NAME

DbMpoolFile − shared memory buffer pool

SYNOPSIS

```
#include <db_cxx.h>

static int
DbMpoolFile::open(DbMpool *mp, char *file, u_int32_t flags, int mode,
        size_t pagesize, DbMpoolFinfo *finfop, DbMpoolFile **mpf);

int
DbMpoolFile::close();

int
DbMpoolFile::get(db_pgno_t *pgnoaddr, u_int32_t flags, void **pagep);

int
DbMpoolFile::put(void *pgaddr, u_int32_t flags);

int
DbMpoolFile::set(void *pgaddr, u_int32_t flags);

int
DbMpoolFile::sync();
```

DESCRIPTION

The DB library is a family of classes that provides a modular programming interface to transactions and record-oriented file access. The library includes support for transactions, locking, logging and file page caching, as well as various indexed access methods. Many of the classes (e.g., the file page caching class) are useful independent of the other DB classes, although some classes are explicitly based on other classes (e.g., transactions and logging). For a general description of the DB package, see *db_intro*(3).

This manual page describes the specific details of the per-file memory pool interface.

The *DbMpool*(3) and *DbMpoolFile*(3) classes are the library interface intended to provide general-purpose, page-oriented buffer management of one or more files. While designed to work with the other Db functions, these functions are also useful for more general purposes. The memory pools (*DbMpool::*'s) are referred to in this document as simply ‘‘pools’’. Pools may be shared between processes. Pools are usually filled by pages from one or more files (*DbMpoolFile*'s). Pages in the pool are replaced in LRU (least-recently-used) order, with each new page replacing the page that has been unused the longest. Pages retrieved from the pool using *DbMpoolFile::get* are ‘‘pinned’’ in the pool, by default, until they are returned to the pool's control using the *DbMpoolFile::put* method.

DbMpoolFile::open

The *DbMpoolFile::open* method opens a file in the pool specified by the *DbMpool* argument, copying the DbMpoolFile pointer representing it into the memory location referenced by *mpf*.

The *file* argument is the name of the file to be opened. If *file* is NULL, a private file is created that cannot be shared with any other process (although it may be shared with other threads).

The *flags* and *mode* arguments specify how files will be opened and/or created when they don't already exist. The flags value is specified by **or**'ing together one or more of the following values:

DB_CREATE
: Create any underlying files, as necessary. If the files do not already exist and the DB_CREATE flag is not specified, the call will fail.

DB_NOMMAP
: Always copy this file into the local cache instead of mapping it into process memory (see the description of the *mp_mmapsize* field of the DbEnv object for further information).

DB_RDONLY
: Open any underlying files for reading only. Any attempt to write the file using the pool functions

will fail, regardless of the actual permissions of the file.

All files created by the method *DbMpoolFile::open* are created with mode *mode* (as described in *chmod*(2)) and modified by the process' umask value at the time of creation (see *umask*(2)). The group ownership of created files is based on the system and directory defaults, and is not further specified by DB.

The *pagesize* argument is the size, in bytes, of the unit of transfer between the application and the pool, although it is not necessarily the unit of transfer between the pool and the source file.

Files opened in the pool may be further configured based on the *finfop* argument to *memp_fopen*, which is a pointer to a structure of type DB_MPOOL_FINFO (typedef'd in <db.h>). No references to the *finfop* structure are maintained by DB, so it may be discarded when the *memp_fopen* function returns. In order to ensure compatibility with future releases of DB, all fields of the DB_MPOOL_FINFO structure that are not explicitly set should be initialized to 0 before the first time the structure is used. Do this by declaring the structure external or static, or by calling the C library routine *bzero*(3) or *memset*(3).

The fields of the DB_MPOOL_FINFO structure used by *DbMpoolFile::open* are described below. If *finfop* is NULL or any of its fields are set to their default value, defaults appropriate for the system are used.

int ftype;
: The *ftype* field should be the same as a *ftype* argument previously specified to the *DbMpool::db_register* method, unless no input or output processing of the file's pages are necessary, in which case it should be 0. (See the description of the *DbMpool::db_register* method for more information.)

