GNU dbm

A Database Manager

by Philip A. Nelson

Manual by Pierre Gaumond and Philip A. Nelson
Updated by Jason Downs

Edition 1.4.1

for GNU `dbm`, Version 1.7.3.

This is Edition 1.4.1 of the *GNU* `dbm` *Manual*, for `gdbm` Version 1.7.3.
Last updated May 19, 1994

Published by the Free Software Foundation
675 Massachusetts Avenue,
Cambridge, MA 02139 USA

1 Copying Conditions.

This library is *free*; this means that everyone is free to use it and free to redistribute it on a free basis. GNU `dbm` (`gdbm`) is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of `gdbm` that they might get from you.

Specifically, we want to make sure that you have the right to give away copies `gdbm`, that you receive source code or else can get it if you want it, that you can change these functions or use pieces of them in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies `gdbm`, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for anything in the `gdbm` distribution. If these functions are modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

`gdbm` is currently distributed under the terms of the GNU General Public License, Version 2. (*NOT* under the GNU General Library Public License.) A copy the GNU General Public License is included with the distribution of `gdbm`.

2 Introduction to GNU dbm.

GNU dbm (gdbm)is a library of database functions that use extendible hashing and works similar to the standard UNIX dbm functions. These routines are provided to a programmer needing to create and manipulate a hashed database. (gdbm is *NOT* a complete database package for an end user.)

The basic use of gdbm is to store key/data pairs in a data file. Each key must be unique and each key is paired with only one data item. The keys can not be directly accessed in sorted order. The basic unit of data in gdbm is the structure:

```
typedef struct {
           char *dptr;
           int  dsize;
        } datum;
```

This structure allows for arbitrary sized keys and data items.

The key/data pairs are stored in a gdbm disk file, called a gdbm database. An application must open a gdbm database to be able manipulate the keys and data contained in the database. gdbm allows an application to have multiple databases open at the same time. When an application opens a gdbm database, it is designated as a reader or a writer. A gdbm database opened by at most one writer at a time. However, many readers may open the database open simultaneously. Readers and writers can not open the gdbm database at the same time.

3 List of functions.

The following is a quick list of the functions contained in the gdbm library. The include file gdbm.h, that can be included by the user, contains a definition of these functions.

```
#include <gdbm.h>

GDBM_FILE gdbm_open(name, block_size, flags, mode, fatal_func);
void gdbm_close(dbf);
int gdbm_store(dbf, key, content, flag);
datum gdbm_fetch(dbf, key);
int gdbm_delete(dbf, key);
datum gdbm_firstkey(dbf);
datum gdbm_nextkey(dbf, key);
int gdbm_reorganize(dbf);
void gdbm_sync(dbf);
int gdbm_exists(dbf, key);
char *gdbm_strerror(errno);
int gdbm_setopt(dbf, option, value, size)
```

The gdbm.h include file is often in the '/usr/local/gnu/include' directory. (The actual location of gdbm.h depends on your local installation of gdbm.)

4 Opening the database.

Initialize `gdbm` system. If the file has a size of zero bytes, a file initialization procedure is performed, setting up the initial structure in the file.

The procedure for opening a `gdbm` file is:

```
GDBM_FILE dbf;

dbf = gdbm_open(name, block_size, flags, mode, fatal_func);
```

The parameters are:

char *name
: The name of the file (the complete name, `gdbm` does not append any characters to this name).

int block_size
: It is used during initialization to determine the size of various constructs. It is the size of a single transfer from disk to memory. This parameter is ignored if the file has been previously initialized. The minimum size is 512. If the value is less than 512, the file system blocksize is used, otherwise the value of `block_size` is used.

int flags
: If `flags` is set to GDBM_READER, the user wants to just read the database and any call to `gdbm_store` or `gdbm_delete` will fail. Many readers can access the database at the same time. If `flags` is set to GDBM_WRITER, the user wants both read and write access to the database and requires exclusive access. If `flags` is set to GDBM_WRCREAT, the user wants both read and write access to the database and if the database does not exist, create a new one. If `flags` is set to GDBM_NEWDB, the user want a new database created, regardless of whether one existed, and wants read and write access to the new database. For all writers (GDBM_WRITER, GDBM_WRCREAT and GDBM_NEWDB) the value GDBM_FAST can be added to the `flags` field using logical or. This option causes `gdbm` to write the database without any disk file synchronization. This allows faster writes, but may produce an inconsistent database in the event of abnormal termination of the writer. Any error detected will cause a return value of NULL and an appropriate value will be in `gdbm_errno` (see Variables). If no errors occur, a pointer to the `gdbm` file descriptor will be returned.

