CFM
Calibration File Manager

Morgan Burke   Ver. 2.2 1993 Oct 1

1. Introduction

What is CFM?

CFM, short for Calibration File Manager, is a relational database developed by Brookhaven Experiment 787 to keep track of large numbers of data files containing calibration and related information for a high-energy physics experiment (although it can be adapted to a variety of other applications). CFM associates sets of data files with experimental run numbers, and allows users to add, delete, and modify filenames, runs, and calibration data types.

CFM can be used in an interactive command language mode, or it can be incorporated into an application that queries the databases for information (*ie.* filenames).

Where is CFM?

Before you can use CFM, you will have to type the following:

```
$ SETUP CFM [version]
```

"Version" is the version number (*eg.* 2.2) of CFM that you wish to use. If not supplied, the current production version is used by default.

The CFM library and image are located in the directory referred to by the environment variable `CFM$MAIN`. To run the image, type the command:

```
$ CFM
```

This is a symbol that points to the CFM image in `CFM$MAIN`.

If you want to link to routines in the CFM library, include the following library specification in your link command:

```
CFM$MAIN:CFM/LIB
```

2. CFM Structure

CFM assumes that an experiment is divided into sequentially numbered *runs.* Each run is conducted under a reasonably constant set of conditions that correspond to a collection of data files used for calibrating and interpreting the data for that run. Often, numerous runs will be conducted under similar conditions, and they will all have the same calibration files. The groups of files associated with these runs are collectively called calibration data sets, or *sets* for short.

The calibration data for each set can consist of various files that address different aspects of the experiment. For example, one file may provide a mapping of physical detector locations to channel numbers, while another file may give pedestal and scaling values to be applied to raw data from some instrument(s). Each aspect of the experiment that can have an associated calibration file is called a calibration data *type.* Each type is referred to by a *component* and and an *attribute,* both being short character identifiers. Components are the physical parts of the experiment or detector requiring calibration, while attributes are the specific kinds of calibrations which need to be done for the various components. For example, one component may be an experiment's drift chamber ("`DC`"), and it may require several different calibrations, such as a channel map, a set of voltage pedestals, and scaling values. The component for all of these calibrations would be "`DC`", while the attributes might be "`MAP`", "`PED`", and "`SCA`". The three calibration types would be a combination of these, *ie.* `DC_MAP`, `DC_PED`, and `DC_SCA`. CFM allows up to 5 characters to identify a component, and up to 3 to identify its attributes.

CFM assumes that all calibration data is stored in files with the name `XXXXX_YYY.ZZZZZ`, where `XXXXX_YYY` is the calibration type that the file is used for, and `ZZZZZ` is some arbitrary integer number (from 1 to 32767) called an *indicator,* which used to uniquely identify the file. For example, one such file name might be `DC_PED.00012`. (Filenames are converted to lower case in Unix.)

CFM maintains three databases to keep track of the runs, sets, and types, respectively. The databases are called `CRF.DAT`, `CIF.DAT`, and `CTF.DAT`, and are stored in the directory referred to by the logical name `CFM$DB`. `CRF.DAT` gives the set that each run belongs to. `CIF.DAT` records the identifier numbers of the files for each calibration type used by each set. `CTF.DAT` contains all the known calibration data types.

Note that CFM does not manage the calibration data files themselves. It only keeps track of file names, and which ones are the proper ones to use for a given experimental run. CFM assumes that all calibration files are stored in a directory referred to by the name `CAL$DB`. It is the duty of the experimenters to ensure that the calibration files actually contain the proper information, that all calibration files that CFM knows about are available for use by users and analysis applications, and that the applications know how to properly interpret the data in the files.

3. Running CFM

When CFM starts up, it conducts checks to see if anyone else is using the databases at the time. If the databases are busy, then the user is permitted to open and read them, but not to make any permanent changes. If the user is the only one using CFM at the time, however, he or she is offered the choice of reading or writing to the databases. Selecting reading prevents the user from making any permanent changes to the databases, and allows other users to come in and use CFM at the same time. Selecting writing instead will give the user write access to the databases, but will lock all other users out until the changes are completed. This is the normal way of updating and changing calibration information.

Note that when CFM is in read-only mode, one can appear to alter the contents of the databases. These changes will not be saved, however, and exist only in memory for the duration of the user's CFM session. CFM will warn the user that the changes he or she is making are impermanent. This feature can be used to test that a complex command will in fact produce the desired changes to the databases, without actually modifying them. Even in write mode, changes that have been made to the databases can be cancelled by using the QUIT command.

