GNU dbm

A Database Manager

by Philip A. Nelson, Jason Downs and Sergey Poznyakoff

Manual by Pierre Gaumond, Philip A. Nelson, Jason Downs,
Sergey Poznyakoff, and Terence Kelly

Edition 1.22

for GNU `dbm`, Version 1.22

Published by the Free Software Foundation, 51 Franklin Street, Fifth Floor Boston, MA 02110-1301, USA

Short Contents

Table of Contents

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 of `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 of `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 3. (*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 extensible hashing and work 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. In particular, zero-length keys or data (`dsize = 0`) are allowed. However, the `dptr` field is required to point to a valid memory location. In other words, `dptr` cannot be NULL. Note also that its type is `char *` for purely historic reasons. You can use any C data type (either scalar or aggregate) both as for key and for data.

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 it. 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 can be opened by at most one writer at a time. However, many readers may open the database simultaneously. Readers and writers can not open the GDBM database at the same time.

Speaking about *application* we usually mean a separate process. However, it is entirely normal for a multi-thread program to operate as a GDBM reader in one thread and writer in another, provided, of course, that the two threads don't operate on the same database simultaneously.

To use the GDBM functions, the programmer must first include the header file `gdbm.h`.

This file defines, among others, the GDBM_FILE data type, an opaque pointer to the structure that represents the opened GDBM database. To access the database, the programmer must first open it using the `gdbm_open` function. The function takes several arguments, the name of the database file being one of them, and returns a GDBM_FILE object on success. This object is then passed to other functions in order to manipulate the database. When the database is no longer needed, the programmer *closes* it using the `gdbm_close` call.

These and other functions are discussed in detail in chapters that follow. Here we show an example illustrating the use of GDBM to look up a key in the database.

```
#include <stdio.h>
#include <string.h>
#include <gdbm.h>

int
main (int argc, char **argv)
```

```
{
  GDBM_FILE gdbf;     /* Database file object pointer */
  datum key, content; /* Key and content data */
  int status = 0;     /* Exit status of the program: 0 - OK, 1 - key
                         not found, 2 - error. */

  /*
   * Validate arguments.
   */
  if (argc != 3)
    {
      fprintf (stderr, "usage: %s DBFILE KEY\n", argv[0]);
      return 2;
    }

  /*
   * Open the database.  The GDBM_READER flag indicates that we only
   * intend to read from it.
   */
  gdbf = gdbm_open (argv[1], 0, GDBM_READER, 0, NULL);
  if (gdbf == NULL)
    {
      fprintf (stderr, "can't open database: %s\n",
               gdbm_strerror (gdbm_errno));
    }

  /*
   * Prepare the lookup key.  Notice, that the terminating \0 character
   * is not counted in the dsize computation.
   */
  key.dptr = argv[2];
  key.dsize = strlen (argv[2]);

  /*
   * Look up the key in the database.
   */
  content = gdbm_fetch (gdbf, key);

  /*
   * Analyze the return.
   */
  if (content.dptr != NULL)
    {
      /*
       * The key is found.  Print the content on the stdout and
       * indicate success.
       */
```

```
        fwrite (content.dptr, content.dsize, 1, stdout);
        putchar ('\n');
        status = 0;
      }
    else if (gdbm_errno == GDBM_ITEM_NOT_FOUND)
      {
        /*
         * There is no such key in the database.
         */
        fprintf (stderr, "no such key\n");
        status = 1;
      }
    else
      {
        /*
         * An error occurred.
         */
        fprintf (stderr, "%s\n", gdbm_db_strerror (gdbf));
        status = 2;
      }

    /*
     * Close the database and return.
     */
    gdbm_close (gdbf);
    return status;
}
```

To compile this example, run

```
cc -oexample example.c -lgdbm
```

To run it, you will need an example database. The easiest way to create it is by using the `gdbtool` program, which is part of the `GDBM` package (see Chapter 24 [gdbmtool], page 52):

```
$ gdbmtool test.gdbm store foo bar
```

This creates database file `test.gdbm` and stores a single record in it. The record's key is '`foo`', and the value is '`bar`'. Now you can run the example program to see how it works:

```
$ ./example test.gdbm foo
bar
$ ./example test.gdbm baz
no such key
```

3 Opening the database

`GDBM_FILE gdbm_open` (*const char* ***name***, *int* ***block_size***, *int* ***flags***, *int* ***mode***, *void* (****fatal_func***)(*const char* *)) [gdbm interface]

Opens or creates a `GDBM` database file.

The arguments are:

name The name of the file (the complete name, `GDBM` does not append any characters to this name).

block_size This parameter is used only when `gdbm_open` has to create a new database file and represents the size of a single transfer from disk to memory. If its value is less than 512, the file system block size is used instead. The size is adjusted so that the block can hold exact number of directory entries, so that the effective block size can be slightly greater than requested. However, if the `GDBM_BSEXACT` flag is set and the size needs to be adjusted, the function will return with error status, setting the `gdbm_errno` variable to `GDBM_BLOCK_SIZE_ERROR`.

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 wants it created if it does not already exist. 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. If an existing database file is opened with the `GDBM_NEWDB` flag, the existing data are destroyed, and an empty database structure is created in its place.

The following constants may also be logically or'd into the database flags:

`GDBM_CLOEXEC` [gdbm_open flag]
Set the close-on-exec flag on the database file descriptor. The `libc` must support the `O_CLOEXEC` flag (see Section "O_CLOEXEC" in *open(2) man page*).

`GDBM_NOLOCK` [gdbm_open flag]
Don't lock the database file. Use this flag if you intend to do locking separately. See Chapter 19 [Locking], page 39.

`GDBM_NOMMAP` [gdbm_open flag]
Disable memory mapping mechanism. Note, that this degrades performance.

`GDBM_PREREAD` [gdbm_open flag]
When mapping `GDBM` file to memory, read its contents immediately, instead of when needed (*prefault reading*). This can be advantageous if you open a *read-only* database and are going to do a lot

of look-ups on it. In this case entire database will be pre-read and look-ups will operate on an in-memory copy. In contrast, `GDBM_PREREAD` should not be used if you open a database (even in read-only mode) only to do a couple of look-ups. Finally, never use `GDBM_PREREAD` when opening a database for updates, especially for inserts: this will degrade performance.

This flag has no effect if `GDBM_NOMMAP` is given, or if the operating system does not support prefault reading. It is known to work on Linux and FreeBSD kernels.

`GDBM_XVERIFY` [gdbm_open flag]

Enable additional consistency checks. With this flag, eventual corruptions of the database are discovered when opening it, instead of when a corrupted structure is read during normal operation. However, on large databases, it can slow down the opening process.

See Chapter 21 [Additional functions], page 42.

The following additional flags are valid when the database is opened for writing (i.e. together with `GDBM_WRITER`, `GDBM_WRCREAT`, or `GDBM_NEWDB`):

`GDBM_SYNC` [gdbm_open flag]

Synchronize all database operations to disk immediately. Notice, that this option entails severe performance degradation and does not necessarily ensure that the resulting database state is consistent. In general, we discourage its use (see Chapter 11 [Sync], page 17). See Chapter 17 [Crash Tolerance], page 28, for a discussion of how to ensure database consistency with minimal performance overhead.

`GDBM_FAST` [gdbm_open flag]

A reverse of `GDBM_SYNC`. Synchronize writes only when needed. This is the default. The flag is provided for compatibility with previous versions of `GDBM`.

The following flags can be used together with `GDBM_NEWDB`. They also take effect when used with `GDBM_WRCREAT`, if the requested database file doesn't exist:

`GDBM_BSEXACT` [gdbm_open flag]

If this flag is set and the requested *block_size* cannot be used without adjustment, `gdbm_open` will refuse to create the databases. In this case it will set the `gdbm_errno` variable to `GDBM_BLOCK_SIZE_ERROR` and return `NULL`.

`GDBM_NUMSYNC` [gdbm_open flag]

Useful only together with `GDBM_NEWDB`, this bit instructs `gdbm_open` to create new database in *extended database format*, a format best suitable for effective crash recovery. See Section 17.8 [Numsync], page 32, for a detailed discussion of this format, and Chapter 17 [Crash Tolerance], page 28, for a discussion of crash recovery.

mode File mode[1], which is used if the file is created.

fatal_func This parameter is deprecated and must always be `NULL`.

Early versions of `GDBM` (prior to 1.13) lacked proper error handling and would abort on any "fatal" error (such as out of memory condition, disk write error, or the like). In these versions, `fatal_func` was provided as a hook, allowing the caller to do proper cleanup before such abnormal exit. As of version 1.22, this functionality is deprecated, although still supported for backward compatibility.

The return value, 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` variable (see Chapter 20 [Variables], page 40). Available error codes are discussed in Chapter 22 [Error codes], page 43.

In all of the following calls, the parameter *dbf* refers to the pointer returned from `gdbm_open` (or `gdbm_fd_open`, described below).

`GDBM_FILE gdbm_fd_open` (*int* ***fd***, *const char* ****name***, *int* ***block_size***, *int* ***flags***, *int* ***mode***, *void* (****fatal_func***)(*const char* **)) [gdbm interface]

Alternative function for opening a `GDBM` database. The *fd* argument is the file descriptor of the database file obtained by a call to `open`(2), `creat`(2) or similar functions. The descriptor is not dup'ed, and will be closed when the returned `GDBM_FILE` is closed. Use `dup`(2) if that is not desirable.

In case of error, the function behaves like `gdbm_open` and *does not close fd*. This can be altered by the following value passed in the *flags* argument:

`GDBM_CLOERROR` [gdbm_open flag]

Close *fd* before exiting on error.

`int gdbm_copy_meta` (*GDBM_FILE* ***dst***, *GDBM_FILE* ***src***) [gdbm interface]

Copy file ownership and mode from *src* to *dst*.

[1] See Section "chmod" in *chmod(2) man page*, and See Section "open a file" in *open(2) man page*.

4 Closing the database

It is important that every file opened is also closed. This is needed to properly update its disk structure and maintain a consistent locking state on the file.

`int gdbm_close` (*GDBM_FILE dbf*) [gdbm interface]

This function closes the `GDBM` file and frees all memory associated with it. The parameter is:

dbf The pointer returned by `gdbm_open`.

`Gdbm_close` returns 0 on success. On error, it sets `gdbm_errno` and system `errno` variables to the codes describing the error and returns -1.

5 Number of Records

`int gdbm_count` (*GDBM_FILE* `dbf`, *gdbm_count_t* *`pcount`) [gdbm interface]
Counts the number of records in the database *dbf*. On success, stores it in the memory location pointed to by *pcount* and returns 0. On error, sets `gdbm_errno` (if relevant, also `errno`) and returns -1.

`int gdbm_bucket_count` (*GDBM_FILE* `dbf`, *size_t* *`pcount`) [gdbm interface]
Counts the number of buckets in the database *dbf*. On success, stores it in the memory location pointed to by *pcount* and return 0. On error, sets `gdbm_errno` (if relevant, also `errno`) and returns -1.

6 Inserting and replacing records in the database

`int gdbm_store` (*GDBM_FILE* `dbf`, *datum* `key`, *datum* `content`, *int* `flag`) [gdbm interface]

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

The parameters are:

dbf The pointer returned by `gdbm_open`.

key The search key.

content The data to be associated with the key.

flag Defines the action to take when the key is already in the database. The value `GDBM_REPLACE` 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.

This function can return the following values:

0 Success. The value of *content* is keyed by *key* in the database.

-1 An error occurred which prevented the item from being stored in the database. Examine the `gdbm_errno` variable to determine the actual cause of the error.

+1 The item was not stored because the argument *flag* was `GDBM_INSERT` and the *key* was already in the database. The `gdbm_errno` variable is set to `GDBM_CANNOT_REPLACE`.

If the function returns -1, `gdbm_errno` can have the following values:

`GDBM_READER_CANT_STORE`
: Database was open in read-only mode, i.e. with the `GDBM_READER` flag. See Chapter 3 [Open], page 5.

`GDBM_MALFORMED_DATA`
: Either *key* or *content* had their `dptr` field set to `NULL`.

  It is OK to have a *zero-length* key or content, i.e. a datum with `dsize` set to 0, but the `dptr` field must always be a non-NULL value.

`GDBM_BAD_HASH_TABLE`
: Database hash table is malformed. This usually means that some error in the application or the library caused memory overrun. The database is marked as needing recovery. All further calls on this database will return with `gdbm_error` set to `GDBM_NEED_RECOVERY`. See Chapter 16 [Recovery], page 26, for a discussion of database recovery process.

`GDBM_BAD_DIR_ENTRY`
: Database directory entry is corrupted. The database is marked as needing recovery. See Chapter 16 [Recovery], page 26.

`GDBM_BAD_BUCKET`
: Database bucket is corrupted. The database is marked as needing recovery. See Chapter 16 [Recovery], page 26.

`GDBM_BAD_AVAIL`
: Database available storage index is corrupted. The database is marked as needing recovery. See Chapter 16 [Recovery], page 26.

`GDBM_FILE_SEEK_ERROR`
: A seek error occurred on the underlying disk file. Examine the system `errno` variable for more detail.

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 of datum in `GDBM` is restricted only by the maximum value for an object of type `int` (type of the `dsize` member of `datum`).

7 Searching for records in the database

`datum gdbm_fetch` (*GDBM_FILE* ***dbf***, *datum* ***key***) [gdbm interface]

Looks up a given *key* and returns the information associated with it. The `dptr` field in the structure that is returned points to a memory block allocated by `malloc`. It is the caller's responsibility to free it when no longer needed.

