
//CDCR/optical/laser/double/nhit  oldn F
//CDCR/optical/flash/N2/stat      ofns F
//CDCR/optical/flash/N2/nhit      ofnn F
//CDCR/optical/flash/Xe/stat      ofxs F
//CDCR/optical/flash/Xe/nhit      ofxn F
//CDCR/optical/sono/stat          osos F
//CDCR/optical/sono/nhit          osnh F
//CDCR/optical/sono/Q             osoq F
//CDCR/optical/led/stat           oles F
//CDCR/optical/led/nhit           olen F
//CDCR/optical/scint/stat         oscs F
//CDCR/optical/scint/nhit         oscn F
//CDCR/rotating                   rota F
//CDCR/gas/N16/stat               gn6s F
//CDCR/gas/N16/nhit               gn6n F
//CDCR/gas/N16/energy             gn63 F
//CDCR/gas/N17/stat               gn7s F
//CDCR/gas/N17/nhit               gn7n F
//CDCR/gas/N17/energy             gn7e F
//CDCR/gas/Li8/stat               gl8s F
//CDCR/gas/Li8/nhit               gl8n F
//CDCR/gas/Li8/energy             gl8e F
//CDCR/accel/Li7/stat             al7s F
//CDCR/accel/Li7/nhit             al7n F
//CDCR/accel/Li7/energy           al7e F
//CDCR/accel/pt/stat              apts F
//CDCR/accel/pt/nhit              aptn F
//CDCR/accel/pt/energy            apte F
//CDCR/prop/Th/stat               pths F
//CDCR/prop/Th/nhit               pthn F
//CDCR/prop/Th/energy             pthe F
//CDCR/prop/Pb212/stat            ppbs F
//CDCR/prop/Pb212/nhit            ppbn F
//CDCR/prop/Pb212/energy          ppbe F
//CDCR/neutron/Cf/hi/stat         nchs F
```

```
//CDCR/neutron/Cf/hi/nhit         nchn F
//CDCR/neutron/Cf/hi/energy       nche F
//CDCR/neutron/Cf/lo/stat         ncls F
//CDCR/neutron/Cf/lo/nhit         ncln F
//CDCR/neutron/Cf/lo/energy       ncle F
//CDCR/neutron/Cf/r_eff           ncre F
//CDCR/NaI/stat                   nais F
//CDCR/NaI/nhit                   nain F
//CDCR/NaI/energy                 naie F
//CDCR/hot_rock/U_Th/stat         huts F
//CDCR/hot_rock/U_Th/nhit         hutn F
//CDCR/hot_rock/U_Th/energy       hute F
#
#    -parameters: cal_parameters.dbs
#
//CDCP
//CDCP/optical/d2o/attenuation            od2a F
//CDCP/optical/d2o/scattering             od2s F
//CDCP/optical/d2o/ext                    od2e F
//CDCP/optical/h2o/attenuation            oh2a F
//CDCP/optical/h2o/scattering             oh2s F
//CDCP/optical/h2o/ext                    oh2e F
//CDCP/optical/acrylic/attenuation/panels oaap F
//CDCP/pmts/rel_qe                        pmrq F
//CDCP/pmts/rel_eff                       pmre F
//CDCP/detector/gain                      detg F
//CDCP/energy/gain                        eneg F
#
########
# NCDs
#   convention:
#      a: array, s: string, c: counter,
#      t: tube, p: preamp, d: daq
#
#   -Physical characteristics: ncd_phys_char.dbs
#
//CDNP
//CDNP/apch ncap F
//CDNP/spch ncsp F
//CDNP/cpch nccp F
//CDNP/tpch nctp F
//CDNP/ppch ncpp F
//CDNP/dpch ncdp F
#
#   -Calibration: ncd_calibration.dbs
#
```