The CFM command line uses syntax conventions that are similar to those of VMS. In other words, commands are case-insensitive, and qualifiers are denoted by preceeding slash ("/") characters. Command arguments are delineated by any number of blanks. Commands, arguments, and qualifiers can be abbreviated any amount, so long as they remain unambiguous. Numerical arguments are always in the form of integers, or comma-separated lists of integers with optional ranges denoted by colons (eg. "`1, 4, 8:12`").
A summary of all CFM commands and their uses is given in the following table:

Command	**Purpose**
`ADD`	add new runs, sets, or types to the databases
`COPY`	copy one set to another
`DELETE`	remove runs, sets, or types from the databases
`EXIT`	leave CFM, saving changes, if any
`FIND`	search for specific sets, types, or calibration files
`HELP`	use the CFM online help facility
`MODIFY`	alter run, set, or type data already in the databases
`QUIT`	leave CFM, and do not save changes
`REPLACE`	replace all instances of a certain set or file with another
`SHOW`	display runs, sets, or types
`STATUS`	summarize the current status of the databases
`UPDATE`	save all changes

Each of these commands is discussed in more detail below. In the command descriptions, various terms are used that have been explained above, but are summarized here for convenience:

`attribute:` a 1-3 character description of a calibration.
`cdf:` a Calibration Data File.
`cdt:` a Calibration Data Type; a component-attribute combination.
`component:` a 1-5 character description of an experimental or detector component.
`icdf:` a number for a Calibration Data File; same as an indicator.
`icdt:` a number that corresponds to a `cdt`.

`indicator`: an integer from 1 to 32767 used to identify specific calibration files.
`list`: a comma separated list of integer values, possibly including ranges of values.
`run`: an experimental run number.
`set`: a numbered collection of calibration files used by a group of runs.

4. CFM Commands

In command syntax descriptions, CFM keywords are shown in `UPPER CASE`, while variable user input is shown in `lower case`. Optional arguments and qualifiers are enclosed in brackets. The character "`|`" is used to represent "or".

ADD

Syntax:
```
ADD SET [indicator_list]
ADD RUNS run_list set
ADD TYPE component attribute
```

The `ADD` command adds a new set, a series of new runs, or a new type to the databases. A new set will be numbered one higher than the current highest numbered set, and will contain indicators that are set to zero, unless the argument `indicator_list` is provided. If `indicator_list` is given, each indicator in the list will be assigned to its respective type in that set. All remaining unassigned indicators will be set to zero. New runs will be assigned the given set, provided they are currently undefined (*ie.* are assigned to set zero). Runs which are already defined will not be changed (use the `MODIFY` command instead). A new type will be numbered one higher than the current highest numbered type.

Examples:		
	`ADD SET 0,0,4,5,84,0,1`	create a new set with the given indicators
	`ADD RUNS 2000:3500 25`	define the given runs to use set 25
	`ADD TYPE DC PED`	define a new type called `DC_PED`

COPY

Syntax:
```
COPY set
```

The `COPY` command copies the given set to a new set. This is the most convenient way of extending the set database, if sets are large and contain many common files between each other.

DELETE

Syntax:
```
DELETE RUNS run_list
DELETE SETS set_list
DELETE TYPE icdt|(component attribute)
```

The `DELETE` command removes data from the databases. `DELETE RUN` causes the given runs to be assigned a set number of zero. `DELETE SET` removes the given sets from the databases, and the remaining sets are adjusted to fill in the resulting blank records. All runs that used the deleted sets are also deleted. `DELETE TYPE` removes the given type from the databases. The remaining types are adjusted to fill in the resulting blank record, and all occurences of calibration files for that type are deleted from the sets.

Examples:
```
DELETE SETS 11:14,17
DELETE TYPE DC PED
```

EXIT

Syntax:
```
EXIT
```

The `EXIT` command ends the CFM session, saving all changes that have been made. Changes will not be saved if the user is in read-only mode.

FIND

Syntax: `FIND RUNS setlist`
`FIND icdt|(component attribute) [indicator]`

The `FIND` command locates specific data in the databases. The first form of the command lists all runs that use sets from the given list of sets. The second form of the command finds all runs and sets that use a calibration file of the given type, and displays which calibration file is used. If the argument `indicator` is provided, only those instances where the calibration file number matches the argument are displayed.

Examples: `FIND RUNS 43,45`	locates all runs that use set 43 or 45
`FIND DC PED`	locates all runs and sets that use any `DC_PED` calibration files.
`FIND DC PED 12`	locates all runs and sets that use the calibration file `DC_PED.00012`

HELP

Syntax: `HELP [command]`

The `HELP` command invokes the VMS help facility to explain CFM concepts and commands. This command only works in the VMS operating system.