If the `dptr` is `NULL`, inspect the value of the `gdbm_errno` variable (see Chapter 20 [Variables], page 40). If it is `GDBM_ITEM_NOT_FOUND`, no data was found. Any other value means an error occurred. Use `gdbm_strerror` function to convert `gdbm_errno` to a human-readable string.

The parameters are:

dbf The pointer returned by `gdbm_open`.

key The search key.

An example of using this function:

```
content = gdbm_fetch (dbf, key);
if (content.dptr == NULL)
  {
    if (gdbm_errno == GDBM_ITEM_NOT_FOUND)
      fprintf(stderr, "key not found\n");
    else
      fprintf(stderr, "error: %s\n", gdbm_db_strerror (dbf));
  }
else
  {
    /* do something with content.dptr */
  }
```

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

`int gdbm_exists` (*GDBM_FILE* ***dbf***, *datum* ***key***) [gdbm interface]

Checks whether the *key* exists in the database *dbf*.

If *key* is found, returns `true` (`1`). If it is not found, returns `false` (`0`) and sets `gdbm_errno` to `GDBM_NO_ERROR` (`0`).

On error, returns `0` and sets `gdbm_errno` to a non-`0` error code.

The parameters are:

dbf The pointer returned by `gdbm_open`.

key The search key.

8 Removing records from the database

To remove some data from the database, use the `gdbm_delete` function.

`int gdbm_delete (`*GDBM_FILE* `dbf`, *datum* `key``)` [gdbm interface]

Deletes the data associated with the given *key*, if it exists in the database *dbf*.

The parameters are:

dbf The pointer returned by `gdbm_open`.

datum key
The search key.

The function returns `-1` if the item is not present or if an error is encountered. Examine the `gdbm_errno` variable or the return from `gdbm_last_errno (dbf)` to know the reason.

The return of `0` marks a successful delete.

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`.

`datum gdbm_firstkey (`*GDBM_FILE* `dbf``)` [gdbm interface]

Initiate sequential access to the database *dbf*. The returned value is the first key accessed in the database. If the `dptr` field in the returned datum is `NULL`, inspect the `gdbm_errno` variable (see Chapter 20 [Variables], page 40). The value of `GDBM_ITEM_NOT_FOUND` means that the database contains no data. Other value means an error occurred.

On success, `dptr` points to a memory block obtained from `malloc`, which holds the key value. The caller is responsible for freeing this memory block when no longer needed.

`datum gdbm_nextkey (`*GDBM_FILE* `dbf`, *datum* `prev``)` [gdbm interface]

This function continues iteration over the keys in *dbf*, initiated by `gdbm_firstkey`. The parameter *prev* holds the value returned from a previous call to `gdbm_nextkey` or `gdbm_firstkey`.

The function returns next key from the database. If the `dptr` field in the returned datum is `NULL` inspect the `gdbm_errno` variable (see Chapter 20 [Variables], page 40). The value of `GDBM_ITEM_NOT_FOUND` means that all keys in the database has been visited. Any other value means an error occurred.

Otherwise, `dptr` points to a memory block obtained from `malloc`, which holds the key value. The caller is responsible for freeing this memory block when no longer needed.

These functions are intended to visit the database in read-only algorithms, for instance, to validate the database or similar operations. The usual algorithm for sequential access is:

```
key = gdbm_firstkey (dbf);
while (key.dptr)
  {
     datum nextkey;

     /* do something with the key */
     ...

     /* Obtain the next key */
     nextkey = gdbm_nextkey (dbf, key);
     /* Reclaim the memory used by the key */
     free (key.dptr);
     /* Use nextkey in the next iteration. */
     key = nextkey;
  }
```

Don't use `gdbm_delete` or `gdbm_store` in such a loop. File visiting is based on a *hash table*. The `gdbm_delete` function 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. So it is possible that some key will not be visited or will be visited twice, if a loop like the following is executed:

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

10 Database reorganization

The following function should be used very seldom.

`int gdbm_reorganize` (*GDBM_FILE dbf*) [gdbm interface]

Reorganizes the database.

The parameter is:

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. This results, in particular, in shortening the length of a `GDBM` file by removing the space occupied by deleted records.

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

Normally, `GDBM` functions don't flush changed data to the disk immediately after a change. This allows for 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.

`int gdbm_sync` (*GDBM_FILE* `dbf`) [gdbm interface]

Synchronizes the changes in *dbf* with its disk file. The parameter is a pointer returned by `gdbm_open`.

This function would usually be called after a complete set of changes have been made to the database and before some long waiting time. This set of changes should preserve application-level invariants. In other words, call `gdbm_sync` only when the database is in a consistent state with regard to the application logic, a state from which you are willing and able to recover. You can think about all database operations between two consecutive `gdbm_sync` calls as constituting a single *transaction*. See Section 17.3 [Synchronizing the Database], page 29, for a detailed discussion about how to properly select the synchronization points.

The `gdbm_close` function 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.

`Gdbm_sync` returns 0 on success. On error, it sets `gdbm_errno` and system `errno` variables to the codes describing the error and returns -1.

Opening the database with `GDBM_SYNC` flag ensures that `gdbm_sync` function will be called after each change, thereby flushing the changes to disk immediately. You are advised against using this flag, however, because it incurs a severe performance penalty, while giving only a moderate guarantee that the *structural* consistency of the database will be preserved in case of failure, and that only unless the failure occurs while being in the `fsync` call. For the ways to ensure proper *logical* consistency of the database, see Chapter 17 [Crash Tolerance], page 28.

12 Changing database format

As of version 1.22, `GDBM` supports databases in two formats: *standard* and *extended*. The standard format is used most often. The *extended* database format is used to provide additional crash resistance (see Chapter 17 [Crash Tolerance], page 28).

Depending on the value of the *flags* parameter in a call to `gdbm_open` (see Chapter 3 [Open], page 5), a database can be created in either format.

The format of an existing database can be changed using the `gdbm_convert` function:

`int gdbm_convert` (*GDBM_FILE* `dbf`, *int* `flag`) [gdbm interface]

Changes the format of the database file *dbf*. Allowed values for *flag* are:

`0` Convert database to the standard format.

`GDBM_NUMSYNC`
Convert database to the extended *numsync* format (see Section 17.8 [Numsync], page 32).

On success, the function returns 0. In this case, it should be followed by a call to `gdbm_sync` (see Chapter 11 [Sync], page 17) or `gdbm_close` (see Chapter 4 [Close], page 8) to ensure the changes are written to the disk.

On error, returns -1 and sets the `gdbm_errno` variable (see Chapter 20 [Variables], page 40).

If the database is already in the requested format, the function returns success (0) without doing anything.

13 Export and Import

GDBM databases can be converted into so-called *flat format* files. Such files cannot be used for searching, their sole purpose is to keep the data from the database for restoring it when the need arrives. There are two flat file formats, which differ in the way they represent the data and in the amount of meta-information stored. Both formats can be used, for example, to migrate between the different versions of GDBM databases. Generally speaking, flat files are safe to send over the network, and can be used to recreate the database on another machine. The recreated database is guaranteed to have the same format and contain the same set of key/value pairs as the database from which the flat file was created. However, it will not constitute a byte-to-byte equivalent of the latter. Various internal structures in the database can differ. In particular, ordering of key/value pairs can be different and the table of available file space will most probably differ, too. For databases in extended format, the `numsync` counter will be reset to 0 (see Section 17.8 [Numsync], page 32). These details are not visible to the application programmer, and are mentioned here only for completeness sake.

The fact that the restored database contains the same set of key/value pairs does not necessarily mean, however, that it can be used in the same way as the original one. For example, if the original database contained non-ASCII data (e.g. C structures, integers etc.), the recreated database can be of any use only if the target machine has the same integer size and byte ordering as the source one and if its C compiler uses the same packing conventions as the one which generated C which populated the original database. In general, such binary databases are not portable between machines, unless you follow some stringent rules on what data is written to them and how it is interpreted.

GDBM version 1.22 supports two flat file formats. The *binary* flat file format was first implemented in version 1.9.1. This format stores only key/data pairs, it does not keep information about the database file itself. As its name implies, files in this format are binary files. This format is supported for backward compatibility.

The *ascii* flat file format encodes all data in Base64 and stores not only key/data pairs, but also the original database file metadata, such as file name, mode and ownership. Files in this format can be sent without additional encapsulation over transmission channels that normally allow only ASCII data, such as, e.g. SMTP. Due to additional metadata they allow for restoring an exact copy of the database, including file ownership and privileges, which is especially important if the database in question contained some security-related data.

We call a process of creating a flat file from a database *exporting* or *dumping* this database. The reverse process, creating the database from a flat file is called *importing* or *loading* the database.

`int gdbm_dump` (*GDBM_FILE* `dbf`, *const char* *`filename`, *int* `format`, *int* `open_flags`, *int* `mode`) [gdbm interface]

Dumps the database file to the named file in requested format. Arguments are:

dbf A pointer to the source database, returned by a prior call to `gdbm_open`.

filename Name of the dump file.

format Output file format. Allowed values are: `GDBM_DUMP_FMT_BINARY` to create a binary dump and `GDBM_DUMP_FMT_ASCII` to create an ASCII dump file.

open_flags How to create the output file. If *flag* is `GDBM_WRCREAT` the file will be created if it does not exist. If it does exist, the `gdbm_dump` will fail.

If *flag* is `GDBM_NEWDB`, the function will create a new output file, replacing it if it already exists.

mode The permissions to use when creating the output file (see Section "open a file" in *open(2) man page*).

`int gdbm_load` (*GDBM_FILE* ***pdbf***, *const char* ****filename***, *int* ***flag***, *int* ***meta_mask***, *unsigned long* ****errline***) [gdbm interface]

Loads data from the dump file *filename* into the database pointed to by *pdbf*. The latter can point to `NULL`, in which case the function will try to create a new database. If it succeeds, the function will return, in the memory location pointed to by *pdbf*, a pointer to the newly created database. If the dump file carries no information about the original database file name, the function will set `gdbm_errno` to `GDBM_NO_DBNAME` and return `-1`, indicating failure.

The *flag* has the same meaning as the *flag* argument to the `gdbm_store` function (see Chapter 6 [Store], page 10).

The *meta_mask* argument can be used to disable restoring certain bits of file's metadata from the information in the input dump file. It is a binary OR of zero or more of the following:

GDBM_META_MASK_MODE
: Do not restore file mode.

GDBM_META_MASK_OWNER
: Do not restore file owner.

The function returns 0 upon successful completion or -1 on fatal errors and 1 on mild (non-fatal) errors.

If a fatal error occurs, `gdbm_errno` will be set to one of the following values:

GDBM_FILE_OPEN_ERROR
: Input file (*filename*) cannot be opened. The `errno` variable can be used to get more detail about the failure.

GDBM_MALLOC_ERROR
: Not enough memory to load data.

GDBM_FILE_READ_ERROR
: Reading from *filename* failed. The `errno` variable can be used to get more detail about the failure.

GDBM_MALFORMED_DATA
GDBM_ILLEGAL_DATA
: Input contained malformed data, i.e. it is not a valid `GDBM` dump file. This often means that the dump file got corrupted during the transfer.

  The `GDBM_ILLEGAL_DATA` is an alias for this error code, maintained for backward compatibility.

GDBM_ITEM_NOT_FOUND
: This error can occur only when the input file is in ASCII format. It indicates that the data part of the record about to be read lacked length specification. Application developers are advised to treat this error equally as `GDBM_MALFORMED_DATA`.

Mild errors mean that the function was able to successfully load and restore the data, but was unable to change the database file metadata afterwards. The table below lists possible values for `gdbm_errno` in this case. To get more detail, inspect the system `errno` variable.

GDBM_ERR_FILE_OWNER
: The function was unable to restore database file owner.

GDBM_ERR_FILE_MODE
: The function was unable to restore database file mode (permission bits).

If an error occurs while loading data from an input file in ASCII format, the number of line in which the error occurred will be stored in the location pointed to by the *errline* parameter, unless it is `NULL`.

If the line information is not available or applicable, *errline* will be set to `0`.

`int gdbm_dump_to_file` (*GDBM_FILE* `dbf`, *FILE* *`fp`, *int* `format`) [gdbm interface]

This is an alternative entry point to `gdbm_dump` (which see). Arguments are:

dbf A pointer to the source database, returned by a call to `gdbm_open`.

fp File to write the data to.

format Format of the dump file. See the *format* argument to the `gdbm_dump` function.

`int gdbm_load_from_file` (*GDBM_FILE* *`pdbf`, *FILE* *`fp`, *int* `replace`, *int* `meta_mask`, *unsigned long* *`line`) [gdbm interface]

This is an alternative entry point to `gdbm_load`. It writes the output to *fp* which must be a file open for writing. The rest of arguments is the same as for `gdbm_load` (excepting of course *flag*, which is not needed in this case).

`int gdbm_export` (*GDBM_FILE* `dbf`, *const char* *`exportfile`, *int* `flag`, *int* `mode`) [gdbm interface]

This function is retained for compatibility with GDBM 1.10 and earlier. It dumps the database to a file in binary dump format and is equivalent to

```
gdbm_dump(dbf, exportfile, GDBM_DUMP_FMT_BINARY, flag, mode)
```

`int gdbm_export_to_file` (*GDBM_FILE* `dbf`, *FILE* *`fp`) [gdbm interface]

This is an alternative entry point to `gdbm_export`. This function writes to file *fp* a binary dump of the database *dbf*.