```
//CDNC
//CDNC/acal ncac F
//CDNC/scal ncsc F
//CDNC/dcal ncdc F
//CDNC/tcal nctc F
//CDNC/pcal ncpc F
#
#   -Parameters: ncd_parameters.dbs
#
//CDNP
//CDNP/apar ncap F
//CDNP/spar ncsp F
//CDNP/dpar ncdp F
//CDNP/tpar nctp F
//CDNP/ppar ncpp F
#
########
# Acrylic Vessel
#
#    -acrylic vessel: av.dbs
#
//CDAV
//CDAV/avap avap F
//CDAV/avab avab F
#
########
# SNOMAN
#
#    -snolib: snoman_snolib.dbs
#
//CDSS
#
# check fltr
#
//CDSS/fltr fltr F
#
# verify location of "ocal" and "tpca"
#
//CDSS/ocal ocal F
//CDSS/tpca tpca F
//CDSS/dsm  dsm  F
//CDSS/egs4 egs4 F
//CDSS/hist hist F
//CDSS/job  job  F
//CDSS/jobp jobp F
//CDSS/mcdq mcdq F
```

```
//CDSS/mcec mcec F
//CDSS/mcma mcma F
//CDSS/mcpi mcpi F
//CDSS/mcpl mcpl F
//CDSS/meda meda F
//CDSS/mmsw mmsw F
//CDSS/ntpc ntpc F
//CDSS/ntpr ntpr F
//CDSS/pdbg pdbg F
//CDSS/pmte pmte F
//CDSS/seed seed F
//CDSS/tcal tcal F
//CDSS/tcln tcln F
//CDSS/teca teca F
//CDSS/teio teio F
//CDSS/tft1 tft1 F
//CDSS/tfte tfte F
//CDSS/tftg tftg F
//CDSS/tftq tftq F
//CDSS/tftt tftt F
//CDSS/tgeo tgeo F
//CDSS/gedp gedp F
//CDSS/geds geds F
//CDSS/tmtt tmtt F
//CDSS/toca toca F
//CDSS/tpck tpck F
//CDSS/tpru tpru F
//CDSS/tsdb tsdb F
//CDSS/tusr tusr F
//CDSS/tupk tupk F
//CDSS/atpa atpa F
//CDSS/atpr atpr F
//CDSS/trdv trdv F
#
#    -detector description: snoman_detector.dbs
#
//CDSD
//CDSD/b8ps b8sp F
//CDSD/bapn banp F
//CDSD/baps basp F
//CDSD/bbpn bbnp F
//CDSD/bbps bbsp F
//CDSD/bcpn bcnp F
//CDSD/bcps bcsp F
//CDSD/bdpn bdnp F
//CDSD/bdps bdsp F
```

```
//CDSD/bupn bunp F
//CDSD/bups busp F
//CDSD/ccbx ccxb F
//CDSD/ccsx ccxs F
//CDSD/cctx ccxt F
//CDSD/mcsd mcds F
//CDSD/ntnr ntrn F
//CDSD/ncbx ncxb F
//CDSD/ncsx ncxs F
//CDSD/nctx ncxt F
//CDSD/mccv mccv F
//CDSD/cbbc cbbc F
#
#    -pmt performance: snoman_pmt.dbs
#
//CDSP
//CDSP/omeg omeg F
//CDSP/pcat pcat F
//CDSP/pmtr pmtr F
//CDSP/pmtt pmtt F
//CDSP/qtca qtca F
//CDSP/spel spel F
//CDSP/twob twob F
#
#    -other stuff (laserball angular dist., CCC->tube mapping, wavelens used for calib.)
#       snoman_other.dbs
#
//CDSO
//CDSO/lasr lasr F
//CDSO/ccct ccct F
//CDSO/wavl wavl F
//CDSO/avar avar F
//CDSO/qvar qvar F
//CDSO/rvar rvar F
//CDSO/fltr fltr F
#
########
# daq
#
#   -board initialization: daq_board_init.dbs
#
//CDDB
#
#   -configuration: daq_configuration.dbs
#
//CDDC/cron cron F
```