MODIFY

Syntax: `MODIFY run_list icdt|(component attribute) indicator`
`MODIFY RUNS run_list set`
`MODIFY SETS set_list icdt|(component attribute) indicator`
`MODIFY TYPE icdt component attribute`

The `MODIFY` command changes the current values in the databases. The first form of this command changes the indicator for the given type, for the given range of runs. In other words, it makes every run in the run list use the calibration file given by the supplied type and indicator number. This command can result in the generation and deletion of data sets. `MODIFY RUNS` makes all runs in the run list use the given data set. If any of the runs in the list are undefined (*ie.* they use set zero), then they will not be changed (use the `ADD` command instead). `MODIFY SETS` changes the given data sets to use the calibration file described by the supplied type and indicator number. This command can result in the generation and deletion of data sets. `MODIFY TYPE` changes the type number to use the given component and attribute. All calibration files that used the previous type are simply renamed to use the new type (indicators are retained).

Examples: `MODIFY 500:800 DC PED 99`	Runs 500 through 800 should use the file `DC_PED.00099` instead of whatever `DC_PED` file they were using before.
`MODIFY RUNS 500:800 75`	Runs 500 though 800 should use data set 75 instead of the ones they were using before.
`MODIFY SETS 6,7 DC PED 12`	Sets 6 and 7 should use the file `DC_PED.00012` instead of whatever `DC_PED` file they were using before.
`MODIFY TYPE 19 DC MAP`	Change type 19 to be `DC_MAP`, instead of what it was before.

QUIT

Syntax: `QUIT`

The `QUIT` command terminates the CFM session, without saving changes. If changes have been made, and the user is in write mode, then he or she will be prompted to make sure the changes should be abandoned. Only changes made since the most recent `UPDATE` command will be discarded.

REPLACE

Syntax: `REPLACE SETS oldset newset`
`REPLACE ALL|run_list icdt|(component attribute) old_indicator new_indicator`

The `REPLACE` command replaces specific occurences of data with new data. `REPLACE SETS` will replace all occurences of `oldset` with `newset` (the old set is not deleted, however). In its second form, this command replaces all calibration files of the given type and with `old_indicator`, with ones that use `new_indicator`. The replacement is only made for the runs in the given list. If the keyword `ALL` is given, however, the replacement is made for all defined runs.

SHOW

Syntax: `SHOW RUNS [run_list]`
`SHOW SETS [/SUMMARY] [set_list]`
`SHOW TYPES [/ALPHA]`

The `SHOW` command displays the database contents. `SHOW RUNS` lists the runs in the run list, along with the sets that each is defined to use. If `run_list` is not given all runs are shown. `SHOW SETS` displays each set in the `set_list`, along with the range of runs and the total number of runs for which it is used. If the `/SUMMARY` qualifer is absent, every calibration file used by the set is displayed, and unused types are also shown (without indicators, of course). `SHOW TYPES` lists all the calibration data types that CFM knows about, in numerical order. If the `/ALPHA` qualifier is used, the types will be displayed in alphabetical order, instead.

UPDATE

Syntax: `UPDATE`

The `UPDATE` command causes all changes that have been made to the databases to be saved. It only functions if the user is in write mode.

5. Using CFM From Within an Application

Application programs can query CFM, asking for the names of calibration files to be used. There are five FORTRAN subroutines in the CFM library that are used for this purpose.

CFMGET

Usage:

```
istat = CFMGET(irun)
INTEGER*4 istat,CFMGET,irun
```

This function tells CFM that the user will be asking for calibration file names for the given run number. The first time `CFMGET` is called, the databases will be opened, and they will remain open until the application exits. If run number 0 is given, the databases are opened, but no data is read. This is a convenient way to initialize CFM from the application, since it will warn you in advance if the databases are unreadable. The possible return codes of CFMGET are as follows:

0	successfully opened/read databases
-1	invalid `irun` argument (negative or beyond database bounds)
1	could not open database files
2	error reading databases
3	databases are temporarily locked (another user is writing to them)

CDFGET

Usage:

```
CALL CDFGET(irun,component,attribute,filename,newfile)
INTEGER*4 irun
CHARACTER component*(*), attribute*(*), filename*80
LOGICAL*1 newfile
```

CDFGET fetches the calibration data file name that is to be used on the given run for the given calibration data type. The application must provide the run number (irun), and the component and attribute of the calibration data type. If the run number is different than that which CFM is currently using, then CFMGET is called for the new run. In VMS, the file name is returned with "CAL$DB:" prepended to it. In Unix, this directory spec is automatically expanded to the full path name. If this calibration file has not been requested before, the flag newfile is set to .TRUE., otherwise it is set to .FALSE.. This flag can be used to prevent the unnecessary re-reading of calibration files that may be used over and over in the course of data analysis.