`int gdbm_import` (*GDBM_FILE* `dbf`, *const char* *`importfile`, *int* `flag`) [gdbm interface]

This function is retained for compatibility with `GDBM` 1.10 and earlier. It loads the file *importfile*, which must be a binary flat file, into the database *dbf* and is equivalent to the following construct:

```
dbf = gdbm_open (importfile, 0,
                 flag == GDBM_REPLACE ?
                   GDBM_WRCREAT : GDBM_NEWDB,
                 0600, NULL);
gdbm_load (&dbf, exportfile, 0, flag, NULL)
```

`int gdbm_import_from_file` (*GDBM_FILE dbf*, *FILE *fp*, *int flag*) [gdbm interface]

An alternative entry point to `gdbm_import`. Reads the binary dump from the file *fp* and stores the key/value pairs to *dbf*. See Chapter 6 [Store], page 10, for a description of *flag*.

This function is equivalent to:

```
dbf = gdbm_open (importfile, 0,
                 flag == GDBM_REPLACE ?
                   GDBM_WRCREAT : GDBM_NEWDB,
                 0600, NULL);
gdbm_load_from_file (dbf, fp, flag, 0, NULL);
```

14 Error handling

The global variable `gdbm_errno` (see Chapter 20 [Variables], page 40) keeps the error code of the most recent error encountered by `GDBM` functions.

To convert this code to human-readable string, use the following function:

`const char * gdbm_strerror` (*gdbm_error* ***errno***) [gdbm interface]
: Converts *errno* (an integer value) into a human-readable descriptive text. Returns a pointer to a static string. The caller must not free the returned pointer or alter the string it points to.

Detailed information about the most recent error that occurred while operating on a `GDBM` file is stored in the `GDBM_FILE` object itself. To retrieve it, the following functions are provided:

`gdbm_error gdbm_last_errno` (*GDBM_FILE* ***dbf***) [gdbm interface]
: Returns the code of the most recent error encountered when operating on *dbf*.

  When `gdbm_last_errno` called immediately after the failed function, its return equals the value of the `gdbm_errno` variable. However, `gdbm_errno` can be changed if any `GDBM` functions (operating on another databases) were called afterwards, and `gdbm_last_errno` will always return the code of the last error that occurred while working with *that* database.

`int gdbm_last_syserr` (*GDBM_FILE* ***dbf***) [gdbm interface]
: Returns the value of the system `errno` variable associated with the most recent error.

  Notice, that not all `GDBM` errors have an associated system error code. The following are the ones that have:

  - GDBM_FILE_OPEN_ERROR
  - GDBM_FILE_WRITE_ERROR
  - GDBM_FILE_SEEK_ERROR
  - GDBM_FILE_READ_ERROR
  - GDBM_FILE_STAT_ERROR
  - GDBM_BACKUP_FAILED
  - GDBM_BACKUP_FAILED
  - GDBM_FILE_CLOSE_ERROR
  - GDBM_FILE_SYNC_ERROR
  - GDBM_FILE_TRUNCATE_ERROR

  For other errors, `gdbm_last_syserr` will return 0.

`int gdbm_check_syserr` (*gdbm_errno* ***err***) [gdbm interface]
: Returns `1`, if the system `errno` value should be inspected to get more info on the error described by `GDBM` error code *err*.

To get a human-readable description of the recent error for a particular database file, use the `gdbm_db_strerror` function:

`const char * gdbm_db_strerror` (*GDBM_FILE dbf*) [gdbm interface]
: Returns textual description of the most recent error encountered when operating on the database *dbf*. The resulting string is often more informative than what would be returned by `gdbm_strerror(gdbm_last_errno(dbf))`. In particular, if there is a system error associated with the recent failure, it will be described as well.

`void gdbm_clear_error` (*GDBM_FILE dbf*) [gdbm interface]
: Clears the error state for the database *dbf*. Normally, this function is called upon the entry to any `GDBM` function.

Certain errors (such as write error when saving stored key) can leave database file in inconsistent state (see Chapter 15 [Database consistency], page 25). When such a critical error occurs, the database file is marked as needing recovery. Subsequent calls to any `GDBM` functions for that database file (except `gdbm_recover`), will return immediately with `GDBM` error code `GDBM_NEED_RECOVERY`. Additionally, the following function can be used to check the state of the database file:

`int gdbm_needs_recovery` (*GDBM_FILE dbf*) [gdbm interface]
: Returns `1` if the database file *dbf* is in inconsistent state and needs recovery.

To restore structural consistency of the database, use the `gdbm_recover` function (see Chapter 16 [Recovery], page 26).

Crash tolerance provides a better way of recovery, because it restores both structural and logical consistency. See Chapter 17 [Crash Tolerance], page 28, for a detailed discussion,

15 Database consistency

In the chapters that follow we will cover different aspects of *database consistency* and ways to maintain it. Speaking about consistency, it is important to distinguish between two different aspects of it: structural and logical consistency.

Structural consistency means that all internal structures of the database are in good order, contain valid data and are coherent with one another. Structural consistency means that the database is in good shape *technically*, but it does not imply that the data it contains are in any way meaningful.

Logical consistency means that the data stored in the database are coherent with respect to the application logic. Usually this implies that structural consistency is observed as well.

For as long as the program is free from memory management errors and each opened database is properly closed before the program terminates, structural consistency is maintained. Maintaining logical consistency is more complex task and its maintenance is entirely the responsibility of the application programmer. See Chapter 17 [Crash Tolerance], page 28, for a detailed discussion.

Both consistency aspects can suffer as a result of both application errors that cause the program to terminate prematurely without properly saving the database, and hardware errors, such as disk failures or power outages. When such situations occur, it becomes necessary to *recover the database*.

In the next chapter we will discuss how to recover structural consistency of a database.

16 Recovering structural consistency

Certain errors (such as write error when saving stored key) can leave database file in *structurally inconsistent state*. When such a critical error occurs, the database file is marked as needing recovery. Subsequent calls to any GDBM functions for that database file (except `gdbm_recover`), will return immediately with `GDBM` error code `GDBM_NEED_RECOVERY`.

To escape from this state and bring the database back to operational state, use the following function:

`int gdbm_recover (`*GDBM_FILE* ***dbf***, *gdbm_recovery* ****rcvr***, *int* ***flags***`)` [gdbm interface]

Check the database file *dbf* and fix eventual errors. The *rcvr* argument points to a structure that has *input members*, providing additional information to alter the behavior of `gdbm_recover`, and *output members*, which are used to return additional statistics about the recovery process (*rcvr* can be `NULL` if no such information is needed).

Each input member has a corresponding flag bit, which must be set in *flags*, in order to instruct the function to use it.

The `gdbm_recover` type is defined as:

```
typedef struct gdbm_recovery_s
{
  /* Input members.
     These are initialized before call to gdbm_recover.
     The flags argument specifies which of them are initialized. */
  void (*errfun) (void *data, char const *fmt, ...);
  void *data;
  size_t max_failed_keys;
  size_t max_failed_buckets;
  size_t max_failures;

  /* Output members.
     The gdbm_recover function fills these before returning. */
  size_t recovered_keys;
  size_t recovered_buckets;
  size_t failed_keys;
  size_t failed_buckets;
  char *backup_name;
} gdbm_recovery;
```

The *input members* modify the behavior of `gdbm_recover`:

`void (*errfun) (`*void* ****data***, *char const* ****fmt***, *...*`)` [input member of `gdbm_recovery`]

If the `GDBM_RCVR_ERRFUN` flag bit is set, `errfun` points to a function that will be called upon each recoverable or non-fatal error that occurred during the recovery. The `data` field of `gdbm_recovery` will be passed to it as its first argument. The *fmt* argument is a `printf`-like (see Section "Format of the

format string" in *printf(3) man page*), format string. The rest of arguments supply parameters for that format.

`void * data` [input member of `gdbm_recovery`]
Supplies first argument for the `errfun` invocations.

`size_t max_failed_keys` [input member of `gdbm_recovery`]
If `GDBM_RCVR_MAX_FAILED_KEYS` is set, this member sets the limit on the number of keys that cannot be retrieved. If the number of failed keys becomes equal to `max_failed_keys`, recovery is aborted and error is returned.

`size_t max_failed_buckets` [input member of `gdbm_recovery`]
If `GDBM_RCVR_MAX_FAILED_BUCKETS` is set, this member sets the limit on the number of buckets that cannot be retrieved or that contain bogus information. If the number of failed buckets becomes equal to `max_failed_buckets`, recovery is aborted and error is returned.

`size_t max_failures` [output member of `gdbm_recovery`]
If `GDBM_RCVR_MAX_FAILURES` is set, this member sets the limit of failures that are tolerated during recovery. If the number of errors becomes equal to `max_failures`, recovery is aborted and error is returned.

The following members are filled on output, upon successful return from the function:

`size_t recovered_keys` [output member of `gdbm_recovery`]
Number of recovered keys.

`size_t recovered_buckets` [output member of `gdbm_recovery`]
Number of recovered buckets.

`size_t failed_keys` [output member of `gdbm_recovery`]
Number of key/data pairs that could not be retrieved.

`size_t failed_buckets` [output member of `gdbm_recovery`]
Number of buckets that could not be retrieved.

`char * backup_name` [output member of `gdbm_recovery`]
Name of the file keeping the copy of the original database, in the state prior to recovery. It is filled if the *GDBM_RCVR_BACKUP* flag is set. The string is allocated using the `malloc` call. The caller is responsible for freeing that memory when no longer needed.

By default, `gdbm_recovery` first checks the database for inconsistencies and attempts recovery only if some were found. The special flag bit `GDBM_RCVR_FORCE` instructs `gdbm_recovery` to omit this check and to perform database recovery unconditionally.

17 Crash Tolerance

Crash tolerance is a new (as of release 1.21) feature that can be enabled at compile time, and used in environments with appropriate support from the OS and the filesystem. As of version 1.22, this means a Linux kernel 5.12.12 or later and a filesystem that supports reflink copying, such as XFS, BtrFS, or OCFS2. If these prerequisites are met, crash tolerance code will be enabled automatically by the `configure` script when building the package.

The crash-tolerance mechanism, when used correctly, guarantees that a logically consistent (see Chapter 15 [Database consistency], page 25) recent state of application data can be recovered following a crash. Specifically, it guarantees that the state of the database file corresponding to the most recent successful `gdbm_sync` call can be recovered.

If the new mechanism is used correctly, crashes such as power outages, OS kernel panics, and (some) application process crashes will be tolerated. Non-tolerated failures include physical destruction of storage devices and corruption due to bugs in application logic. For example, the new mechanism won't help if a pointer bug in your application corrupts `GDBM`'s private in-memory data which in turn corrupts the database file.

In the following sections we will describe how to enable crash tolerance in your application and what to do if a crash occurs.

The design rationale of the crash tolerance mechanism is described in detail in the article, *Crashproofing the Original NoSQL Key-Value Store*, by Terence Kelly, *ACM Queue magazine*, July/August 2021, available from the ACM Digital Library. If you have difficulty retrieving this paper, please contact the author at `tpkelly@acm.org`, `tpkelly@cs.princeton.edu`, or `tpkelly@eecs.umich.edu`.

17.1 Using Proper Filesystem

Use a filesystem that supports reflink copying. Currently XFS, BtrFS, and OCFS2 support reflink. You can create such a filesystem if you don't have one already. (Note that reflink support may require that special options be specified at the time of filesystem creation; this is true of XFS.) The most conventional way to create a filesystem is on a dedicated storage device. However it is also possible to create a filesystem *within an ordinary file* on some other filesystem.

For example, the following commands, executed as root, will create a smallish XFS filesystem inside a file on another filesystem:

```
mkdir XFS
cd XFS
truncate --size 512m XFSfile
mkfs -t xfs -m crc=1 -m reflink=1 XFSfile
mkdir XFSmountpoint
mount -o loop XFSfile XFSmountpoint
```

The XFS filesystem is now available in directory `XFSmountpoint`. Now, create a directory where your unprivileged user account may create and delete files:

```
cd XFSmountpoint
mkdir test
chown user:group test
```

(where *user* and *group* are the user and group names of the unprivileged account the application uses).

Reflink copying via `ioctl(FICLONE)` should work for files in and below this directory. You can test reflink copying using the GNU `cp` program:

```
cp --reflink=always file1 file2
```

See Section "reflink" in GNU *Coreutils*.

Your GNU dbm database file and two *snapshot* files described below must all reside on the same reflink-capable filesystem.

17.2 Enabling crash tolerance

Open a GNU dbm database with `gdbm_open`. Whenever possible, use the extended `GDBM` format (see Section 17.8 [Numsync], page 32). Generally speaking, this means using the `GDBM_NUMSYNC` flag when creating the database. Unless you know what you are doing, do not specify the `GDBM_SYNC` flag when opening the database. The reason is that you want your application to explicitly control when `gdbm_sync` is called; you don't want an implicit sync on every database operation (see Chapter 11 [Sync], page 17).

Request crash tolerance by invoking the following interface:

```
int gdbm_failure_atomic (GDBM_FILE dbf, const char *even,
                         const char *odd);
```

The *even* and *odd* arguments are the pathnames of two files that will be created and filled with *snapshots* of the database file. These two files must not exist when `gdbm_failure_atomic` is called and must reside on the same reflink-capable filesystem as the database file.

After you call `gdbm_failure_atomic`, every call to `gdbm_sync` will make an efficient reflink snapshot of the database file in either the *even* or the *odd* snapshot file; consecutive `gdbm_sync` calls alternate between the two, hence the names. The permission bits and `mtime` timestamps on the snapshot files determine which one contains the state of the database file corresponding to the most recent successful `gdbm_sync`. See Section 17.4 [Crash recovery], page 30, for discussion of crash recovery.

