NAME

dbopen – database access methods

SYNOPSIS

#include <sys/types.h>
#include <limits.h>
#include <db.h>

DB *
dbopen(const char *file, int flags, int mode, DBTYPE type,
const void *openinfo);

DESCRIPTION

Dbopen is the library interface to database files. The supported file formats are btree, hashed and UNIX file oriented. The btree format is a representation of a sorted, balanced tree structure. The hashed format is an extensible, dynamic hashing scheme. The flat-file format is a byte stream file with fixed or variable length records. The formats and file format specific information are described in detail in their respective manual pages *btree*(3), *hash*(3) and *recno*(3).

Dbopen opens *file* for reading and/or writing. Files never intended to be preserved on disk may be created by setting the file parameter to NULL.

The *flags* and *mode arguments* are as specified to the *open*(2) routine, however, only the O_CREAT, O_EXCL, O_EXLOCK, O_RDONLY, O_RDWR, O_SHLOCK and O_TRUNC flags are meaningful. (Note, opening a database file O_WRONLY is not possible.)

The *type* argument is of type DBTYPE (as defined in the <db.h> include file) and may be set to DB_BTREE, DB_HASH or DB_RECNO.

The *openinfo* argument is a pointer to an access method specific structure described in the access method's manual page. If *openinfo* is NULL, each access method will use defaults appropriate for the system and the access method.

Dbopen returns a pointer to a DB structure on success and NULL on error. The DB structure is defined in the <db.h> include file, and contains at least the following fields:

```
typedef struct {
        DBTYPE type;
        int (*close)(const DB *db);
        int (*del)(const DB *db, const DBT *key, u_int flags);
        int (*fd)(const DB *db);
        int (*get)(const DB *db, DBT *key, DBT *data, u_int flags);
        int (*put)(const DB *db, DBT *key, const DBT *data,
                u_int flags);
        int (*sync)(const DB *db, u_int flags);
        int (*seq)(const DB *db, DBT *key, DBT *data, u_int flags);
} DB;
```

These elements describe a database type and a set of functions performing various actions. These functions take a pointer to a structure as returned by *dbopen*, and sometimes one or more pointers to key/data structures and a flag value.

type The type of the underlying access method (and file format).

close A pointer to a routine to flush any cached information to disk, free any allocated resources, and close the underlying file(s). Since key/data pairs may be cached in memory, failing to sync the file with a *close* or *sync* function may result in inconsistent or lost information. *Close* routines return -1 on error (setting *errno*) and 0 on success.

del A pointer to a routine to remove key/data pairs from the database.

The parameter *flag* may be set to the following value:

R_CURSOR
: Delete the record referenced by the cursor. The cursor must have previously been initialized.

Delete routines return -1 on error (setting *errno*), 0 on success, and 1 if the specified *key* was not in the file.

fd
: A pointer to a routine which returns a file descriptor representative of the underlying database. A file descriptor referencing the same file will be returned to all processes which call *dbopen* with the same *file* name. This file descriptor may be safely used as a argument to the *fcntl*(2) and *flock*(2) locking functions. The file descriptor is not necessarily associated with any of the underlying files used by the access method. No file descriptor is available for in memory databases. *Fd* routines return -1 on error (setting *errno*), and the file descriptor on success.

get
: A pointer to a routine which is the interface for keyed retrieval from the database. The address and length of the data associated with the specified *key* are returned in the structure referenced by *data*. *Get* routines return -1 on error (setting *errno*), 0 on success, and 1 if the *key* was not in the file.

put
: A pointer to a routine to store key/data pairs in the database.

The parameter *flag* may be set to one of the following values:

R_CURSOR
: Replace the key/data pair referenced by the cursor. The cursor must have previously been initialized.

R_IAFTER
: Append the data immediately after the data referenced by *key*, creating a new key/data pair. The record number of the appended key/data pair is returned in the *key* structure. (Applicable only to the DB_RECNO access method.)

R_IBEFORE
: Insert the data immediately before the data referenced by *key*, creating a new key/data pair. The record number of the inserted key/data pair is returned in the *key* structure. (Applicable only to the DB_RECNO access method.)

R_NOOVERWRITE
: Enter the new key/data pair only if the key does not previously exist.

R_SETCURSOR
: Store the key/data pair, setting or initializing the position of the cursor to reference it. (Applicable only to the DB_BTREE and DB_RECNO access methods.)

R_SETCURSOR is available only for the DB_BTREE and DB_RECNO access methods because it implies that the keys have an inherent order which does not change.