CDFUSE

Usage:
```
CALL CDFUSE(component,attribute,filename)
CHARACTER*(*) component,attribute,filename
```

CDFUSE can be used to override the calibration file definitions in the CFM databases. If a user requires a different calibration file for a certain calibration data type, he or she can call CDFUSE, giving the component, attribute, and new file name (which does not necessarily have to follow the CFM naming conventions). Whenever a calibration file for that calibration data type is requested in the future, the replacement file name will be returned instead of file name in the CFM databases. For example, if a user wants to test out a new version of a calibration file for drift chamber pedstals, called PEDESTALS.DAT, he or she might make the following call:

```
CALL CDFUSE('DC','PED','DISK:[MYDIR]PEDESTALS.DAT')
```

All subsequent requests for a DC_PED calibration file will return this file name instead of the normal CFM file name. A CDFUSE request can be cancelled by giving a blank file name. For example, to cancel the above request, make the following call:

```
CALL CDFUSE('DC','PED',' ')
```

All subsequent requests for a DC_PED calibration file will then return the normal CFM file name.

GETCDF

Usage:
```
CALL GETCDF(irun,component,attribute,filename,newfile)
INTEGER*4 irun
CHARACTER component*(*), attribute*(*), filename*80
LOGICAL*1 newfile
```

GETCDF functions identically to CDFGET, except that it stores the entire CFM database in memory, so as not to tie them up and prevent others from updating them. It faster and less obtrusive than the CFMGET/CDFGET combination, but uses up significantly more memory. It should not be used in conjunction with CDFGET or CFMGET. A call with irun equal to zero can be used to initially read the databases and report any access problems; otherwise the databases will be read on the first call.

SHOWCDF

Usage:
```
INCLUDE 'CFM$SOURCE:CFMMAIN.CMN'
CALL SHOWCDF(runset(irun))
```

SHOWCDF displays all calibration files being used in run IRUN, including those defined with CDFUSE calls. This routine only works in conjunction with GETCDF, not with CFMGET or CDFGET.

6. CFM Limitations

CFM does not manage the calibration files themselves, and does not concern itself with the contents of these files. These issues are the responsibility of the experimenters. CFM assumes that all calibration files (except those referred to in a call to CDFUSE) reside in the directory referred to by the logical name CAL$DB ($CAL_DB on Unix systems).

CFM has a maximum number of runs, sets, and types that it will monitor. These limits are determined at compile time, and are PARAMETERs in the include file `CFM.PAR`. To extend the databases beyond these limits, these parameters must be adjusted, and CFM recompiled. No alteration of the database files is necessary. As of this writing, the default limits of CFM for BNL Experiment 787 were:

Maximum number of runs	= 50000
Maximum number of sets	= 500
Maximum number of types	= 256

These parameters can also be reduced to decrease the memory requirements of CFM, if they are excessive for the needs of the experiment.

In busy analysis environments where there is almost always some kind of analysis going on, and the databases are in more or less constant use, it can be difficult to find a window of opportunity to write to the databases. If this is a problem, the `GETCDF` routine can be used to fetch calibration file names instead of `CDFGET`. Otherwise, an acceptible solution is to copy the contents of the `CFM$DB` directory to a private directory, and redefine `CFM$DB` to point to this directory instead of the normal CFM area. One can then run CFM on this private version of the databases, make all the required changes, and copy the new databases back into the official area. *MAKE SURE THAT THE FILE PROTECTIONS AND OWNERSHIPS ARE CORRECT WHEN YOU DO THIS!* It is best to do it as the user ID that normally owns the CFM files.

CFM is sensitive to the state of its lock files, which tell it how the databases are currently being used, and whether users are permitted to use CFM in read-only or write modes. These lock files should always have world read AND write access for CFM to work properly. Of course, the databases must have world read access.

7. Other CFM Files

CFM maintains a log of all input and output during a CFM session, which is stored in a file referenced by the logical name `CFMLOG`. Normally, `CFMLOG` is defined as `CFM$LOG:CFM.LOG`, but a user can redefine it if he or she wishes.

CFM also maintains a record of all modifications made to the databases by users in write mode. This information is stored in the file `CFM$DB:CFM.RECORD`, with one entry per line. Each entry consists of the following fields: the name of the user who made the modification, the time and date of the modification, and the CFM command line that altered the databases. This file can be used to monitor updates that are made to the databases, and track down the sources of any problems that may arise. These entries are added to `CFM$DB:CFM.RECORD` as soon as the command resulting in the modification is complete. If the user later decides to `QUIT` CFM, abandoning the modifications, the record of the aborted attempt to alter the databases is still retained, although it will be terminated by a note saying that the changes were not saved.

CFM uses two lock files to monitor the current usage of the databases. One is `CFM$DB:CFM.READLOCK`, which tells CFM whether or not a new user is permitted to read the databases at that time, and the other is `CFM$DB:CFM.WRITELOCK`, which tells CFM whether or not a new user is permitted to write to the databases. These files contain no data; CFM only tests its write access to the files to determine what it needs to know.