17.3 Synchronizing the Database

When your application knows that the state of the database is consistent (i.e., all relevant application-level invariants hold), you may call `gdbm_sync`. For example, if your application manages bank accounts, transferring money from one account to another should maintain the invariant that the sum of the two accounts is the same before and after the transfer: It is correct to decrement account '`A`' by $7, increment account '`B`' by $7, and then call `gdbm_sync`. However it is *not* correct to call `gdbm_sync` *between* the decrement of '`A`' and the increment of '`B`', because a crash immediately after that call would destroy money. The general rule is simple, sensible, and memorable: Call `gdbm_sync` only when the database is in a state from which you are willing and able to recover following a crash. (If you think about it you'll realize that there's never any other moment when you'd really want to call `gdbm_sync`, regardless of whether crash-tolerance is enabled. Why on earth would you push the state of an inconsistent unrecoverable database down to durable media?).

17.4 Crash recovery

If a crash occurs, the snapshot file (*even* or *odd*) containing the database state reflecting the most recent successful `gdbm_sync` call is the snapshot file whose permission bits are read-only and whose last-modification timestamp is greatest. If both snapshot files are readable, we choose the one with the most recent last-modification timestamp. Modern operating systems record timestamps in nanoseconds, which gives sufficient confidence that the timestamps of the two snapshots will differ. However, one can't rule out the possibility that the two snapshot files will both be readable and have identical timestamps[1]. To cope with this, `GDBM` version 1.21 introduced the new *extended database format*, which stores in the database file header the number of synchronizations performed so far. This number can reliably be used to select the most recent snapshot, independently of its timestamp. We strongly suggest using this new format when writing crash-tolerant applications. See Section 17.8 [Numsync], page 32, for a detailed discussion.

The `gdbm_latest_snapshot` function is provided, that selects the right snapshot among the two. Invoke it as:

```
const char *recovery_file = NULL;
result = gdbm_latest_snapshot (even, odd, &recovery_file);
```

where *even* and *odd* are names of the snapshot files. On success, it stores the pointer to the most recent snapshot file name in *recovery_file* and returns `GDBM_SNAPSHOT_OK`. To finalize the recovery, rename this file to the name of your database file and re-open it using `gdbm_open`. You should discard the remaining snapshot.

If an error occurs, `gdbm_latest_snapshot` returns one of the following error codes.

`GDBM_SNAPSHOT_BAD` [gdbm_latest_snapshot]

Neither snapshot file is readable. This means that the crash has occurred before `gdbm_failure_atomic` completed. In this case, it is best to fall back on a safe backup copy of the data file.

`GDBM_SNAPSHOT_ERR` [gdbm_latest_snapshot]

System error occurred in `gdbm_latest_snapshot`. Examine the system `errno` variable for details. Its possible values are:

`EACCES`

The file mode of one of the snapshot files was incorrect. Each snapshot file can be either readable (0400) or writable (0200), but not both. This probably means that someone touched one or both snapshot files after the crash and before your attempt to recover from it. This case needs additional investigation. If you're sure that the only change someone made to the files is altering their modes, and your database is in *numsync* format (see Section 17.8 [Numsync], page 32), you can reset the modes to 0400 and retry the recovery.

This error can also be returned by underlying `stat` call, meaning that search permission was denied for one of the directories in the path prefix of a snapshot file name. That again means that someone has messed with permissions after the crash.

[1] This can happen, for example, if the storage is very fast and the system clock is low-resolution, or if the system administrator sets the system clock backwards. In the latter case one can end up with the most recent snapshot file having modification time earlier than that of the obsolete snapshot.

`EINVAL` Some arguments passed to `gdbm_latest_snapshot` were not valid. It is a programmer's error which means that your application needs to be fixed.

`ENOSYS` Function is not implemented. This means `GDBM` was built without crash-tolerance support.

`Other value (EBADF, EFAULT, etc)`
An error occurred when trying to `stat` the snapshot file. See Section "ERRORS" in *stat(2) man page*, for a discussion of possible `errno` values.

`GDBM_SNAPSHOT_SAME` [gdbm_latest_snapshot]
File modes and modification dates of both snapshot files are exactly the same. This can happen only if numsync is not available (see Section 17.8 [Numsync], page 32).

`GDBM_SNAPSHOT_SUSPICIOUS` [gdbm_latest_snapshot]
For the database in extended *numsync* format (see Section 17.8 [Numsync], page 32): the `numsync` values of the two snapshot differ by more than one. Check the arguments to the `gdbm_latest_snapshot` function. The most probably reason of such an error is that the *even* and *odd* parameters point to snapshot files belonging to different database files.

If you get any of these errors, we strongly suggest to undertake *manual recovery*.

17.5 Manual crash recovery

Manual recovery is usually performed with the help of the `gdbmtool` utility. Start `gdbmtool` in read-only mode (the `-r`) option. Once in the command shell, issue the following command:

```
snapshot a b
```

where *a* and *b* are names of the two snapshot files you configured using the `gdbm_failure_atomic` function. This command investigates both files and prints out detailed diagnostics.

Its output begins with a line listing one of the error codes above, followed by a colon and a textual description of the error. The lines that follow show details for each snapshot file.

Each snapshot description begins with the snapshot file name followed by a colon and four fields, in this order:

1. File permission bits in octal.
2. File permission bits in `ls -l` notation.
3. Modification timestamp.
4. Numsync counter. For databases in standard `GDBM` format, this field is '`N/A`'. If the counter cannot be obtained because of error, this field is '`?`'.

Any errors or inconsistencies discovered are reported in the lines that follow, one error per line. Here's an example of the `snapshot` command output, describing the `GDBM_SNAPSHOT_ERR` condition:

```
gdbmtool> snapshot even.dbf odd.dbf
GDBM_SNAPSHOT_ERR: Error selecting snapshot.
even.dbf: 200 -w------- 1627820627.485681330 ?
odd.dbf: 600 rw------- 1627820627.689503918 301
odd.dbf: ERROR: bad file mode
```

Line 2 lists the meta-data of the snapshot `even.dbf`. The `numsync` field contains question mark because the file permissions (write-only) prevented `gdbmtool` from opening it.

The lines for `odd.dbf` show the actual reason for the error: bad file mode (read-write). Apparently, the file mode has been changed manually after the crash. The timestamp of the file, which is more recent than that of `even.dbf`, suggests that it might be used for recovery. To confirm this guess, change the mode of the `even.dbf` to read-only and repeat the `snapshot` command:

```
gdbmtool> ! chmod 400 even.dbf
gdbmtool> snapshot even.dbf odd.dbf
GDBM_SNAPSHOT_ERR: Error selecting snapshot.
even.dbf: 400 r-------- 1627820627.485681330 300
odd.dbf: 600 rw------- 1627820627.689503918 301
odd.dbf: ERROR: bad file mode
```

This shows the numsync value of the `even.dbf` file, which is exactly one less than that of `odd.dbf`. This means that the latter should be selected for recovery.

For completeness sake, you can change the mode of `odd.dbf` to read-only as well and repeat the `snapshot` command. In this case you will see:

```
gdbmtool> ! chmod 400 odd.dbf
gdbmtool> snapshot even.dbf odd.dbf
GDBM_SNAPSHOT_OK: Selected the most recent snapshot.
odd.dbf: 400 r-------- 1627820627.689503918 301
```

17.6 Performance Impact

The purpose of a parachute is not to hasten descent. Crash tolerance is a safety mechanism, not a performance accelerator. Reflink copying is designed to be as efficient as possible, but making snapshots of the GNU dbm database file on every `gdbm_sync` call entails overheads. The performance impact of `GDBM` crash tolerance will depend on many factors including the type and configuration of the underlying storage system, how often the application calls `gdbm_sync`, and the extent of changes to the database file between consecutive calls to `gdbm_sync`.

17.7 Availability

To ensure that application data can survive the failure of one or more storage devices, replicated storage (e.g., RAID) may be used beneath the reflink-capable filesystem. Some cloud providers offer block storage services that mimic the interface of individual storage devices but that are implemented as high-availability fault-tolerant replicated distributed storage systems. Installing a reflink-capable filesystem atop a high-availability storage system is a good starting point for a high-availability crash-tolerant `GDBM`.

17.8 Numsync Extension

In Section 17.4 [Crash recovery], page 30, we have shown that for database recovery, one should select the snapshot whose permission bits are read-only and whose last-modification timestamp is greatest. However, there may be cases when a crash occurs at such a time that both snapshot files remain readable. It may also happen, that their permissions had

been reset to read-only and/or modification times inadvertently changed before recovery. To make it possible to select the right snapshot in such cases, a new *extended database format* was introduced in GDBM version 1.21. This format adds to the database header the `numsync` field, which holds the number of synchronizations the database underwent before being closed or abandoned due to a crash.

A readable snapshot is a consistent copy of the database at a given point of time. Thus, if both snapshots of a database in extended format are readable, it will suffice to examine their `numsync` counters and select the one whose `numsync` is greater. That's what the `gdbm_latest_snapshot` function does in this case.

It is worth noticing, that the two counters should differ exactly by one. If the difference is greater than that, `gdbm_latest_snapshot` will return a special status code, `GDBM_SNAPSHOT_SUSPICIOUS`. If, during a recovery attempt, you get this status code, we recommend to proceed with the manual recovery (see Section 17.5 [Manual crash recovery], page 31).

To create a database in extended format, call `gdbm_open` with both `GDBM_NEWDB` and `GDBM_NUMSYNC` flags:

```
dbf = gdbm_open(dbfile, 0, GDBM_NEWDB|GDBM_NUMSYNC, 0600, NULL);
```

Notice, that this flag must always be used together with `GDBM_NEWDB` (see Chapter 3 [Open], page 5). It is silently ignored when used together with another opening flag.

A standard `GDBM` database can be converted to the extended format and vice versa. To convert an existing database to the extended format, use the `gdbm_convert` function (see Chapter 12 [Database format], page 18):

```
rc = gdbm_convert(dbf, GDBM_NUMSYNC);
```

You can do the same using the `gdbmtool` utility (see Section 24.2.2 [commands], page 59):

```
gdbmtool dbname upgrade
```

To convert a database from extended format back to the standard `GDBM` format, do:

```
rc = gdbm_convert(dbf, 0);
```

To do the same from the command line, run:

```
gdbmtool dbname downgrade
```

17.9 Crash Tolerance API

`int gdbm_failure_atomic` (*GDBM_FILE* ***dbf***, *const char* ***even***, *const char* ***odd***) [gdbm interface]

Enables crash tolerance for the database file *dbf*. The *even* and *odd* arguments are the pathnames of two files that will be created and filled with snapshots of the database file. These two files must not exist when `gdbm_failure_atomic` is called and must reside on the same reflink-capable filesystem as the database file.

Returns 0 on success. On failure, returns -1 and sets `gdbm_errno` to one of the following values:

`GDBM_ERR_USAGE`
: Improper function usage. Either *even* or *odd* is `NULL`, or they point to the same string.

`GDBM_NEED_RECOVERY`
: The database needs recovery. See Chapter 16 [Recovery], page 26.

`GDBM_ERR_SNAPSHOT_CLONE`
: Failed to clone the database file into a snapshot. Examine the system `errno` variable for details.

If one of the following error codes is returned, examine the system `errno` variable for details:

`GDBM_ERR_REALPATH`
: Call to `realpath` function failed. `realpath` is used to determine actual path names of the snapshot files.

`GDBM_FILE_OPEN_ERROR`
: Unable to create snapshot file.

`GDBM_FILE_SYNC_ERROR`
: Failed to sync a snapshot file or one of directories in its pathname, during initial synchronization.

`GDBM_FILE_CLOSE_ERROR`
: Failed to close a snapshot file or one of directories in its pathname, during initial synchronization.

`GDBM_ERR_FILE_MODE`
: The `fchmod` call on one of the snapshot files failed.

Notes:

- It is not an error to call `gdbm_failure_atomic` several times. Each subsequent call closes the previously configured snapshot files and installs new ones instead.
- Crash tolerance settings are cleared by functions `gdbm_recover` (see Chapter 16 [Recovery], page 26) and `gdbm_reorganize` (see Chapter 10 [Reorganization], page 16). In case of `gdbm_recover`, it should not be a problem, because if you enabled crash tolerance, the procedure described in Section 17.4 [Crash recovery], page 30 is the preferred way of recovering the database. If, however, you decided to call either function even though you had enabled crash tolerance previously, be sure to call `gdbm_failure_atomic` again with the same arguments as before (provided that the call returns successfully).

`int gdbm_latest_snapshot` (*const char *`even`, const char *`odd`, const char **`retval`*) [gdbm interface]

Selects between two snapshots, *even* and *odd*, the one to be used for crash recovery. On success, stores a pointer to the selected filename in the memory location pointed to by *retval* and returns `GDBM_SNAPSHOT_OK`. If neither snapshot file is usable, the function returns `GDBM_SNAPSHOT_BAD`. If a system error occurs, it returns `GDBM_SNAPSHOT_ERR` and sets `errno` to the error code describing the problem. Finally, in the unlikely case that it cannot select between the two snapshots (this means they are both readable and have exactly the same `mtime` timestamp), the function returns `GDBM_SNAPSHOT_SAME`.

If the ‘`numsync`’ extension is enabled (see Section 17.8 [Numsync], page 32), the function can also return the `GDBM_SNAPSHOT_SUSPICIOUS` status code. This happens when the `numsync` counters in the two snapshots differ by more than one.

See Section 17.4 [Crash recovery], page 30, for a detailed description of possible return codes and their interpretation.

If any value other than `GDBM_SNAPSHOT_OK` is returned, it is guaranteed that the function did not touch *retval*. In this case it is recommended to switch to manual recovery procedure, letting the user examine the snapshots and take the appropriate action. see Section 17.5 [Manual crash recovery], page 31, for details.

18 Setting options

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

`int gdbm_setopt (`*GDBM_FILE* `dbf`, *int* `option`, *void* *`value`, *int* `size``)` [gdbm interface]

Sets an option on the database or returns the value of an option.

