Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'dbm_clearerr.3p' command

$ man dbm_clearerr.3p

DBM_CLEARERR(3P) POSIX Programmer's Manual DBM_CLEARERR(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey, dbm_open, dbm_store ? database functions

SYNOPSIS

```
#include <ndbm.h>
int dbm_clearerr(DBM *db);
void dbm_close(DBM *db);
int dbm_delete(DBM *db, datum key);
int dbm_error(DBM *db);
datum dbm_fetch(DBM *db, datum key);
datum dbm_firstkey(DBM *db);
datum dbm_nextkey(DBM *db);
DBM *dbm_open(const char *file, int open_flags, mode_t file_mode);
int dbm_store(DBM *db, datum key, datum content, int store_mode);
```

DESCRIPTION

These functions create, access, and modify a database.

A datum consists of at least two members, dptr and dsize. The dptr

member points to an object that is dsize bytes in length. Arbitrary bi?
nary data, as well as character strings, may be stored in the object
pointed to by dptr.

A database shall be stored in one or two files. When one file is used,
the name of the database file shall be formed by appending the suffix
.db to the file argument given to dbm_open(). When two files are used,
the names of the database files shall be formed by appending the suf?
fixes .dir and .pag respectively to the file argument.

The dbm_open() function shall open a database. The file argument to the
function is the pathname of the database. The open_flags argument has
the same meaning as the flags argument of open() 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 file_mode ar?
gument has the same meaning as the third argument of open().

The dbm_open() function need not accept pathnames longer than
{PATH_MAX}-4 bytes (including the terminating null), or pathnames with
a last component longer than {NAME_MAX}-4 bytes (excluding the termi?
nating null).

The dbm_close() function shall close a database. The application shall
ensure that argument db is a pointer to a dbm structure that has been
returned from a call to dbm_open().

These database functions shall support an internal block size large
enough to support key/content pairs of at least 1023 bytes.

The dbm_fetch() function shall read a record from a database. The argu?
ment db is a pointer to a database structure that has been returned
from a call to dbm_open(). The argument key is a datum that has been
initialized by the application to the value of the key that matches the
key of the record the program is fetching.

The dbm_store() function shall write a record to a database. The argu?
ment db is a pointer to a database structure that has been returned
from a call to dbm_open(). The argument key is a datum that has been
initialized by the application to the value of the key that identifies
(for subsequent reading, writing, or deleting) the record the applica?

tion is writing. The argument content is a datum that has been initial?
ized by the application to the value of the record the program is writ?
ing. The argument store_mode controls whether dbm_store() replaces any
pre-existing record that has the same key that is specified by the key
argument. The application shall set store_mode to either DBM_INSERT or
DBM_REPLACE. If the database contains a record that matches the key ar?
gument and store_mode is DBM_REPLACE, the existing record shall be re?
placed with the new record. If the database contains a record that
matches the key argument and store_mode is DBM_INSERT, the existing
record shall be left unchanged and the new record ignored. If the data?
base does not contain a record that matches the key argument and
store_mode is either DBM_INSERT or DBM_REPLACE, the new record shall be
inserted in the database.
If the sum of a key/content pair exceeds the internal block size, the
result is unspecified. Moreover, the application shall ensure that all
key/content pairs that hash together fit on a single block. The
dbm_store() function shall return an error in the event that a disk
block fills with inseparable data.
The dbm_delete() function shall delete a record and its key from the
database. The argument db is a pointer to a database structure that has
been returned from a call to dbm_open(). The argument key is a datum
that has been initialized by the application to the value of the key
that identifies the record the program is deleting.
The dbm_firstkey() function shall return the first key in the database.
The argument db is a pointer to a database structure that has been re?
turned from a call to dbm_open().
The dbm_nextkey() function shall return the next key in the database.
The argument db is a pointer to a database structure that has been re?
turned from a call to dbm_open(). The application shall ensure that
the dbm_firstkey() function is called before calling dbm_nextkey().
Subsequent calls to dbm_nextkey() return the next key until all of the
keys in the database have been returned.
The dbm_error() function shall return the error condition of the data?

base. The argument db is a pointer to a database structure that has been returned from a call to dbm_open().