int mode
: File mode (see chmod(2) and open(2) if the file is created).

void (*fatal_func) ()

A function for `gdbm` to call if it detects a fatal error. The only parameter of this function is a string. If the value of NULL is provided, `gdbm` will use a default function.

The return value, `dbf`, is the pointer needed by all other functions to access that `gdbm` file. If the return is the NULL pointer, `gdbm_open` was not successful. The errors can be found in `gdbm_errno` for `gdbm` errors and in `errno` for file system errors (for error codes, see `gdbm.h`).

In all of the following calls, the parameter `dbf` refers to the pointer returned from `gdbm_open`.

5 Closing the database.

It is important that every file opened is also closed. This is needed to update the reader/writer count on the file. This is done by:

```
gdbm_close(dbf);
```

The parameter is:

GDBM_FILE dbf
: The pointer returned by `gdbm_open`.

Closes the `gdbm` file and frees all memory associated with the file `dbf`.

6 Inserting and replacing records in the database.

The function `gdbm_store` inserts or replaces records in the database.

```
ret = gdbm_store(dbf, key, content, flag);
```

The parameters are:

GDBM_FILE dbf
: The pointer returned by `gdbm_open`.

datum key
: The `key` data.

datum content
: The data to be associated with the key.

int flag
: Defines the action to take when the key is already in the database. The value GDBM_REPLACE (defined in `gdbm.h`) asks that the old data be replaced by the new `content`. The value GDBM_INSERT asks that an error be returned and no action taken if the `key` already exists.

The values returned in `ret` are:

-1
: The item was not stored in the database because the caller was not an official writer or either `key` or `content` have a NULL dptr field. Both `key` and `content` must have the dptr field be a non-NULL value. Since a NULL dptr field is used by other functions to indicate an error, a NULL field cannot be valid data.

+1
: The item was not stored because the argument `flag` was GDBM_INSERT and the `key` was already in the database.

0
: No error. `content` is keyed by `key`. The file on disk is updated to reflect the structure of the new database before returning from this function.

If you store data for a `key` that is already in the data base, `gdbm` replaces the old data with the new data if called with GDBM_REPLACE. You do not get two data items for the same `key` and you do not get an error from `gdbm_store`.

The size in `gdbm` is not restricted like `dbm` or `ndbm`. Your data can be as large as you want.

7 Searching for records in the database.

Looks up a given `key` and returns the information associated with that key. The pointer in the structure that is returned is a pointer to dynamically allocated memory block. To search for some data:

```
content = gdbm_fetch(dbf, key);
```

The parameters are:

GDBM_FILE dbf
: The pointer returned by `gdbm_open`.

datum key
: The `key` data.

The datum returned in `content` is a pointer to the data found. If the dptr is NULL, no data was found. If dptr is not NULL, then it points to data allocated by malloc. `gdbm` does not automatically free this data. The user must free this storage when done using it. This eliminates the need to copy the result to save it for later use (you just save the pointer).

You may also search for a particular key without retrieving it, using:

```
ret = gdbm_exists(dbf, key);
```

The parameters are:

GDBM_FILE dbf
: The pointer returned by `gdbm_open`.

datum key
: The `key` data.

Unlike `gdbm_fetch`, this routine does not allocate any memory, and simply returns true or false, depending on whether the `key` exists, or not.

8 Removing records from the database.

To remove some data from the database:

```
ret = gdbm_delete(dbf, key);
```

The parameters are:

GDBM_FILE dbf
: The pointer returned by `gdbm_open`.

datum key
: The `key` data.

The ret value is -1 if the item is not present or the requester is a reader. The ret value is 0 if there was a successful delete.

`gdbm_delete` removes the keyed item and the `key` from the database `dbf`. The file on disk is updated to reflect the structure of the new database before returning from this function.