The parameters are:

dbf The pointer returned by `gdbm_open`.

option The option to be set or retrieved.

value A pointer to the value to which *option* will be set or where to place the option value (depending on the option).

size The length of the data pointed to by *value*.

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

The valid options are:

GDBM_SETCACHESIZE
GDBM_CACHESIZE

Set the size of the internal bucket cache. The *value* should point to a `size_t` holding the desired cache size, or the constant `GDBM_CACHE_AUTO`, to set the best cache size automatically.

By default, a newly open database is configured to adapt the cache size to the number of index buckets in the database file. This provides for the best performance.

Use this option if you wish to limit the memory usage at the expense of performance. If you chose to do so, please bear in mind that cache becomes effective when its size is greater then 2/3 of the number of index bucket counts in the database. The best performance results are achieved when cache size equals the number of buckets. For example:

```
size_t bn;
gdbm_bucket_count (dbf, &bn);
ret = gdbm_setopt (dbf, GDBM_SETCACHESIZE, &bn, sizeof (bn));
```

To set the best cache size, use the constant `GDBM_CACHE_AUTO`:

```
size_t bn = GDBM_CACHE_AUTO;
ret = gdbm_setopt (dbf, GDBM_SETCACHESIZE, &bn, sizeof (bn));
```

GDBM_GETCACHESIZE

Return the size of the internal bucket cache. The *value* should point to a `size_t` variable, where the size will be stored.

GDBM_GETFLAGS

Return the flags describing the state of the database. The *value* should point to an `int` variable where to store the flags. On success, its value will be similar

to the flags used when opening the database (see Chapter 3 [Open], page 5), except that it will reflect the current state (which may have been altered by another calls to `gdbm_setopt`).

GDBM_FASTMODE
: Enable or disable the *fast writes mode*, i.e. writes without subsequent synchronization. The *value* should point to an integer: `TRUE` to enable fast mode, and `FALSE` to disable it.

  This option is retained for compatibility with previous versions of `GDBM`. Its effect is the reverse of `GDBM_SETSYNCMODE` (see below).

GDBM_SETSYNCMODE
GDBM_SYNCMODE
: Turn on or off file system synchronization operations. This setting defaults to off. The *value* should point to an integer: `TRUE` to turn synchronization on, and `FALSE` to turn it off.

  Note, that this option is a reverse of `GDBM_FASTMODE`, i.e. calling `GDBM_SETSYNCMODE` with `TRUE` has the same effect as calling `GDBM_FASTMODE` with `FALSE`.

  The `GDBM_SYNCMODE` option is provided for compatibility with earlier versions.

GDBM_GETSYNCMODE
: Return the current synchronization status. The *value* should point to an `int` where the status will be stored.

GDBM_SETCENTFREE
GDBM_CENTFREE
: *NOTICE: This feature is still under study.*

  Set central free block pool to either on or off. The default is off, which is how previous versions of `GDBM` handled free blocks. If set, this option causes all subsequent free blocks to be placed in the *global* pool, allowing (in theory) more file space to be reused more quickly. The *value* should point to an integer: `TRUE` to turn central block pool on, and `FALSE` to turn it off.

  The `GDBM_CENTFREE` option is provided for compatibility with earlier versions.

GDBM_SETCOALESCEBLKS
GDBM_COALESCEBLKS
: *NOTICE: This feature is still under study.*

  Set free block merging to either on or off. The default is off, which is how previous versions of `GDBM` handled free blocks. If set, this option causes adjacent free blocks to be merged. This can become a CPU expensive process with time, though, especially if used in conjunction with GDBM_CENTFREE. The *value* should point to an integer: `TRUE` to turn free block merging on, and `FALSE` to turn it off.

GDBM_GETCOALESCEBLKS
: Return the current status of free block merging. The *value* should point to an `int` where the status will be stored.

GDBM_SETMAXMAPSIZE
: Sets maximum size of a memory mapped region. The *value* should point to a value of type `size_t`, `unsigned long` or `unsigned`. The actual value is rounded to the nearest page boundary (the page size is obtained from `sysconf(_SC_PAGESIZE)`).

GDBM_GETMAXMAPSIZE
: Return the maximum size of a memory mapped region. The *value* should point to a value of type `size_t` where to return the data.

GDBM_SETMMAP
: Enable or disable memory mapping mode. The *value* should point to an integer: `TRUE` to enable memory mapping or `FALSE` to disable it.

GDBM_GETMMAP
: Check whether memory mapping is enabled. The *value* should point to an integer where to return the status.

GDBM_GETDBNAME
: Return the name of the database disk file. The *value* should point to a variable of type `char**`. A pointer to the newly allocated copy of the file name will be placed there. The caller is responsible for freeing this memory when no longer needed. For example:

```
char *name;

if (gdbm_setopt (dbf, GDBM_GETDBNAME, &name, sizeof (name)))
  {
     fprintf (stderr, "gdbm_setopt failed: %s\n",
              gdbm_strerror (gdbm_errno));
  }
else
  {
    printf ("database name: %s\n", name);
    free (name);
  }
```

GDBM_GETBLOCKSIZE
: Return the block size in bytes. The *value* should point to `int`.

19 File Locking

With locking disabled (if `gdbm_open` was called with `GDBM_NOLOCK`), the user may want to perform their own file locking on the database file in order to prevent multiple writers operating on the same file simultaneously.

In order to support this, the `gdbm_fdesc` routine is provided.

`int gdbm_fdesc (`*GDBM_FILE* `dbf``)` [gdbm interface]

Returns the file descriptor of the database *dbf*. This value can be used as an argument to `flock`, `lockf` or similar calls.

20 Useful global variables

The following global variables and constants are available:

`gdbm_error gdbm_errno` [Variable]
: This variable contains error code from the last failed `GDBM` call. See Chapter 22 [Error codes], page 43, for a list of available error codes and their descriptions.

  Use `gdbm_strerror` (see Chapter 14 [Errors], page 23) to convert it to a descriptive text.

`const char * gdbm_errlist[]` [Variable]
: This variable is an array of error descriptions, which is used by `gdbm_strerror` to convert error codes to human-readable text (see Chapter 14 [Errors], page 23). You can access it directly, if you wish so. It contains `_GDBM_MAX_ERRNO + 1` elements and can be directly indexed by the error code to obtain a corresponding descriptive text.

`int const gdbm_syserr[]` [Variable]
: Array of boolean values indicating, for each `GDBM` error code, whether the value of `errno`(3) variable is meaningful for this error code. See [gdbm_check_syserr], page 23.

`_GDBM_MIN_ERRNO` [Constant]
: The minimum error code used by `GDBM`.

`_GDBM_MAX_ERRNO` [Constant]
: The maximum error code used by `GDBM`.

`const char * gdbm_version` [Variable]
: A string containing the version information.

`int const gdbm_version_number[3]` [Variable]
: This variable contains the `GDBM` version numbers:

Index	**Meaning**
0	Major number
1	Minor number
2	Patchlevel number

  Additionally, the following constants are defined in the `gdbm.h` file:

  GDBM_VERSION_MAJOR
  : Major number.

  GDBM_VERSION_MINOR
  : Minor number.

  GDBM_VERSION_PATCH
  : Patchlevel number.

  These can be used to verify whether the header file matches the library.

To compare two split-out version numbers, use the following function:

`int gdbm_version_cmp` (*int const* `a`[*3*], *int const* `b`[*3*]) [gdbm interface]

Compare two version numbers. Return `-1` if *a* is less than *b*, `1` if *a* is greater than *b* and `0` if they are equal.

Comparison is done from left to right, so that:

```
a = { 1, 8, 3 };
b = { 1, 8, 3 };
gdbm_version_cmp (a, b) ⇒ 0

a = { 1, 8, 3 };
b = { 1, 8, 2 };
gdbm_version_cmp (a, b) ⇒ 1

a = { 1, 8, 3 };
b = { 1, 9. 0 };
gdbm_version_cmp (a, b) ⇒ -1
```

21 Additional functions

`int gdbm_avail_verify` (*GDBM_FILE* ***dbf***) [gdbm interface]

Verify if the available block stack is in consistent state. On success, returns 0. If any errors are encountered, sets the `gdbm_errno` to `GDBM_BAD_AVAIL`, marks the database as needing recovery (see Chapter 16 [Recovery], page 26) and return -1.

22 Error codes

This chapter summarizes error codes which can be set by the functions in GDBM library.

`GDBM_NO_ERROR` [Error Code]
No error occurred.

`GDBM_MALLOC_ERROR` [Error Code]
Memory allocation failed. Not enough memory.

`GDBM_BLOCK_SIZE_ERROR` [Error Code]
This error is set by the `gdbm_open` function (see Chapter 3 [Open], page 5), if the value of its *block_size* argument is incorrect and the `GDBM_BSEXACT` flag is set.

`GDBM_FILE_OPEN_ERROR` [Error Code]
The library was not able to open a disk file. This can be set by `gdbm_open` (see Chapter 3 [Open], page 5), `gdbm_dump` (`gdbm_export`) and `gdbm_load` (`gdbm_import`) functions (see Chapter 13 [Flat files], page 19).

Inspect the value of the system `errno` variable to get more detailed diagnostics.

`GDBM_FILE_WRITE_ERROR` [Error Code]
Writing to a disk file failed. This can be set by `gdbm_open` (see Chapter 3 [Open], page 5), `gdbm_dump` (`gdbm_export`) and `gdbm_load` (`gdbm_import`) functions.

Inspect the value of the system `errno` variable to get more detailed diagnostics.

`GDBM_FILE_SEEK_ERROR` [Error Code]
Positioning in a disk file failed. This can be set by `gdbm_open` (see Chapter 3 [Open], page 5) function.

Inspect the value of the system `errno` variable to get a more detailed diagnostics.

`GDBM_FILE_READ_ERROR` [Error Code]
Reading from a disk file failed. This can be set by `gdbm_open` (see Chapter 3 [Open], page 5), `gdbm_dump` (`gdbm_export`) and `gdbm_load` (`gdbm_import`) functions.

Inspect the value of the system `errno` variable to get a more detailed diagnostics.

`GDBM_BAD_MAGIC_NUMBER` [Error Code]
The file given as argument to `gdbm_open` function is not a valid `GDBM` file: it has a wrong magic number.

`GDBM_EMPTY_DATABASE` [Error Code]
The file given as argument to `gdbm_open` function is not a valid `GDBM` file: it has zero length.

`GDBM_CANT_BE_READER` [Error Code]
This error code is set by the `gdbm_open` function if it is not able to lock file when called in `GDBM_READER` mode (see Chapter 3 [Open], page 5).

`GDBM_CANT_BE_WRITER` [Error Code]
This error code is set by the `gdbm_open` function if it is not able to lock file when called in writer mode (see Chapter 3 [Open], page 5).

`GDBM_READER_CANT_DELETE` [Error Code]
: Set by the `gdbm_delete` (see Chapter 8 [Delete], page 13) if it attempted to operate on a database that is open in read-only mode (see Chapter 3 [Open], page 5).

`GDBM_READER_CANT_STORE` [Error Code]
: Set by the `gdbm_store` (see Chapter 6 [Store], page 10) if it attempted to operate on a database that is open in read-only mode (see Chapter 3 [Open], page 5).

`GDBM_READER_CANT_REORGANIZE` [Error Code]
: Set by the `gdbm_reorganize` (see Chapter 10 [Reorganization], page 16) if it attempted to operate on a database that is open in read-only mode (see Chapter 3 [Open], page 5).

`GDBM_ITEM_NOT_FOUND` [Error Code]
: Requested item was not found. This error is set by `gdbm_delete` (see Chapter 8 [Delete], page 13) and `gdbm_fetch` (see Chapter 7 [Fetch], page 12) when the requested *key* value is not found in the database.

`GDBM_REORGANIZE_FAILED` [Error Code]
: The `gdbm_reorganize` function is not able to create a temporary database. See Chapter 10 [Reorganization], page 16.

`GDBM_CANNOT_REPLACE` [Error Code]
: Cannot replace existing item. This error is set by the `gdbm_store` if the requested *key* value is found in the database and the *flag* parameter is not `GDBM_REPLACE`. See Chapter 6 [Store], page 10, for a detailed discussion.

`GDBM_MALFORMED_DATA` [Error Code]
`GDBM_ILLEGAL_DATA` [Error Code]
: Input data was malformed in some way. When returned by `gdbm_load`, this means that the input file was not a valid `GDBM` dump file (see [gdbm_load function], page 20). When returned by `gdbm_store`, this means that either *key* or *content* parameter had its `dptr` field set to `NULL` (see Chapter 6 [Store], page 10).

  The `GDBM_ILLEGAL_DATA` is an alias for this error code, maintained for backward compatibility. Its use in modern applications is discouraged.

`GDBM_OPT_ALREADY_SET` [Error Code]
: Requested option can be set only once and was already set. As of version 1.22, this error code is no longer used. In prior versions it could have been returned by the `gdbm_setopt` function when setting the `GDBM_CACHESIZE` value.

`GDBM_OPT_BADVAL` [Error Code]
`GDBM_OPT_ILLEGAL` [Error Code]
: The *option* argument is not valid or the *value* argument points to an invalid value in a call to `gdbm_setopt` function. See Chapter 18 [Options], page 36.

  `GDBM_OPT_ILLEGAL` is an alias for this error code, maintained for backward compatibility. Modern applications should not use it.

`GDBM_BYTE_SWAPPED` [Error Code]
: The `gdbm_open` function (see Chapter 3 [Open], page 5) attempts to open a database which is created on a machine with different byte ordering.