The dbm_clearerr() function shall clear the error condition of the database. The argument db is a pointer to a database structure that has been returned from a call to dbm_open().

The dptr pointers returned by these functions may point into static storage that may be changed by subsequent calls.

These functions need not be thread-safe.

RETURN VALUE

The dbm_store() and dbm_delete() functions shall return 0 when they succeed and a negative value when they fail.

The dbm_store() function shall return 1 if it is called with a flags value of DBM_INSERT and the function finds an existing record with the same key.

The dbm_error() function shall return 0 if the error condition is not set and return a non-zero value if the error condition is set.

The return value of dbm_clearerr() is unspecified.

The dbm_firstkey() and dbm_nextkey() functions shall return a key da? tum. When the end of the database is reached, the dptr member of the key is a null pointer. If an error is detected, the dptr member of the key shall be a null pointer and the error condition of the database shall be set.

The dbm_fetch() function shall return a content datum. If no record in the database matches the key or if an error condition has been detected in the database, the dptr member of the content shall be a null pointer.

The dbm_open() function shall return a pointer to a database structure.

If an error is detected during the operation, dbm_open() shall return a (DBM *)0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The following code can be used to traverse the database:

for(key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))

The dbm_* functions provided in this library should not be confused in any way with those of a general-purpose database management system. These functions do not provide for multiple search keys per entry, they do not protect against multi-user access (in other words they do not lock records or files), and they do not provide the many other useful database functions that are found in more robust database management systems. Creating and updating databases by use of these functions is relatively slow because of data copies that occur upon hash collisions. These functions are useful for applications requiring fast lookup of relatively static information that is to be indexed by a single key.

Note that a strictly conforming application is extremely limited by these functions: since there is no way to determine that the keys in use do not all hash to the same value (although that would be rare), a strictly conforming application cannot be guaranteed that it can store more than one block's worth of data in the database. As long as a key collision does not occur, additional data may be stored, but because there is no way to determine whether an error is due to a key collision or some other error condition (dbm_error() being effectively a Bool? ean), once an error is detected, the application is effectively limited to guessing what the error might be if it wishes to continue using these functions.

The dbm_delete() function need not physically reclaim file space, al? though it does make it available for reuse by the database.

After calling dbm_store() or dbm_delete() during a pass through the keys by dbm_firstkey() and dbm_nextkey(), the application should reset the database by calling dbm_firstkey() before again calling dbm_nex? tkey(). The contents of these files are unspecified and may not be portable.

Applications should take care that database pathname arguments speci?

fied to dbm_open() are not prefixes of unrelated files. This might be done, for example, by placing databases in a separate directory.

Since some implementations use three characters for a suffix and others use four characters for a suffix, applications should ensure that the maximum portable pathname length passed to dbm_open() is no greater than {PATH_MAX}-4 bytes, with the last component of the pathname no greater than {NAME_MAX}-4 bytes.

RATIONALE

Previously the standard required the database to be stored in two files, one file being a directory containing a bitmap of keys and hav? ing .dir as its suffix. The second file containing all data and having .pag as its suffix. This has been changed not to specify the use of the files and to allow newer implementations of the Berkeley DB interface using a single file that have evolved while remaining compatible with the application programming interface. The standard developers consid? ered removing the specific suffixes altogether but decided to retain them so as not to pollute the application file name space more than necessary and to allow for portable backups of the database.

FUTURE DIRECTIONS

None.

SEE ALSO

open()

The Base Definitions volume of POSIX.1?2017, <ndbm.h>

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Por? table Operating System Interface (POSIX), The Open Group Base Specifi? cations Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.ker?nel.org/doc/man-pages/reporting_bugs.html .