9 Sequential access to records.

The next two functions allow for accessing all items in the database. This access is not `key` sequential, but it is guaranteed to visit every `key` in the database once. The order has to do with the hash values. `gdbm_firstkey` starts the visit of all keys in the database. `gdbm_nextkey` finds and reads the next entry in the hash structure for `dbf`.

```
key = gdbm_firstkey(dbf);

nextkey = gdbm_nextkey(dbf, key);
```

The parameters are:

GDBM_FILE dbf
: The pointer returned by `gdbm_open`.

datum `key`

datum nextkey
: The `key` data.

The return values are both datum. If `key`.dptr or nextkey.dptr is NULL, there is no first `key` or next `key`. Again notice that dptr points to data allocated by malloc and `gdbm` will not free it for you.

These functions were intended to visit the database in read-only algorithms, for instance, to validate the database or similar operations.

File `visiting` is based on a `hash table`. `gdbm_delete` re-arranges the hash table to make sure that any collisions in the table do not leave some item `un-findable`. The original key order is NOT guaranteed to remain unchanged in ALL instances. It is possible that some key will not be visited if a loop like the following is executed:

```
key = gdbm_firstkey ( dbf );
while ( key.dptr ) {
   nextkey = gdbm_nextkey ( dbf, key );
   if ( some condition ) {
      gdbm_delete ( dbf, key );
      free ( key.dptr );
   }
   key = nextkey;
}
```

10 Database reorganization.

The following function should be used very seldom.

```
ret = gdbm_reorganize(dbf);
```

The parameter is:

GDBM_FILE dbf
: The pointer returned by `gdbm_open`.

If you have had a lot of deletions and would like to shrink the space used by the `gdbm` file, this function will reorganize the database. `gdbm` will not shorten the length of a `gdbm` file (deleted file space will be reused) except by using this reorganization.

This reorganization requires creating a new file and inserting all the elements in the old file `dbf` into the new file. The new file is then renamed to the same name as the old file and `dbf` is updated to contain all the correct information about the new file. If an error is detected, the return value is negative. The value zero is returned after a successful reorganization.

11 Database Synchronization

If your database was opened with the GDBM_FAST flag, `gdbm` does not wait for writes to the disk to complete before continuing. This allows faster writing of databases at the risk of having a corrupted database if the application terminates in an abnormal fashion. The following function allows the programmer to make sure the disk version of the database has been completely updated with all changes to the current time.

```
gdbm_sync(dbf);
```

The parameter is:

GDBM_FILE dbf
: The pointer returned by `gdbm_open`.

This would usually be called after a complete set of changes have been made to the database and before some long waiting time. `gdbm_close` automatically calls the equivalent of `gdbm_sync` so no call is needed if the database is to be closed immediately after the set of changes have been made.

12 Error strings.

To convert a `gdbm` error code into English text, use this routine:

```
ret = gdbm_strerror(errno)
```

The parameter is:

gdbm_error errno
: The `gdbm` error code, usually `gdbm_errno`.

The appropiate phrase for reading by humans is returned.

13 Seting options.

Gdbm now supports the ability to set certain options on an already open database.

```
ret = gdbm_setopt(dbf, option, value, size)
```

The parameters are:

GDBM_FILE dbf
: The pointer returned by gdbm_open.

int option
: The option to be set.

int *value
: A pointer to the value to which option will be set.

int size
: The length of the data pointed to by value.

The valid options are currently:

GDBM_CACHESIZE - Set the size of the internal bucket cache. This option may only be set once on each GDBM_FILE descriptor, and is set automatically to 100 upon the first access to the database.

GDBM_FASTMODE - Set fast mode to either on or off. This allows fast mode to be toggled on an already open and active database. value (see below) should be set to either TRUE or FALSE.

The return value will be -1 upon failure, or 0 upon success. The global variable gdbm_errno will be set upon failure.

For instance, to set a database to use a cache of 10, after opening it with gdbm_open, but prior to accessing it in any way, the following code could be used:

```
int value = 10;
ret = gdbm_setopt(dbf, GDBM_CACHESIZE, &value, sizeof(int));
```

14 Two useful variables.

The following two variables are variables that may need to be used:

gdbm_error gdbm_errno
: The variable that contains more information about `gdbm` errors (`gdbm.h` has the definitions of the error values).

const char * gdbm_version
: The string containing the version information.