`GDBM_BAD_FILE_OFFSET` [Error Code]
: The `gdbm_open` function (see Chapter 3 [Open], page 5) sets this error code if the file it tries to open has a wrong magic number.

`GDBM_BAD_OPEN_FLAGS` [Error Code]
: Set by the `gdbm_dump` (`gdbm_export`) function if supplied an invalid *flags* argument. See Chapter 13 [Flat files], page 19.

`GDBM_FILE_STAT_ERROR` [Error Code]
: Getting information about a disk file failed. The system `errno` will give more details about the error.

  This error can be set by the following functions: `gdbm_open`, `gdbm_reorganize`.

`GDBM_FILE_EOF` [Error Code]
: End of file was encountered where more data was expected to be present. This error can occur when fetching data from the database and usually means that the database is truncated or otherwise corrupted.

  This error can be set by any `GDBM` function that does I/O. Some of these functions are: `gdbm_delete`, `gdbm_exists`, `gdbm_fetch`, `gdbm_dump`, `gdbm_load`, `gdbm_export`, `gdbm_import`, `gdbm_reorganize`, `gdbm_firstkey`, `gdbm_nextkey`, `gdbm_store`.

`GDBM_NO_DBNAME` [Error Code]
: Output database name is not specified. This error code is set by `gdbm_load` (see [gdbm_load], page 20) if the first argument points to `NULL` and the input file does not specify the database name.

`GDBM_ERR_FILE_OWNER` [Error Code]
: This error code is set by `gdbm_load` if it is unable to restore database file owner. It is a mild error condition, meaning that the data have been restored successfully, only changing the target file owner failed. Inspect the system `errno` variable to get a more detailed diagnostics.

`GDBM_ERR_FILE_MODE` [Error Code]
: This error code is set by `gdbm_load` if it is unable to restore database file mode. It is a mild error condition, meaning that the data have been restored successfully, only changing the target file owner failed. Inspect the system `errno` variable to get a more detailed diagnostics.

`GDBM_NEED_RECOVERY` [Error Code]
: Database is in inconsistent state and needs recovery. Call `gdbm_recover` if you get this error. See Chapter 16 [Recovery], page 26, for a detailed description of recovery functions.

`GDBM_BACKUP_FAILED` [Error Code]
: The GDBM engine is unable to create backup copy of the file.

`GDBM_DIR_OVERFLOW` [Error Code]
: Bucket directory would overflow the size limit during an attempt to split hash bucket. This error can occur while storing a new key.

`GDBM_BAD_BUCKET` [Error Code]
Invalid index bucket is encountered in the database. Database recovery is needed (see Chapter 16 [Recovery], page 26).

`GDBM_BAD_HEADER` [Error Code]
This error is set by `gdbm_open` and `gdbm_fd_open`, if the first block read from the database file does not contain a valid `GDBM` header.

`GDBM_BAD_AVAIL` [Error Code]
The available space stack is invalid. This error can be set by `gdbm_open` and `gdbm_fd_open`, if the extended database verification was requested (`GDBM_XVERIFY`). It is also set by the `gdbm_avail_verify` function (see Chapter 21 [Additional functions], page 42).

Database recovery is needed (see Chapter 16 [Recovery], page 26).

`GDBM_BAD_HASH_TABLE` [Error Code]
Hash table in a bucket is invalid. This error can be set by the following functions: `gdbm_delete`, `gdbm_exists`, `gdbm_fetch`, `gdbm_firstkey`, `gdbm_nextkey`, and `gdbm_store`.

Database recovery is needed (see Chapter 16 [Recovery], page 26).

`GDBM_BAD_DIR_ENTRY` [Error Code]
Bad directory entry found in the bucket. The database recovery is needed (see Chapter 16 [Recovery], page 26).

`GDBM_FILE_CLOSE_ERROR` [Error Code]
The `gdbm_close` function was unable to close the database file descriptor. The system `errno` variable contains the corresponding error code.

`GDBM_FILE_SYNC_ERROR` [Error Code]
Cached content couldn't be synchronized to disk. Examine the `errno` variable to get more info,

Database recovery is needed (see Chapter 16 [Recovery], page 26).

`GDBM_FILE_TRUNCATE_ERROR` [Error Code]
File cannot be truncated. Examine the `errno` variable to get more info.

This error is set by `gdbm_open` and `gdbm_fd_open` when called with the `GDBM_NEWDB` flag.

`GDBM_BUCKET_CACHE_CORRUPTED` [Error Code]
The bucket cache structure is corrupted. Database recovery is needed (see Chapter 16 [Recovery], page 26).

`GDBM_BAD_HASH_ENTRY` [Error Code]
This error is set during sequential access (see Chapter 9 [Sequential], page 14), if the next hash table entry does not contain the expected key. This means that the bucket is malformed or corrupted and the database needs recovery (see Chapter 16 [Recovery], page 26).

`GDBM_ERR_SNAPSHOT_CLONE` [Error Code]

Set by the `gdbm_failure_atomic` function if it was unable to clone the database file into a snapshot. Inspect the system `errno` variable for the underlying cause of the error. If `errno` is `EINVAL` or `ENOSYS`, crash tolerance settings will be removed from the database.

See Section 17.9 [Crash Tolerance API], page 33.

`GDBM_ERR_REALPATH` [Error Code]

Set by the `gdbm_failure_atomic` function if the call to `realpath` function failed. `realpath` is used to determine actual path names of the snapshot files. Examine the system `errno` variable for details.

See Section 17.9 [Crash Tolerance API], page 33.

`GDBM_ERR_USAGE` [Error Code]

Function usage error. That includes invalid argument values, and the like.

23 Compatibility with standard dbm and ndbm

Gdbm includes a compatibility layer, which provides traditional ndbm and older dbm functions. The layer is compiled and installed if the --enable-libgdbm-compat option is used when configuring the package.

The compatibility layer consists of two header files: ndbm.h and dbm.h and the libgdbm_compat library.

Older programs using ndbm or dbm interfaces can use libgdbm_compat without any changes. To link a program with the compatibility library, add the following two options to the cc invocation: -lgdbm -lgdbm_compat. The -L option may also be required, depending on where GDBM is installed, e.g.:

```
cc ... -lgdbm -lgdbm_compat
```

Databases created and manipulated by the compatibility interfaces consist of two different files: *file*.dir and *file*.pag. This is required by the POSIX specification and corresponds to the traditional usage. Note, however, that despite the similarity of the naming convention, actual data stored in these files has not the same format as in the databases created by other dbm or ndbm libraries. In other words, you cannot access a standard UNIX dbm file with GNU dbm!

Compatibility interface includes only functions required by POSIX (see Section 23.1 [ndbm], page 48) or present in the traditional DBM implementation (see Section 23.2 [dbm], page 50). Advanced GDBM features, such as crash tolerance, cannot be used with such databases.

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

23.1 NDBM interface functions

The functions below implement the POSIX ndbm interface:

DBM * dbm_open (*char *file*, *int flags*, *int mode*) [ndbm]

Opens a database. The *file* argument is the full name of the database file to be opened. The function opens two files: *file*.pag and *file*.dir. The *flags* and *mode* arguments have the same meaning as the second and third arguments of open (see Section "open" in *open(2) man page*), except that a database opened for write-only access opens the files for read and write access and the behavior of the O_APPEND flag is unspecified.

The function returns a pointer to the DBM structure describing the database. This pointer is used to refer to this database in all operations described below.

Any error detected will cause a return value of NULL and an appropriate value will be stored in gdbm_errno (see Chapter 20 [Variables], page 40).

void dbm_close (*DBM *dbf*) [ndbm]

Closes the database. The *dbf* argument must be a pointer returned by an earlier call to dbm_open.

`datum dbm_fetch` (*DBM *dbf*, *datum key*) [ndbm]

Reads a record from the database with the matching key. The *key* argument supplies the key that is being looked for.

If no matching record is found, the `dptr` member of the returned datum is `NULL`. Otherwise, the `dptr` member of the returned datum points to the memory managed by the compatibility library. The application should never free it.

`int dbm_store` (*DBM *dbf*, *datum key*, *datum content*, *int mode*) [ndbm]

Writes a key/value pair to the database. The argument *dbf* is a pointer to the `DBM` structure returned from a call to `dbm_open`. The *key* and *content* provide the values for the record key and content. The *mode* argument controls the behavior of `dbm_store` in case a matching record already exists in the database. It can have one of the following two values:

`DBM_REPLACE`
: Replace existing record with the new one.

`DBM_INSERT`
: The existing record is left unchanged, and the function returns `1`.

If no matching record exists in the database, new record will be inserted no matter what the value of the *mode* is.

`int dbm_delete` (*DBM *dbf*, *datum key*) [ndbm]

Deletes the record with the matching key from the database. If the function succeeds, `0` is returned. Otherwise, if no matching record is found or if an error occurs, `-1` is returned.

`datum dbm_firstkey` (*DBM *dbf*) [ndbm]

Initializes iteration over the keys from the database and returns the first key. Note, that the word '`first`' does not imply any specific ordering of the keys.

If there are no records in the database, the `dptr` member of the returned datum is `NULL`. Otherwise, the `dptr` member of the returned datum points to the memory managed by the compatibility library. The application should never free it.

`datum dbm_nextkey` (*DBM *dbf*) [ndbm]

Continues the iteration started by `dbm_firstkey`. Returns the next key in the database. If the iteration covered all keys in the database, the `dptr` member of the returned datum is `NULL`. Otherwise, the `dptr` member of the returned datum points to the memory managed by the compatibility library. The application should never free it.

The usual way of iterating over all the records in the database is:

```
for (key = dbm_firstkey (dbf); key.ptr; key = dbm_nextkey (dbf))
  {
    /* do something with the key */
  }
```

The loop above should not try to delete any records from the database, otherwise the iteration is not guaranteed to cover all the keys. See Chapter 9 [Sequential], page 14, for a detailed discussion of this.

`int dbm_error` (*DBM *dbf*) [ndbm]
: Returns the error condition of the database: `0` if no errors occurred so far while manipulating the database, and a non-zero value otherwise.

`void dbm_clearerr` (*DBM *dbf*) [ndbm]
: Clears the error condition of the database.

`int dbm_dirfno` (*DBM *dbf*) [ndbm]
: Returns the file descriptor of the '`dir`' file of the database. It is guaranteed to be different from the descriptor returned by the `dbm_pagfno` function (see below).

  The application can lock this descriptor to serialize accesses to the database.

`int dbm_pagfno` (*DBM *dbf*) [ndbm]
: Returns the file descriptor of the '`pag`' file of the database. See also `dbm_dirfno`.

`int dbm_rdonly` (*DBM *dbf*) [ndbm]
: Returns `1` if the database *dbf* is open in a read-only mode and `0` otherwise.

23.2 DBM interface functions

The functions below are provided for compatibility with the old UNIX '`DBM`' interface. Only one database at a time can be manipulated using them.

`int dbminit` (*char *file*) [dbm]
: Opens a database. The *file* argument is the full name of the database file to be opened. The function opens two files: *`file`*`.pag` and *`file`*`.dir`. If any of them does not exist, the function fails. It never attempts to create the files.

  The database is opened in the read-write mode, if its disk permissions permit.

  The application must ensure that the functions described below in this section are called only after a successful call to `dbminit`.

`int dbmclose` (*void*) [dbm]
: Closes the database opened by an earlier call to `dbminit`.

`datum fetch` (*datum key*) [dbm]
: Reads a record from the database with the matching key. The *key* argument supplies the key that is being looked for.

  If no matching record is found, the `dptr` member of the returned datum is `NULL`. Otherwise, the `dptr` member of the returned datum points to the memory managed by the compatibility library. The application should never free it.

`int store` (*datum key*, *datum content*) [dbm]
: Stores the key/value pair in the database. If a record with the matching key already exists, its content will be replaced with the new one.

  Returns `0` on success and `-1` on error.

`int delete` (*datum key*) [dbm]
: Deletes a record with the matching key.

  If the function succeeds, `0` is returned. Otherwise, if no matching record is found or if an error occurs, `-1` is returned.

`datum firstkey` (*void*) [dbm]

Initializes iteration over the keys from the database and returns the first key. Note, that the word '`first`' does not imply any specific ordering of the keys.

If there are no records in the database, the `dptr` member of the returned datum is `NULL`. Otherwise, the `dptr` member of the returned datum points to the memory managed by the compatibility library. The application should never free it.

`datum nextkey` (*datum key*) [dbm]

Continues the iteration started by a call to `firstkey`. Returns the next key in the database. If the iteration covered all keys in the database, the `dptr` member of the returned datum is `NULL`. Otherwise, the `dptr` member of the returned datum points to the memory managed by the compatibility library. The application should never free it.

24 Examine and modify a GDBM database

The `gdbmtool` utility allows you to view and modify an existing `GDBM` database or to create a new one.

When invoked without arguments, it tries to open a database file called `junk.gdbm`, located in the current working directory. You can change this default by supplying the name of the database as argument to the program, e.g.:

```
$ gdbmtool file.db
```

The database will be opened in read-write mode, unless the `-r` (`--read-only`) option is specified, in which case it will be opened only for reading.

If the database does not exist, `gdbmtool` will create it. There is a special option `-n` (`--newdb`), which instructs the utility to create a new database. If it is used and if the database already exists, it will be deleted, so use it sparingly.

24.1 gdbmtool invocation

When started without additional arguments, `gdbmtool` operates on the default database `junk.gdbm`. Otherwise, the first argument supplies the name of the database to operate upon. If neither any additional arguments nor the `-f` (`--file`) option are given, `gdbmtool` opens starts interactive shell and receives commands directly from the human operator.