R_IAFTER and R_IBEFORE are available only for the DB_RECNO access method because they each imply that the access method is able to create new keys. This is only true if the keys are ordered and independent, record numbers for example.

The default behavior of the *put* routines is to enter the new key/data pair, replacing any previously existing key.

Put routines return -1 on error (setting *errno*), 0 on success, and 1 if the R_NOOVERWRITE *flag* was set and the key already exists in the file.

seq
: A pointer to a routine which is the interface for sequential retrieval from the database. The address and length of the key are returned in the structure referenced by *key*, and the address and length of the data are returned in the structure referenced by *data*.

Sequential key/data pair retrieval may begin at any time, and the position of the ‘‘cursor’’ is not affected by calls to the *del*, *get*, *put*, or *sync* routines. Modifications to the database during a sequential scan will be reflected in the scan, i.e. records inserted behind the cursor will not be returned while records inserted in front of the cursor will be returned.

The flag value **must** be set to one of the following values:

R_CURSOR
: The data associated with the specified key is returned. This differs from the *get* routines in that it sets or initializes the cursor to the location of the key as well. (Note, for the DB_BTREE access method, the returned key is not necessarily an exact match for the specified key. The returned key is the smallest key greater than or equal to the specified key, permitting partial key matches and range searches.)

R_FIRST
: The first key/data pair of the database is returned, and the cursor is set or initialized to reference it.

R_LAST
: The last key/data pair of the database is returned, and the cursor is set or initialized to reference it. (Applicable only to the DB_BTREE and DB_RECNO access methods.)

R_NEXT
: Retrieve the key/data pair immediately after the cursor. If the cursor is not yet set, this is the same as the R_FIRST flag.

R_PREV
: Retrieve the key/data pair immediately before the cursor. If the cursor is not yet set, this is the same as the R_LAST flag. (Applicable only to the DB_BTREE and DB_RECNO access methods.)

R_LAST and R_PREV are available only for the DB_BTREE and DB_RECNO access methods because they each imply that the keys have an inherent order which does not change.

Seq routines return -1 on error (setting *errno*), 0 on success and 1 if there are no key/data pairs less than or greater than the specified or current key. If the DB_RECNO access method is being used, and if the database file is a character special file and no complete key/data pairs are currently available, the *seq* routines return 2.

sync
: A pointer to a routine to flush any cached information to disk. If the database is in memory only, the *sync* routine has no effect and will always succeed.

The flag value may be set to the following value:

R_RECNOSYNC
: If the DB_RECNO access method is being used, this flag causes the sync routine to apply to the btree file which underlies the recno file, not the recno file itself. (See the *bfname* field of the *recno*(3) manual page for more information.)

Sync routines return -1 on error (setting *errno*) and 0 on success.

KEY/DATA PAIRS

Access to all file types is based on key/data pairs. Both keys and data are represented by the following data structure:

```
typedef struct {
        void *data;
        size_t size;
} DBT;
```

The elements of the DBT structure are defined as follows:

data
: A pointer to a byte string.

size
: The length of the byte string.

Key and data byte strings may reference strings of essentially unlimited length although any two of them must fit into available memory at the same time. It should be noted that the access methods provide no guarantees about byte string alignment.

ERRORS

The *dbopen* routine may fail and set *errno* for any of the errors specified for the library routines *open*(2) and *malloc*(3) or the following:

[EFTYPE]
: A file is incorrectly formatted.

[EINVAL]
: A parameter has been specified (hash function, pad byte etc.) that is incompatible with the current file specification or which is not meaningful for the function (for example, use of the cursor without prior initialization) or there is a mismatch between the version number of file and the software.

The *close* routines may fail and set *errno* for any of the errors specified for the library routines *close*(2), *read*(2), *write*(2), *free*(3), or *fsync*(2).

The *del*, *get*, *put* and *seq* routines may fail and set *errno* for any of the errors specified for the library routines *read*(2), *write*(2), *free*(3) or *malloc*(3).

The *fd* routines will fail and set *errno* to ENOENT for in memory databases.

The *sync* routines may fail and set *errno* for any of the errors specified for the library routine *fsync*(2).

SEE ALSO

btree(3), *hash*(3), *mpool*(3), *recno*(3)

BUGS

The typedef DBT is a mnemonic for ‘‘data base thang’’, and was used because noone could think of a reasonable name that wasn’t already used.

The file descriptor interface is a kluge and will be deleted in a future version of the interface.

None of the access methods provide any form of concurrent access, locking, or transactions.