MIT/GNU Scheme GDBM Plugin Manual

a GNU database manager plugin (version 1.0)
for MIT/GNU Scheme version 10.1.11
4 June 2020

by Matt Birkholz

This manual documents MIT/GNU Scheme GDBM 1.0.

1 Introduction to GNU dbm.

This plugin is a dynamically loadable wrapper for the GNU dbm (DataBase Management) C library. This manual is a derivative of Edition 1.5 of the *GNU dbm Manual*, for library version 1.8.3, last updated October 15, 2002.

GNU dbm (gdbm) is a library of database functions that use extendible hashing; it works similarly to the standard UNIX dbm functions. 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 key/data pairs are stored in a gdbm disk file, called a gdbm database. A program must connect to a gdbm database to be able manipulate the keys and data contained in it. Gdbm allows Scheme to connect to multiple databases at the same time. When Scheme connects to a gdbm database, the connection is designated as a *reader* or a *writer*. A gdbm database may be connected to at most one writer at a time. However, many readers may connect to the database simultaneously. Readers and writers may not connect to the database at the same time. (Note that these restrictions are not enforced by the library nor the wrapper.)

Each connection is encapsulated in a Scheme `gdbf` structure which should be used by one Scheme thread at a time. A mutex is used to block any thread attempting to access the database while an operation is in progress. No file locks are used by gdbm or the Scheme wrapper to ensure exclusive access by a Scheme writer.

2 The exported bindings.

The following is a quick list of the procedures provided by the plugin.

```
gdbm-open
gdbm-close
gdbm-store
gdbm-fetch
gdbm-delete
gdbm-firstkey
gdbm-nextkey
gdbm-reorganize
gdbm-sync
gdbm-exists?
gdbm-setopt
```

Neither `gdbm_errno` nor `gdbm_strerror` are exposed because the plugin automatically tests and calls them to detect errors and convert codes into strings. `gdbm_fdesc` is also not exposed, treated as an implementation detail the plugin should probably hide, used by tricky code that cooperates with multiple file locking libraries.

There is one global variable, `gdbm-version`, which is initialized from the library's `gdbm_version` string.

And several constants:

```
gdbm_cachesize
        gdbm_fast
        gdbm_insert
        gdbm_newdb
        gdbm_reader
        gdbm_replace
        gdbm_wrcreat
        gdbm_writer
```

You can load these bindings into your global environment with the following expresson.

```
(load-option 'gdbm)
```

And you can include these bindings in your package description (`.pkg`) file with the following expression.

```
(global-definitions gdbm/)
```

3 Opening the database.

Connect to the file. If the file has a size of zero bytes, a file initialization procedure is performed, setting up the initial structure in the file.

The procedure for opening a gdbm file is:

`gdbm-open` *name block-size flags mode* [Procedure]

The parameters are:

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

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

flags If *flags* is `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 `gdbm_writer`, the user wants both read and write access to the database and requires exclusive access. If *flags* is `gdbm_wrcreat`, the user wants both read and write access to the database and if the database does not exist, create a new one. If *flags* is `gdbm_newdb`, the user want a new database created, regardless of whether one existed, and wants read and write access to the new database. The following may also be logically or'd into the database flags: `gdbm_sync`, which causes all database operations to be synchronized to the disk, and `gdbm_nolock`, which prevents the library from performing any locking on the database file. `gdbm_fast` is now obsolete, since gdbm defaults to no-sync mode.

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

The return value is the object needed by all other procedures to access that gdbm file.

4 Closing the database.

It is important that every file opened is also closed. This is needed to update the reader/writer count on the file. Scheme will do this automatically if an open gdbm object is garbage collected, but you can close the file immediately with the `gdbm-close` procedure.

`gdbm-close` *dbf* [Procedure]

The parameter is:

dbf The object returned by `gdbm-open`.

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

5 Inserting and replacing records in the database.

The procedure `gdbm-store` inserts or replaces records in the database.