```
#
#    -run headers: daq_run_headers.dbs
#
//CDDR
########
# pmt
#
#   pmt.dbs
#
//CDPM
//CDPM/walk walk F
//CDPM/wqwd wqwd F
//CDPM/gain gain F
//CDPM/gqwd gqwd F
#
########
# psup
#
#   -psup.dbs
#
//CDUP
#
########
# Assay
#
#    -All assay information: assay.dbs
#
//CDAS
#
#    -actual assay results
#
//CDAS/AH2O ah2o F
//CDAS/AD2O ad2o F
#
#    -counters' raw spectra, fitting/error analysis results
#
//CDAS/ACNT acnt F
#
########
# water
#
#   -water.dbs
#
//CDWT
#
########
```

```
# cma
#
#   -cma.dbs
#
//CDCM
//CDCM/CMAL/AVLC avlc F
//CDCM/CMAL/CHST chst F
//CDCM/CMAL/CHRT chrt F
//CDCM/CMAL/CHBV chbv F
//CDCM/CMAL/CHBA chba F
//CDCM/CMAL/CWET cwet F
//CDCM/CMAL/CWRT cwrt F
//CDCM/CMAL/CCET ccet F
//CDCM/CMAL/CCLT cclt F
//CDCM/CMAL/CAMP camp F
//CDCM/CMAL/CSPS csps F
//CDCM/CMAL/CDPS cdps F
//CDCM/CMAL/COPS cops F
//CDCM/CMAL/COP2 cop2 F
//CDCM/CMAL/CAMP camp F
//CDCM/CMAL/C24V c24v F
//CDCM/CMAL/C12V c12v F
//CDCM/CMAL/CCBT ccbt F
//CDCM/CMAL/CDRT cdrt F
//CDCM/CMAL/CBLD cbld F
//CDCM/CMAL/CBXY cbxy F
//CDCM/CMAL/COO1 coo1 F
//CDCM/CMAL/NO2O no2o F
//CDCM/CMAL/COO2 coo2 F
//CDCM/CMAL/COIN coin F
//CDCM/CMAL/O2DK o2dk F
//CDCM/CMAL/O2UT o2ut F
//CDCM/CMAL/O2RF o2rf F
//CDCM/CMAL/O2CW o2cw F
//CDCM/CMAL/TJCT tjct F
//CDCM/CMAL/TCTR tctr F
//CDCM/CMAL/TUTL tutl F
//CDCM/CMAL/TDEK tdek F
//CDCM/CMAL/FZTP fztp F
//CDCM/CMAL/PABS pabs F
//CDCM/CMAL/PDLT pdlt F
//CDCM/CMAL/HMUT hmut F
//CDCM/CMAL/HMDK hmdk F
//CDCM/CMAL/MC06 mc06 F
//CDCM/CMAL/MC13 mc13 F
//CDCM/CMAL/MC07 mc07 F
```

```
//CDCM/CMAL/MC14 mc14 F
//CDCM/CMAL/MC01 mc01 F
//CDCM/CMAL/MC08 mc08 F
//CDCM/CMAL/MC02 mc02 F
//CDCM/CMAL/MC09 mc09 F
//CDCM/CMAL/MC03 mc03 F
//CDCM/CMAL/MC10 mc10 F
//CDCM/CMAL/MC04 mc04 F
//CDCM/CMAL/MC11 mc11 F
//CDCM/CMAL/MC05 mc05 F
//CDCM/CMAL/MC12 mc12 F
//CDCM/CMAL/MTRV mtrv F
//CDCM/CMAL/MTRA mtra F
//CDCM/CMAL/LOAD load F
//CDCM/CMAL/UPPW uppw F
//CDCM/CMAL/UPBV upbv F
//CDCM/CMAL/UPBA upba F
//CDCM/CMAL/UPSA upsa F
//CDCM/CMAL/UPST upst F
#
########
# tapes
#
#   -tapes.dbs
#
//CDTP
#
########
# test
#
#   -test.dbs
#
//CDTT
//CDTT/test test
```

D. Titles File Format

The following documentation can be found in full form on the web at `http://wwwcn.cern.ch/asdoc/zebra_html3/zebramain.html` (the Titles section of the CERN Zebra online manual).