15 Compatibility with standard dbm and ndbm.

GNU dbm files are not sparse. You can copy them with the UNIX cp command and they will not expand in the copying process.

There is a compatibility mode for use with programs that already use UNIX dbm and UNIX ndbm.

GNU dbm has compatibility functions for dbm. For dbm compatibility functions, you need the include file dbm.h.

In this compatibility mode, no gdbm file pointer is required by the user, and Only one file may be opened at a time. All users in compatibility mode are assumed to be writers. If the gdbm file is a read only, it will fail as a writer, but will also try to open it as a reader. All returned pointers in datum structures point to data that gdbm WILL free. They should be treated as static pointers (as standard UNIX dbm does). The compatibility function names are the same as the UNIX dbm function names. Their definitions follow:

```
int dbminit(name);
int store(key, content);
datum fetch(key);
int delete(key);
datum firstkey();
datum nextkey(key);
int dbmclose();
```

Standard UNIX dbm and GNU dbm do not have the same data format in the file. You cannot access a standard UNIX dbm file with GNU dbm! If you want to use an old database with GNU dbm, you must use the conv2gdbm program.

Also, GNU dbm has compatibility functions for ndbm. For ndbm compatibility functions, you need the include file ndbm.h.

Again, just like ndbm, any returned datum can be assumed to be static storage. You do not have to free that memory, the ndbm compatibility functions will do it for you.

The functions are:

```
DBM *dbm_open(name, flags, mode);
```

```
void dbm_close(file);
datum dbm_fetch(file, key);
int dbm_store(file, key, content, flags);
int dbm_delete(file, key);
datum dbm_firstkey(file);
datum dbm_nextkey(file);
int dbm_error(file);
int dbm_clearerr(file);
int dbm_dirfno(file);
int dbm_pagfno(file);
int dbm_rdonly(file);
```

If you want to compile an old C program that used UNIX `dbm` or `ndbm` and want to use `gdbm` files, execute the following `cc` command:

```
cc ... -L /usr/local/lib -lgdbm
```

16 Converting dbm files to gdbm format.

The program conv2gdbm has been provided to help you convert from dbm databases to gdbm. The usage is:

```
conv2gdbm [-q] [-b block_size] dbm_file [gdbm_file]
```

The options are:

-q
: Causes conv2gdbm to work quietly.

block_size
: Is the same as in gdbm_open.

dbm_file
: Is the name of the dbm file without the .pag or .dir extensions.

gdbm_file
: Is the complete file name. If not included, the gdbm file name is the same as the dbm file name without any extensions. That is conv2gdbm dbmfile converts the files dbmfile.pag and dbmfile.dir into a gdbm file called dbmfile.

17 Problems and bugs.

If you have problems with GNU `dbm` or think you've found a bug, please report it. Before reporting a bug, make sure you've actually found a real bug. Carefully reread the documentation and see if it really says you can do what you're trying to do. If it's not clear whether you should be able to do something or not, report that too; it's a bug in the documentation!

Before reporting a bug or trying to fix it yourself, try to isolate it to the smallest possible input file that reproduces the problem. Then send us the input file and the exact results `gdbm` gave you. Also say what you expected to occur; this will help us decide whether the problem was really in the documentation.

Once you've got a precise problem, send e-mail to:

```
Internet: 'bug-gnu-utils@prep.ai.mit.edu'.
UUCP: 'mit-eddie!prep.ai.mit.edu!bug-gnu-utils'.
```

Please include the version number of GNU `dbm` you are using. You can get this information by printing the variable `gdbm_version` (see Variables).

Non-bug suggestions are always welcome as well. If you have questions about things that are unclear in the documentation or are just obscure features, please report them too.

You may contact the author by:


```
  e-mail:  phil@cs.wwu.edu
 us-mail:  Philip A. Nelson
           Computer Science Department
           Western Washington University
           Bellingham, WA 98226
```


You may contact the current maintainer by:


```
  e-mail:  downsj@CSOS.ORST.EDU
```