If more than one argument is given, all arguments past the database name are parsed as `gdbmtool` commands (see Section 24.2 [shell], page 54, for a description of available commands) and executed in turn. All commands, except the last one, should be terminated with semicolons. Semicolon after the last command is optional. Note, that semicolons should be escaped in order to prevent them from being interpreted by the shell.

Finally, if the `-f` (`--file`) option is supplied, its argument specifies the name of the disk file with `gdbmtool` script. The program will open that file and read commands from it.

The following table summarizes all `gdbmtool` command line options:

`-b` *size*
`--block-size=`*size*
: Set block size.

`-c` *size*
`--cache-size=`*size*
: Set cache size.

`-d` *fd*
`--db-descriptor=`*fd*
: Use the database referred to by the file descriptor *fd*. This must be a valid open file descriptor, obtained by a call to `open` (see Section "open a file" in *open(2) man page*), `creat` or a similar function. The database will be opened using `gdbm_fd_open` (see [gdbm_fd_open], page 7).

  This option is intended for use by automatic test suites.

`-f` *file*

`--file` *file*
: Read commands from *file*, instead of the standard input.

`-h`
`--help` Print a concise help summary.

`-N`
`--norc` Don't read startup files (see Section 24.2.4 [startup files], page 65).

`-n`
`--newdb` Create the database.

`-l`
`--no-lock`
Disable file locking.

`-m`
`--no-mmap`
Disable memory mapping.

`-T`
`--timing` Print time spent in each command. This is equivalent to setting the `timing` variable. See Section 24.2.1 [variables], page 55.

`-t`
`--trace` Enable command tracing. This is equivalent to setting the `trace` variable. See Section 24.2.1 [variables], page 55.

`-q`
`--quiet` Don't print the usual welcome banner at startup. This is the same as setting the variable `quiet` in the startup file. See [quiet], page 56.

`-r`
`--read-only`
Open the database in read-only mode.

`-s`
`--synchronize`
Synchronize to the disk after each write.

`-V`
`--version`
Print program version and licensing information and exit.

`--usage` Print a terse invocation syntax summary along with a list of available command line options.

`-x`
`--extended`
`--numsync`
Create new database in extended (numsync) format (see Section 17.8 [Numsync], page 32). This option sets the `format` variable to '`numsync`'. See [format variable], page 57.

24.2 gdbmtool interactive mode

After successful startup, `gdbmtool` starts a loop, in which it reads commands from the standard input, executes them and prints results on the standard output. If the standard input is attached to a console, `gdbmtool` runs in interactive mode, which is indicated by its *prompt*:

```
gdbmtool> _
```

The utility finishes when it reads the `quit` command (see below) or detects end-of-file on its standard input, whichever occurs first.

A `gdbmtool` command consists of a *command verb*, optionally followed by *arguments*, separated by any amount of white space and terminated with a newline or semicolon. A command verb can be entered either in full or in an abbreviated form, as long as that abbreviation does not match any other verb. For example, `co` can be used instead of `count` and `ca` instead of `cache`.

Any sequence of non-whitespace characters appearing after the command verb forms an argument. If the argument contains whitespace or unprintable characters it must be enclosed in double quotes. Within double quotes the usual *escape sequences* are understood, as shown in the table below:

Sequence	Replaced with
\a	Audible bell character (ASCII 7)
\b	Backspace character (ASCII 8)
\f	Form-feed character (ASCII 12)
\n	Newline character (ASCII 10)
\r	Carriage return character (ASCII 13)
\t	Horizontal tabulation character (ASCII 9)
\v	Vertical tabulation character (ASCII 11)
\\	Single slash
\"	Double quote

Table 24.1: Backslash escapes

In addition, a backslash immediately followed by the end-of-line character effectively removes that character, allowing to split long arguments over several input lines.

Command parameters may be optional or mandatory. If the number of actual arguments is less than the number of mandatory parameters, `gdbmtool` will prompt you to supply missing arguments. For example, the `store` command takes two mandatory parameters, so if you invoked it with no arguments, you would be prompted twice to supply the necessary data, as shown in example below:

```
gdbmtool> store
key? three
data? 3
```

However, such prompting is possible only in interactive mode. In non-interactive mode (e.g. when running a script), all arguments must be supplied with each command, otherwise `gdbmtool` will report an error and exit immediately.

If the package is compiled with GNU Readline, the input line can be edited (see Section "Command Line Editing" in *GNU Readline Library*).

24.2.1 Shell Variables

A number of `gdbmtool` parameters is kept in its internal variables. To examine or modify variables, use the `set` command (see [set], page 58).

`bool confirm` [gdbmtool variable]

Whether to ask for confirmation before certain destructive operations, such as truncating the existing database.

Default is `true`.

`string delim1` [gdbmtool variable]

A string used to delimit fields of a structured datum on output (see Section 24.2.3 [definitions], page 63).

Default is '`,`' (a comma). This variable cannot be unset.

`string delim2` [gdbmtool variable]

A string used to delimit array items when printing a structured datum (see Section 24.2.3 [definitions], page 63).

Default is '`,`' (a comma). This variable cannot be unset.

`string errorexit` [gdbmtool variable]
`bool errorexit` [gdbmtool variable]

Comma-delimited list of `GDBM` error codes which cause program termination. Error codes are specified via their canonical names (see Chapter 22 [Error codes], page 43). The `GDBM_` prefix can be omitted. Code name comparison is case-insensitive. Each error code can optionally be prefixed with minus sign, to indicate that it should be removed from the resulting list, or with plus sign (which is allowed for symmetry). A special code '`all`' stands for all available error codes.

In boolean context, the `true` value is equivalent to '`all`', and `false` (i.e. variable unset) is equivalent to '`-all`'.

`string errormask` [gdbmtool variable]
`bool errormask` [gdbmtool variable]

Comma-delimited list of `GDBM` error codes which are masked, i.e. which won't trigger a diagnostic message if they occur. The syntax is the same as described for `errorexit`.

`string pager` [gdbmtool variable]

The name and command line of the pager program to pipe output to. This program is used in interactive mode when the estimated number of output lines is greater then the number of lines on your screen.

The default value is inherited from the environment variable `PAGER`. Unsetting this variable disables paging.

`string ps1` [gdbmtool variable]

Primary prompt string. Its value can contain *conversion specifiers*, consisting of the '`%`' character followed by another character. These specifiers are expanded in the resulting prompt as follows:

Sequence	**Expansion**

%f	name of the current database file
%p	program invocation name
%P	package name (‘GDBM’)
%v	program version
%_	single space character
%%	%

The default value is ‘%p>%_’, i.e. the program name, followed by a “greater than” sign, followed by a single space.

`string ps2` [gdbmtool variable]

Secondary prompt. See `ps1` for a description of its value. This prompt is displayed before reading the second and subsequent lines of a multi-line command.

The default value is ‘%_>%_’.

`bool timing` [gdbmtool variable]

When each command terminates, print an additional line listing times spent in that command. The line is formatted as follows:

```
[reorganize r=0.070481 u=0.000200 s=0.000033]
```

Here, ‘`reorganize`’ is the name of the command that finished, the number after ‘`r=`’ is real time spent executing the command, the number after ‘`u=`’ is the user CPU time used and the number after ‘`s=`’ is the system CPU time used.

`bool trace` [gdbmtool variable]

Enable command tracing. This is similar to the shell `-t` option: before executing each command, `gdbmtool` will print on standard error a line starting with a plus sign and followed by the command name and its arguments.

`bool quiet` [gdbmtool variable]

Whether to display a welcome banner at startup. To affect `gdbmtool`, this variable should be set in a startup script file (see Section 24.2.4 [startup files], page 65). See [-q option], page 53.

The following variables control how the database is opened:

`numeric blocksize` [gdbmtool variable]

Sets the block size. See Chapter 3 [Open], page 5. Unset by default.

`numeric cachesize` [gdbmtool variable]

Sets the cache size. See Chapter 18 [Options], page 36.

This variable affects the currently opened database immediately. It is also used by `open` command.

To enable automatic cache size selection, unset this variable. This is the default.

`string filename` [gdbmtool variable]

Name of the database file. If the `open` command is called without argument (e.g. called implicitly), this variable names the database file to open. If `open` is called with file name argument, upon successful opening of the database the `filename` variable is initialized with its file name.

This variable cannot be unset.

`number fd` [gdbmtool variable]

File descriptor of the database file to open. If this variable is set, its value must be an open file descriptor referring to a `GDBM` database file. The `open` command will use `gdbm_fd_open` function to use this file (see [gdbm_fd_open], page 7). When this database is closed, the descriptor will be closed as well and the `fd` variable will be unset.

See also the `-d` (`--db-descriptor`) command line option in Section 24.1 [invocation], page 52.

`string format` [gdbmtool variable]

Defines the format in which new databases will be created. Allowed values are:

'`standard`'
: Databases will be created in standard format. This is the format used by all `GDBM` versions prior to 1.21. This value is the default.

'`numsync`'
: Extended format, best for crash-tolerant applications. See Section 17.8 [Numsync], page 32, for a discussion of this format.

`string open` [gdbmtool variable]

Open mode. The following values are allowed:

newdb
: Truncate the database if it exists or create a new one. Open it in read-write mode.

  Technically, this sets the `GDBM_NEWDB` flag in call to `gdbm_open`. See Chapter 3 [Open], page 5.

wrcreat
rw
: Open the database in read-write mode. Create it if it does not exist. This is the default.

  Technically speaking, it sets the `GDBM_WRCREAT` flag in call to `gdbm_open`. See Chapter 3 [Open], page 5.

reader
readonly
: Open the database in read-only mode. Signal an error if it does not exist.

  This sets the `GDBM_READER` flag (see Chapter 3 [Open], page 5).

Attempting to set any other value or to unset this variable results in error.

`number filemode` [gdbmtool variable]

File mode (in octal) for creating new database files and database dumps.

`bool lock` [gdbmtool variable]

Lock the database. This is the default.

Setting this variable to false or unsetting it results in passing `GDBM_NOLOCK` flag to `gdbm_open` (see Chapter 3 [Open], page 5).

`bool mmap` [gdbmtool variable]

Use memory mapping. This is the default.

Setting this variable to false or unsetting it results in passing `GDBM_NOMMAP` flag to `gdbm_open` (see Chapter 3 [Open], page 5).

`bool sync` [gdbmtool variable]
: Flush all database writes on disk immediately. Default is false. See Chapter 3 [Open], page 5.

`bool coalesce` [gdbmtool variable]
: Enables the *coalesce* mode, i.e. merging of the freed blocks of `GDBM` files with entries in available block lists. This provides for effective memory management at the cost of slight increase in execution time when calling `gdbm_delete`. See Chapter 18 [Options], page 36.

  This variable affects the currently opened database immediately and will be used by `open` command, when it is invoked.

`bool centfree` [gdbmtool variable]
: Set to `true`, enables the use of central free block pool in newly opened databases. See Chapter 18 [Options], page 36.

  This variable affects the currently opened database immediately and will be used by `open` command, when it is invoked.

The following commands are used to list or modify the variables:

`set` [*assignments*] [command verb]
: When used without arguments, lists all variables and their values. Unset variables are shown after a comment sign (‘`#`’). For string and numeric variables, values are shown after an equals sign. For boolean variables, only the variable name is displayed if the variable is `true`. If it is `false`, its name is prefixed with ‘`no`’.

  For example:

```
# blocksize is unset
# cachesize is unset
nocentfree
nocoalesce
confirm
delim1=","
delim2=","
# fd is unset
filemode=644
filename="junk.gdbm"
format="standard"
lock
mmap
open="wrcreat"
pager="less"
ps1="%p>%_"
ps2="%_>%_"
# quiet is unset
nosync
```

  If used with arguments, the `set` command alters the specified variables. In this case, arguments are variable assignments in the form ‘*`name=value`*’. For boolean variables,

the *value* is interpreted as follows: if it is numeric, `0` stands for `false`, any non-zero value stands for `true`. Otherwise, the values `on`, `true`, and `yes` denote `true`, and `off`, `false`, `no` stand for `false`. Alternatively, only the name of a boolean variable can be supplied to set it to `true`, and its name prefixed with `no` can be used to set it to false. For example, the following command sets the `delim2` variable to '`;`' and the `confirm` variable to `false`:

```
set delim2=";" noconfirm
```

`unset` *variables* [command verb]

Unsets the listed variables. The effect of unsetting depends on the variable. Unless explicitly described in the discussion of the variables above, unsetting a boolean variable is equivalent to setting it to `false`. Unsetting a string variable is equivalent to assigning it an empty string.

24.2.2 Gdbmtool Commands

`avail` [command verb]

Print the *avail list*.

`bucket` *num* [command verb]

Print the bucket number *num* and set it as the current one.

`cache` [command verb]

Print the bucket cache.

`close` [command verb]

Close the currently open database.

`count` [command verb]

Print the number of entries in the database.

`current` [command verb]

Print the current bucket.

`debug` [[+-]*token*...] [command verb]

If `GDBM` is configured with additional debugging, this statement queries or sets `GDBM` internal debugging level. This is intended for debugging and testing purposes and requires good knowledge of `GDBM` internals. The use of this command is not recommended.

`delete` *key* [command verb]

Delete record with the given *key*

`dir` [command verb]

Print hash directory.

`downgrade` [command verb]

Downgrade the database from extended to the standard database format. See Section 17.8 [Numsync], page 32.

`export` *`file-name`* [*truncate*] [*binary*|*ascii*] [command verb]
: Export the database to the flat file *file-name*. See Chapter 13 [Flat files], page 19, for a description of the flat file format and its purposes. This command will not overwrite an existing file, unless the '`truncate`' parameter is also given. Another optional argument determines the type of the dump (see Chapter 13 [Flat files], page 19). By default, ASCII dump is created.

  The global variable `filemode` specifies the permissions to use for the created output file.