`gdbm-store` *dbf key content flag* [Procedure]

The parameters are:

dbf The object returned by `gdbm-open`.

key A non-empty string, converted to utf-8 bytes for lookup in the database.

content Another non-empty string, the content to be stored in the database file, also converted to utf-8.

flag The action to take when *key* is already in the database. The value of `gdbm_replace` indicates that the old content should be replaced by *content*. The value of `gdbm_insert` indicates that `#f` should be returned and no action taken if *key* already exists.

The values returned are:

`#t` Success. *content* is keyed by *key*. The file on disk is updated to reflect the structure of the new database before returning from this procedure.

`#f` The item was not stored because *flag* was `gdbm_insert` and *key* was already in the database.

An error is signaled if the caller is not a writer.

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

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

6 Searching for records in the database.

Read content associated with a key.

`gdbm-fetch` *dbf key* [Procedure]

The parameters are:

dbf The object returned by `gdbm-open`.

key A non-empty string, converted to utf-8 bytes for lookup in the database.

The return value is a string created from the utf-8 bytes found in the database, or `#f` if no content was found.

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

`gdbm-exists?` *dbf key* [Procedure]

The parameters are:

dbf The pointer returned by `gdbm-open`.

key A non-empty string, converted to utf-8 bytes for lookup in the database.

Unlike `gdbm-fetch` this procedure does not read any content and simply returns true or false depending on whether *key* exists.

7 Removing records from the database.

To remove some content from the database:

`gdbm-delete` *dbg key* [Procedure]

The parameters are:

dbf The object returned by `gdbm-open`.

key A non-empty string, converted to utf-8 bytes for lookup in the database.

The return value is `#f` if the item is not present or the requester is a reader. The return value is `#t` if there was a successful delete.

The keyed content and the key are removed from the database. The file on disk is updated to reflect the structure of the new database before returning from this procedure.

8 Sequential access to records.

The next two functions allow for accessing all content in a database *dbf*. 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` *dbf* [Procedure]
: Starts the visit of all keys in the database *dbf*. Returns the first key to visit, converting its utf-8 bytes to a string. If there are no keys, returns `#f`.

`gdbm-nextkey` *dbf key* [Procedure]
: Returns the key to visit after *key*, converting its utf-8 bytes to a string. If there are no more keys, returns `#f`.

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

Visiting keys traverses a hash table which writers may re-arrange. The original key order is *not* guaranteed to remain unchanged in all instances. It is possible that some key will not be visited if the database is changed while traversing the table.

9 Database reorganization.

The following procedure should be used very seldom.

`gdbm-reorganize` *dbf* [Procedure]

If you have made a lot of deletions and would like to shrink the space used by the gdbm file, this function will reorganize the database. Gdbm will not shorten a gdbm file (will not reuse deleted space) until this procedure is called.

The 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.

10 Database Synchronization

Unless you opened your database with the `gdbm_sync` flag, gdbm does not wait for writes to be flushed to the disk. This allows faster writing of databases at the risk of having a corrupted database if Scheme terminates in an abnormal fashion. The following function allows the programmer to flush all changes to disk.

`gdbm-sync` *dbf* [Procedure]

This would usually be called after a complete set of changes have been made to the database and before some long waiting time. `Gdbm-close` always flushes any changes to disk.

11 Setting options.

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

`gdbm-setopt` *dbf option value* [Procedure]

The parameters are:

dbf The pointer returned by `gdbm-open`.

option The option to be set, the value of `gdbm_cachesize` or `gdbm_syncmode`.

value The value to be set, an integer.

If *option* is `gdbm_cachesize` the size of the internal bucket cache is set to the given integer. This option may only be set once on a database, and is set to 100 by default when the database is first accessed.

If *option* is `gdbm_syncmode` file system synchronization is turned on or off. By default it is off. *Value* should `1` to turn it on, or `0` to turn it off.

The obsolete and experimental options `GDBM_FASTMODE`, `GDBM_CENTFREE` and `GDBM_COALESCEBLKS` are not supported by this plugin.