```
Outline

The input to TZINIT is presented on a formatted text file of 80-column
card-images, giving the titles one after the other, as shown by the
example on the next page.
```

Each title starts with the 'title header line' *DO ... which selects mode and options for the reading of its associated ''title data''. These have to come immediately and completely after the title header line.

Normal mode is ''free-field format'' where the data are read line-by-line and the text is interpreted. The mode of each data word is taken from the way it is written: integer data have neither the decimal point nor an exponent, floating data have either or both, Hollerith data start with : or ' or ", octal data are pre-fixed with #O.

With ''Fortran format'' the data are read by a single Fortran READ statement with the FORMAT taken from the -F option on the title header line. This is useful for computer generated titles.

''Control lines'' to TZINIT may be given interspersed with the titles. Global options are selected with control lines; other examples are the comment line with *-- in column 1, and the input terminator *FINISH.

Control lines to TZINIT, including title-header lines starting with *DO, are handled case-insensitive, ie. they are converted to upper case before processing.

```
*-------            TITLE VERSION A3
*PRINT
*.-------1---------2---------3---------4---------5---------6---------7
*DO  RTBC  -iF                  #. 810127     CERN  ROLL 103
13.       1100.
           .99999  +.00042  +.00302
          -.00042   .99993  -.0023
          -.00302  +.0023    .99993     0.     +.015     1.963

*DO  CAM1  -e -n27 -c11 -iF      #. 810121 17.27  ROLL 103
 1 CAM       3.  12.    9.067    11.8516   75.6561     25.    -75.6561
 1 MED       2.        1.51      1.5       1.458     2.382     1.57
             4.06      1.0884
 1 DIST       -.00169     -.00004      .015        .0011      -.0001
               .00394     -.00245      .00378      .051        .0023
               .033        .09

*DO  FID1  -c11 -iF              #. 810121 17.27  ROLL 103  ERASME
            12.
 1 4   1     1.  -17.2475  -18.6369
 1 4   2     2.  -17.2595   -4.2708
 1 4   3     3.    -.7761   -4.3177
```

```
 1 1   1    11.  -13.8081  -15.9118
 1 1   2    12.  -13.7343   -7.5253
 1 1   3    13.   -4.8669   -7.5898
 1 1   4    14.   -4.8931  -15.9473
 1 2   1    21.  -13.0353  -15.6000
 1 2   2    22.  -12.8979   -7.3464
 1 2   4    24.   -3.7895  -15.6767
 1 3   3    33.   -2.6674  -17.2822
 1 3   4    34.   -3.1636  -17.8522

*DO  CAM2  -e -n27 -c11 -iF      #. 810121 17.27  ROLL 103
 2 CAM       3.  12.    9.1488  -10.3208   75.673      25.    -75.673
 2 MED       2.        1.51      1.5       1.458     2.382     1.57
             4.06      1.0884
 2 DIST       -.0017       .00005      .015       -.0011      -.0001
               .00043     -.00443     -.00278      .0065      -.0024
               .033        .05

*DO  FID2  -c11/40 -iF           #. 810121 17.27  ROLL 103  ERASME
            13.
 2 4   1     1.  -17.4350    3.9107     examples of other representations
 2 4   2     2.  -17.4786   18.2503     integer    15  -24  0
 2 4   3     3.    -.8612   18.2306     floating   -.123E-7  1E-12
 2 1   1    11.  -13.8891    7.4810     octal      #0731244600
 2 1   2    12.  -13.8296   15.8810     hex        #xffff
 2 1   3    13.   -4.9569   15.7810     Hollerith  :PISA :BARI
 2 1   4    14.   -4.9724    7.4133                :K*(1430)
 2 2   1    21.  -13.1122    7.3163                'text with blanks'
 2 2   3    23.   -3.7338   15.3468                "text with ' and blank"
 2 2   4    24.   -3.8710    7.1994
 2 3   1    31.  -15.4499    4.5935
 2 3   2    32.  -15.8947    5.0865
 2 3   4    34.   -3.2437    4.4470
*.-------1---------2---------3---------4---------5---------6---------7
```

Title header lines

A title-header line signals the start of data for a new title bank; it specifies the ID of the title bank, the way the data are to be read, and further options. Its format is:

```
   *DO  <idh> [idn]  [-<opt><option text>]   [#. comment ]
```

*DO in cols 1/4 marks the header line;

<idh>
: is the 4 character Hollerith bank ID;

<idn>
: is the numeric bank ID, integer, if any;

<opt>
: is a single option letter,

<otx>
: is the option text, if needed.

The following options are available:

-F(format)
: read the data with a single formatted Fortran READ statement, using the FORMAT given, which must not contain blanks, the brackets must be present.

-Itext
: set the I/O characteristic for the bank, with text:
  B, I, F or D if the bank is all bits, integer, floating, or double precision;
  (string) where string is the I/O characteristic, (see routine MZFORM);
  the default is: type undefined

-Nn expect n data words, for -F this must be exact; for free-field format this may be omitted if less than 2000 words are to be read and if the size is not to be changed by TZUSER. A bank will initially be lifted to accomodate <n> data words to be read and will be shortened when input processing is complete.

-Un call TZUSER for this bank, passing n as NPARA, zero if absent.

For free-field format only:

-Ca/e
: use columns a/e only on each line, defaults: 1 for a, 80 for e
  Ex.: -C11/72 use only the text on columns 11 to 72 on each line, except: '*' in col. 1 is forbidden

-En expect exactly <n> words (as given by -Nn). If less or more data words are found, TZINIT will print a message and go to ZFATAL on end of input.

-Sn true size of the bank is <n> words (as given by -Nn). The bank is not reduced to the number of data words actually read.

-AnCW
: initialize Hollerith handling as though the control-item #A[n][C][W] had been read

Fortran formatted input

With Fortran formatted input all the data for the complete title bank are read with a single READ statement using the format given on the header line. For this to be possible the exact number of words to be read must be specified with the -N option.

Example

```
*DO  FIDU 1 -IF -N37 -F(10X,F10.0/(10X,3F10.0))
           12.
 1 4   1    1.     -17.2475  -18.6369
 1 4   2    2.     -17.2595   -4.2708
 1 4   3    3.       -.7761   -4.3177
 1 1   1   11.     -13.8081  -15.9118
 1 1   2   12.     -13.7343   -7.5253
 1 1   3   13.      -4.8669   -7.5898
 1 1   4   14.      -4.8931  -15.9473
 1 2   1   21.     -13.0353  -15.6000
 1 2   2   22.     -12.8979   -7.3464
 1 2   4   24.      -3.7895  -15.6767
 1 3   3   33.      -2.6674  -17.2822
 1 3   4   34.      -3.1636  -17.8522
*.-------1---------2---------3---------4---------5
```

The option -IF specifying the I/O characteristic is important in this case. It causes the execution of a Fortran READ statement with an I/O list of type REAL; on some machines Fortran is not able to read acting solely under FORMAT control, ignoring the type of the I/O list.

References

1. Till, *Teach Yourself PERL in 21 Days*, SAMS Publishing, 1996. Excellent instructional text and reference manual.

   Schwartz, *Learning Perl*, O'Reilly & Associates (http://www.ora.com/), 1994. A good starting point.

   Wall and Schwartz, *Programming Perl*, O'Reilly & Associates, 1992. A good reference, but doesn't cover PERL 5.

   Wall, Christiansen and Schwartz, *Programming Perl* (2nd Edition), O'Reilly & Associates, 1996. A good reference, covers PERL 5.

2. CERN Program Library Long Writeup Q180, HEPDB Database Management Package Reference Manual v1.19. This manual may be obtained via anonymous ftp at `upenn5.hep.upenn.edu` in the `pub/cowen/snodb` directory as `hepdb_1.19.ps.gz` or at `asisftp.cern.ch` in the `cernlib/doc/ps.dir` directory as `hepdb.ps.Z`.