DBT *pgcookie;
: The *pgcookie* argument contains the byte string that is passed to the *pgin* and *pgout* functions for this file, if any. If no *pgin* or *pgout* functions are specified, the *pgcookie* field should be NULL. (See the description of the *DbMpool::db_register* method for more information.)

u_int8_t *fileid;
: The *fileid* field is a unique identifier for the file. The mpool functions must be able to uniquely identify files in order that multiple processes sharing a file will correctly share its underlying pages. Normally, the *fileid* field should be NULL and the mpool functions will use the file's device and inode numbers (see *stat*(2)) for this purpose. On some filesystems, (e.g., FAT or NFS) file device and inode numbers are not necessarily unique across system reboots. **Applications wanting to maintain a shared memory buffer pool across system reboots, where the pool contains pages from files stored on such filesystems, must specify a unique file identifier to the** *DbMpoolFile::open* call and each process opening or registering the file must provide the same unique identifier. If the *fileid* field is non-NULL, it must reference a DB_FILE_ID_LEN (as defined in <db_cxx.h>) length array of bytes that will be used to uniquely identify the file. This should not be necessary for most applications. Specifically, it is not necessary if the memory pool is re-instantiated after each system reboot, the application is using the Db access methods instead of calling the pool functions explicitly, or the files in the memory pool are stored on filesystems where the file device and inode numbers do not change across system reboots.

int32_t lsn_offset;
: The *lsn_offset* argument is the zero-based byte offset in the page of the page's log sequence number (LSN), or −1 if no LSN offset is specified. (See the description of the *DbMpool::sync* method for more information.)

u_int32_t clear_len;
: The *clear_len* field is the number of initial bytes in a page that should be set to zero when the page is created as a result of the DB_MPOOL_CREATE or DB_MPOOL_NEW flags being specified to *DbMpoolFile::get*. If *finfop* is NULL or *clear_len* is 0, the entire page is cleared.

The *DbMpoolFile::open* method throws a *DbException*(3) or returns the value of *errno* on failure and 0 on success.

DbMpoolFile::close

The *DbMpoolFile::close* method closes the source file indicated by the DbMpoolFile object. This method does not imply a call to *DbMpoolFile::sync*, i.e. no pages are written to the source file as as a result of calling *DbMpoolFile::close*.

In addition, if the *file* argument to *DbMpoolFile::open* was NULL, any underlying files created for this DbMpoolFile will be removed.

The *DbMpoolFile::close* method throws a *DbException*(3) or returns the value of *errno* on failure and 0 on success.

DbMpoolFile::get

The *DbMpoolFile::get* method copies a pointer to the page with the page number specified by *pgnoaddr*, from the source file specified by the DbMpoolFile object into the memory location referenced by *pagep*. If the page does not exist or cannot be retrieved, *DbMpoolFile::get* will fail.

The returned page is size_t type aligned.

Page numbers begin at 0, e.g., the first page in the file is page number 0, not page number 1.

The *flags* argument is specified by **or**'ing together one or more of the following values:

DB_MPOOL_CREATE

If the specified page does not exist, create it. In this case, the *pgin* method, if specified, is called.

DB_MPOOL_LAST

Return the last page of the source file and copy its page number to the location referenced by *pgnoaddr*.

DB_MPOOL_NEW

Create a new page in the file and copy its page number to the location referenced by *pgnoaddr*. In this case, the *pgin* method, if specified, is not called.

The DB_MPOOL_CREATE, DB_MPOOL_LAST and DB_MPOOL_NEW flags are mutually exclusive.

Created pages have all their bytes set to 0, unless otherwise specified when the file was opened.

All pages returned by *DbMpoolFile::get* will be retained (i.e. ''pinned'') in the pool until a subsequent call to *DbMpoolFile::put*.

The *DbMpoolFile::get* method throws a *DbException*(3) or returns the value of *errno* on failure and 0 on success.

DbMpoolFile::put

The *DbMpoolFile::put* method indicates that the page referenced by *pgaddr* can be evicted from the pool. *Pgaddr* must be an address previously returned by *DbMpoolFile::get*.

The *flags* argument is specified by **or**'ing together one or more of the following values:

DB_MPOOL_CLEAN

Clear any previously set modification information (i.e., don't bother writing the page back to the source file).

DB_MPOOL_DIRTY

The page has been modified and must be written to the source file before being evicted from the pool.

DB_MPOOL_DISCARD

The page is unlikely to be useful in the near future, and should be discarded before other pages in the pool.

The DB_MPOOL_CLEAN and DB_MPOOL_DIRTY flags are mutually exclusive.

The *DbMpoolFile::put* method throws a *DbException*(3) or returns the value of *errno* on failure and 0 on success.

DbMpoolFile::set

The *DbMpoolFile::set* method sets the flags associated with the page referenced by *pgaddr* without unpinning it from the pool. *Pgaddr* must be an address previously returned by *DbMpoolFile::get*. The *flags* argument to *DbMpoolFile::set* is specified by **or**'ing together one or more of the values specified as flags for the *DbMpoolFile::put* call.