`fetch` *`key`* [command verb]
: Fetch and display the record with the given *key*.

`first` [command verb]
: Fetch and display the first record in the database. Subsequent records can be fetched using the `next` command (see below). See Chapter 9 [Sequential], page 14, for more information on sequential access.

`hash` *`key`* [command verb]
: Compute and display the hash value for the given *key*.

`header` [command verb]
: Print file header.

`help` [command verb]
`?` [command verb]
: Print a concise command summary, showing each command verb with its parameters and a short description of what it does. Optional arguments are enclosed in square brackets.

`import` *`file-name`* [*replace*] [*nometa*] [command verb]
: Import data from a flat dump file *file-name* (see Chapter 13 [Flat files], page 19). If the word '`replace`' is given as an argument, any records with the same keys as the already existing ones will replace them. The word '`nometa`' turns off restoring meta-information from the dump file.

`history` [command verb]
`history` *`count`* [command verb]
`history` *`n count`* [command verb]
: Shows the command history list with line numbers. When used without arguments, shows entire history. When used with one argument, displays *count* last commands from the history. With two arguments, displays *count* commands starting from *n*th command. Command numbering starts with 1.

  This command is available only if `GDBM` was compiled with GNU Readline. The history is saved in file `.gdbmtool_history` in the user's home directory. If this file exists upon startup, it is read to populate the history. Thus, command history is preserved between `gdbmtool` invocations.

`list` [command verb]
: List the contents of the database.

`next` [*key*] [command verb]

Sequential access: fetch and display the next record. If the *key* is given, the record following the one with this key will be fetched.

Issuing several `next` commands in row is rather common. A shortcut is provided to facilitate such use: if the last entered command was `next`, hitting the ***Enter*** key repeats it without arguments.

See also `first`, above.

See Chapter 9 [Sequential], page 14, for more information on sequential access.

`open` *filename* [command verb]
`open` [command verb]

Open the database file *filename*. If used without arguments, the database name is taken from the variable `filename`.

If successful, any previously open database is closed and the `filename` variable is updated. Otherwise, if the operation fails, the currently opened database remains unchanged.

This command takes additional information from the following variables:

`filename` Name of the database to open, if no argument is given.

`fd` File descriptor to use. If set, this must be an open file descriptor referring to a valid database file. The database will be opened using `gdbm_fd_open` (see [gdbm_fd_open], page 7). The file descriptor will be closed and the variable unset upon closing the database.

`filemode` Specifies the permissions to use in case a new file is created.

`open` The database access mode. See [The *open* variable], page 57, for a list of its values.

`lock` Whether or not to lock the database. Default is `on`.

`mmap` Use the memory mapping. Default is `on`.

`sync` Synchronize after each write. Default is `off`.

See [open parameters], page 56, for a detailed description of these variables.

`perror` [*code*] [command verb]

Describe the given `GDBM` error code.

The description occupies one or two lines. The second line is present if the system error number should be checked when handling this code. In this case, the second line states '`Examine errno`'.

If *code* is omitted, the latest error that occurred in the current database is described. Second line of the output (if present), contains description of the latest system error.

Example:

```
gdbmtool> perror 3
GDBM error code 3: "File open error"
Examine errno.
```

`quit` [command verb]

Close the database and quit the utility.

`recover` [*options*] [command verb]

Recover the database from structural inconsistencies. See Chapter 15 [Database consistency], page 25.

The following *options* are understood:

`backup` Create a backup copy of the original database.

`max-failed-buckets=`*n*
Abort recovery process if *n* buckets could not be recovered.

`max-failed-keys=`*n*
Abort recovery process if *n* keys could not be recovered.

`max-failures=`*n*
Abort recovery process after *n* failures. A *failure* in this context is either a key or a bucket that failed to be recovered.

`summary` Print the recovery statistics at the end of the run. The statistics includes number of successfully recovered, failed and duplicate keys and the number of recovered and failed buckets.

`verbose` Verbosely list each error encountered.

`reorganize` [command verb]

Reorganize the database (see Chapter 10 [Reorganization], page 16).

`shell` *command* [command verb]
`!` *command* [command verb]

Execute *command* via current shell. If *command* is empty, shell is started without additional arguments. Otherwise, it is run as '`$SHELL -c` *command*'.

For convenience, *command* is not parsed as `gdbmtool` command line. It is passed to the shell verbatim. It can include newline characters if these are preceded by a backslash or appear within singly or doubly quoted strings.

When using `!` form, be sure to separate it from *command* by whitespace, otherwise it will be treated as readline *event specifier*.

`snapshot` *filename filename* [command verb]

Analyze two snapshot files and select the most recent of them. In case of error, display a detailed diagnostics and meta-information of both snapshots.

See Section 17.5 [Manual crash recovery], page 31, for a detailed discussion.

`source` *filename* [command verb]

Read `gdbmtool` commands from the file *filename*.

`status` [command verb]

Print current program status. The following example shows the information displayed:

```
Database file: junk.gdbm
Database is open
define key string
define content string
```

The two `define` strings show the defined formats for key and content data. See Section 24.2.3 [definitions], page 63, for a detailed discussion of their meaning.

`store` *key data* [command verb]
: Store the *data* with *key* in the database. If *key* already exists, its data will be replaced.

`sync` [command verb]
: Synchronize the database with the disk storage (see Chapter 11 [Sync], page 17).

`upgrade` [command verb]
: Upgrade the database from standard to extended database format. See Section 17.8 [Numsync], page 32.

`version` [command verb]
: Print the version of `gdbm`.

24.2.3 Data Definitions

`GDBM` databases are able to keep data of any type, both in the key and in the content part of a record. Quite often these data are structured, i.e. they consist of several fields of various types. `Gdbmtool` provides a mechanism for handling such kind of records.

The `define` command defines a record structure. The general syntax is:

```
define what definition
```

where *what* is `key` to defining the structure of key data and `content` to define the structure of the content records.

The *definition* can be of two distinct formats. In the simplest case it is a single data type. For example,

```
define content int
```

defines content records consisting of a single integer field. Supported data types are:

char
: Single byte (signed).

short
: Signed short integer.

ushort
: Unsigned short integer.

int
: Signed integer.

unsigned
uint
: Unsigned integer.

long
: Signed long integer.

ulong
: Unsigned long integer.

llong
: Signed long long integer.

ullong
: Unsigned long long integer.

float A floating point number.

double Double-precision floating point number.

string Array of bytes.

stringz Null-terminated string, trailing null being part of the string.

All numeric data types (integer as well as floating point) have the same respective widths as in C language on the host where the database file resides.

The `string` and `stringz` are special. Both define a string of bytes, similar to '`char x[]`' in C. The former defines an array of bytes, the latter - a null-terminated string. This makes a difference, in particular, when the string is the only part of datum. Consider the following two definitions:

1. `define key string`
2. `define key stringz`

Now, suppose we want to store the string "ab" in the key. Using the definition (1), the `dptr` member of `GDBM datum` will contain two bytes: '`a`', and '`b`'. Consequently, the `dsize` member will have the value 2. Using the definition (2), the `dptr` member will contain three bytes: '`a`', '`b`', and ASCII 0. The `dsize` member will have the value 3.

The definition (1) is the default for both key and content.

The second form of the `define` statement is similar to the C `struct` statement and allows for defining structural data. In this form, the *definition* part is a comma-separated list of data types and variables enclosed in curly braces. In contrast to the rest of `gdbm` commands, this command is inherently multiline and is terminated with the closing curly brace. For example:

```
define content {
        int status,
        pad 8,
        char id[3],
        string name
}
```

This defines a structure consisting of three members: an integer `status`, an array of 3 bytes `id`, and an array of bytes `name`. Notice the `pad` statement: it allows to introduce padding between structure members. Another useful statement is `offset`: it specifies that the member following it begins at the given offset in the structure. Assuming the size of `int` is 8 bytes, the above definition can also be written as

```
define content {
        int status,
        offset 16,
        char id[3],
        string name
}
```

NOTE: The `string` type can reasonably be used only if it is the last or the only member of the data structure. That's because it provides no information about the number of elements in the array, so it is interpreted to contain all bytes up to the end of the datum.

When displaying the structured data, `gdbmtool` precedes each value with the corresponding field name and delimits parts of the structure with the string defined in the `delim1` variable (see Section 24.2.1 [variables], page 55). Array elements are delimited using the string from `delim2`. For example:

```
gdbmtool> fetch foo
status=2,id={ a, u, x },name="quux"
```

To supply a structured datum as an argument to a `gdbmtool` command, use the same notation, e.g.:

```
gdbmtool> store newkey { status=2, id={a,u,x}, name="quux" }
```

The order in which the fields are listed is not significant. The above command can as well be written as:

```
gdbmtool> store newkey { id={a,u,x}, status=2, name="quux" }
```

You are not required to supply all defined fields. Any number of them can be omitted, provided that at least one remains. The omitted fields are filled with 0:

```
gdbmtool> store newkey { name="bar" }
gdbmtool> fetch newkey
status=0,id={ ,, },name=bar
```

Yet another way to supply structured data to a command is by listing the value for each field in the order they are defined, without field names:

```
gdbmtool> store newkey { 2, {a,u,x}, "quux" }
```

24.2.4 Startup Files

Upon startup `gdbmtool` looks for a file named `.gdbmtoolrc` first in the current working directory and, if not found, in the home directory of the user who started the command.

If found, this file is read and interpreted as a list of `gdbmtool` commands. This allows you to customize the program behavior.

Following is an example startup file which disables the welcome banner, sets command line prompt to contain the name of the database file in parentheses and defines the structure of the database content records:

```
set quiet
set ps1="(%f) "
define key stringz
define content {
        int time,
        pad 4,
        int status
}
```

25 The gdbm_dump utility

The `gdbm_dump` utility creates a flat file dump of a `GDBM` database (see Chapter 13 [Flat files], page 19). It takes one mandatory argument: the name of the source database file. The second argument, if given, specifies the name of the output file. If not given, `gdbm_dump` will produce the dump on the standard output.

For example, the following invocation creates a dump of the database `file.db` in the file `file.dump`:

```
$ gdbm_dump file.db file.dump
```

By default the utility creates dumps in ASCII format (see Chapter 13 [Flat files], page 19). Another format can be requested using the `--format` (`-H`) option.

The `gdbm_dump` utility understands the following command line options:

`-H` *fmt*
`--format=`*fmt*
: Select output format. Valid values for *fmt* are: `binary` or `0` to select binary dump format, and `ascii` or `1` to select ASCII format.

`-h`
`--help`
: Print a concise help summary.

`-V`
`--version`
: Print program version and licensing information and exit.

`--usage`
: Print a terse invocation syntax summary along with a list of available command line options.

26 The gdbm_load utility

The `gdbm_load` utility restores a `GDBM` database from a flat file. The utility requires at least one argument: the name of the input flat file. If it is '`-`', the standard input will be read. The format of the input file is detected automatically.

By default the utility attempts to restore the database under its original name, as stored in the input file. It will fail to do so if the input is in binary format. In that case, the name of the database must be given as the second argument.

In general, if two arguments are given, the second one is treated as the name of the database to create, overriding the file name specified in the flat file.

The utility understands the following command line arguments:

`-b` *num*
`--block-size=`*num*
: Sets block size. See Chapter 3 [Open], page 5.

`-c` *num*
`--cache-size=`*num*
: Sets cache size. See Chapter 18 [Options], page 36.

`-M`
`--mmap`
: Use memory mapping.

`-m` *mode*

`--mode=`*mode*
: Sets the file mode. The argument is the desired file mode in octal.

`-n`
`--no-meta`
: Do not restore file meta-data (ownership and mode) from the flat file.

`-r`
`--replace`
: Replace existing keys.

`-u` *user*`[:`*group*`]`
`--user=`*user*`[:`*group*`]`
: Set file owner. The *user* can be either a valid user name or UID. Similarly, the *group* is either a valid group name or GID. If *group* is not given, the main group of *user* is used.

  User and group parts can be separated by a dot, instead of the colon.

`-h`
`--help`
: Print a concise help summary.

`-V`
`--version`
: Print program version and licensing information and exit.

`--usage`
: Print a terse invocation syntax summary along with a list of available command line options.

27 Exit codes

All GDBM utilities return uniform exit codes. These are summarized in the table below:

Code	Meaning
0	Successful termination.
1	A fatal error occurred.
2	Program was unable to restore file ownership or mode.
3	Command line usage error.

28 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 `bug-gdbm@gnu.org`.

Please include the version number of GNU `dbm` you are using. You can get this information by printing the variable `gdbm_version` (see Chapter 20 [Variables], page 40).

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 authors and maintainers by e-mail: Philip Nelson `phil@cs.wwu.edu`, Jason Downs `downsj@downsj.com`, Sergey Poznyakoff `gray@gnu.org` or `gray@gnu.org.ua`.

Crash tolerance support written by Terence Kelly `tpkelly@acm.org`, `tpkelly@cs.princeton.edu`, or `tpkelly@eecs.umich.edu`.

29 Additional resources

For the latest updates and pointers to additional resources, visit http://www.gnu.org/software/gdbm.

In particular, a copy of GDBM documentation in various formats is available online at http://www.gnu.org/software/gdbm/manual.html.

Latest versions of GDBM can be downloaded from anonymous FTP: ftp://ftp.gnu.org/gnu/gdbm, or via HTTP from http://ftp.gnu.org/gnu/gdbm, or via HTTPS from https://ftp.gnu.org/gnu/gdbm, or from any GNU mirror worldwide. See http://www.gnu.org/order/ftp.html, for a list of mirrors.

To track GDBM development, visit http://puszcza.gnu.org.ua/projects/gdbm.

Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at

your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V