NAME

ndbm, dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey, dbm_open, dbm_store – data base subroutines

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(char **file***, int** *flags***, int** *mode***);**

int dbm_store(DBM **db***, datum** *key***, datum** *content***, int** *flags***);**

MT-LEVEL

Unsafe

DESCRIPTION

These functions maintain *key* / *content* pairs in a data base. The functions will handle very large (a billion blocks) data base and will access a keyed item in one or two file system accesses. This package replaces the earlier **dbm**(3B) library, which managed only a single data base.

*key*s and *content*s are described by the **datum** typedef. A **datum** specifies a string of *dsize* bytes pointed to by *dptr*. Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has **.dir** as its suffix. The second file contains all data and has **.pag** as its suffix.

Before a data base can be accessed, it must be opened by **dbm_open( )**. This will open and/or create the files *file***.dir** and *file***.pag** depending on the flags parameter (see **open**(2)).

A data base is closed by calling **dbm_close( )**.

Once open, the data stored under a key is accessed by **dbm_fetch( )** and data is placed under a key by **dbm_store( )**. The *flags* field can be either **DBM_INSERT** or **DBM_REPLACE**. **DBM_INSERT** will only insert new entries into the data base and will not change an existing entry with the same key. **DBM_REPLACE** will replace an existing entry if it has the same key. A key (and its associated contents) is deleted by **dbm_delete( )**. A linear pass through all keys in a data base may be made, in an (apparently) random order, by use of **dbm_firstkey( )** and **dbm_nextkey( ). dbm_firstkey( )** will return the first key in the data base. **dbm_nextkey( )** will return the next key in the data base. This code will traverse the data base:

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

dbm_error( ) returns non-zero when an error has occurred reading or writing the data base. **dbm_clearerr( )** resets the error condition on the named data base.

RETURN VALUES

All functions that return an **int** indicate errors with negative values. A return value of **0** indicates no error. Routines that return a **datum** indicate errors with a **NULL** (0) *dptr*. If **dbm_store( )** is called with a *flags* value of **DBM_INSERT** and finds an existing entry with the same key, it returns **1**.

EXAMPLES

The following example stores and retrieves a phone number, using the name as the key. Note that this example does not include error checking.

```
#include <ndbm.h>
#include <stdio.h>
#include <fcntl.h>

#define NAME          "Bill"
#define PHONE_NO      "123-4567"
#define DB_NAME       "phones"

main()
{
        DBM *db;
        datum name = {NAME, sizeof (NAME)};
        datum put_phone_no = {PHONE_NO, sizeof (PHONE_NO)};
        datum get_phone_no;

        /* Open the database and store the record */
        db = dbm_open(DB_NAME, O_RDWR | O_CREAT, 0660);
        (void) dbm_store(db, name, put_phone_no, DBM_INSERT);

        /* Retrieve the record */
        get_phone_no = dbm_fetch(db, name);
        (void) printf("Name: %s, Phone Number: %s0, name.dptr,
          get_phone_no.dptr);

        /* Close the database */
        dbm_close(db);
        return (0);
}
```

SEE ALSO

ar(1), **cat**(1), **cp**(1), **tar**(1), **open**(2), **dbm**(3B), **netconfig**(4)

NOTES

The **.pag** file will contain holes so that its apparent size may be larger than its actual content. Older versions of the UNIX operating system may create real file blocks for these holes when touched. These files cannot be copied by normal means (**cp**(1), **cat**(1), **tar**(1), **ar**(1)) without filling in the holes.

dptr pointers returned by these subroutines point into static storage that is changed by subsequent calls.

The sum of the sizes of a *key / content* pair must not exceed the internal block size (currently 1024 bytes). Moreover all *key / content* pairs that hash together must fit on a single block. **dbm_store( )** will return an error in the event that a disk block fills with inseparable data.

dbm_delete( ) does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by **dbm_firstkey( )** and **dbm_nextkey( )** depends on a hashing function.

There are no interlocks and no reliable cache flushing; thus concurrent updating and reading is risky.

The database files (*file*.**dir** and *file*.**pag**) are binary and are architecture-specific (for example, they depend on the architecture's byte order.) These files are not guaranteed to be portable across architectures.