The *DbMpoolFile::set* method throws a *DbException*(3) or returns the value of *errno* on failure and 0 on success.

DbMpoolFile::sync

The *DbMpoolFile::sync* method writes all pages associated with the DbMpoolFile object that were marked as modified using *DbMpoolFile::put* or *DbMpoolFile::set*, back to the source file. If any of the modified pages are also pinned (i.e., currently referenced by this or another process) *DbMpoolFile::sync* will ignore them.

The *DbMpoolFile::sync* method throws a *DbException*(3) or returns the value of *errno* on failure, 0 on success, and DB_INCOMPLETE if there were pages which were modified but which *DbMpoolFile::sync* was unable to write.

ERRORS

Methods marked as returning *errno* will, by default, throw an exception that encapsulates the error information. The default error behavior can be changed, see *DbException*(3).

The *DbMpoolFile::open* method may fail and throw a *DbException*(3)

or return *errno* for any of the errors specified for the following DB and library functions: DBmemp->pgin(3), DBmemp->pgout(3), DbLog::compare(3), DbLog::flush(3), close(2), fcntl(2), fflush(3), fsync(2), lseek(2), malloc(3), memcmp(3), memcpy(3), memset(3), mmap(2), open(2), sigfillset(3), sigprocmask(2), stat(2), strcpy(3), strdup(3), strerror(3), strlen(3), time(3), unlink(2), and write(2).

In addition, the *DbMpoolFile::open* method may fail and throw a *DbException*(3) or return *errno* for the following conditions:

[EINVAL]

An invalid flag value or parameter was specified.

The file has already been entered into the pool, and the *pagesize* value is not the same as when the file was entered into the pool, or the length of the file is not zero or a multiple of the *pagesize*.

The DB_RDONLY flag was specified for an in-memory pool.

The *DbMpoolFile::close* method may fail and throw a *DbException*(3)

or return *errno* for any of the errors specified for the following DB and library functions: close(2), fcntl(2), fflush(3), munmap(2), and strerror(3).

The *DbMpoolFile::get* method may fail and throw a *DbException*(3)

or return *errno* for any of the errors specified for the following DB and library functions: DBmemp->pgin(3), DBmemp->pgout(3), DbLog::compare(3), DbLog::flush(3), close(2), fcntl(2), fflush(3), fsync(2), lseek(2), malloc(3), memcmp(3), memcpy(3), memset(3), mmap(2), open(2), read(2), sigfillset(3), sigprocmask(2), stat(2), strcpy(3), strdup(3), strerror(3), strlen(3), time(3), unlink(2), and write(2).

In addition, the *DbMpoolFile::get* method may fail and throw a *DbException*(3) or return *errno* for the following conditions:

[EAGAIN]
The page reference count has overflowed. (This should never happen unless there's a bug in the application.)

[EINVAL]
An invalid flag value or parameter was specified.

The DB_MPOOL_NEW flag was set and the source file was not opened for writing.

The requested page does not exist and DB_MPOOL_CREATE was not set.

More than one of DB_MPOOL_CREATE, DB_MPOOL_LAST and DB_MPOOL_NEW was set.

[ENOMEM]
The cache is full and no more pages will fit in the pool.

The *DbMpoolFile::put* method may fail and throw a *DbException*(3)

or return *errno* for any of the errors specified for the following DB and library functions: DBmemp->pgin(3), DBmemp->pgout(3), DbLog::compare(3), DbLog::flush(3), close(2), fcntl(2), fflush(3), fsync(2), lseek(2), malloc(3), memcmp(3), memcpy(3), memset(3), mmap(2), open(2), sigfillset(3), sigprocmask(2), stat(2), strcpy(3), strdup(3), strerror(3), strlen(3), time(3), unlink(2), and write(2).

In addition, the *DbMpoolFile::put* method may fail and throw a *DbException*(3) or return *errno* for the following conditions:

[EACCES]
The DB_MPOOL_DIRTY flag was set and the source file was not opened for writing.

[EINVAL]
An invalid flag value or parameter was specified.

The *pgaddr* parameter does not reference a page returned by *DbMpoolFile::get*.

More than one of DB_MPOOL_CLEAN and DB_MPOOL_DIRTY was set.

The *DbMpoolFile::set* method may fail and throw a *DbException*(3)

or return *errno* for any of the errors specified for the following DB and library functions: fcntl(2), and fflush(3).

In addition, the *DbMpoolFile::set* method may fail and throw a *DbException*(3) or return *errno* for the following conditions:

[EINVAL]
An invalid flag value or parameter was specified.

The *DbMpoolFile::sync* method may fail and throw a *DbException*(3)

or return *errno* for any of the errors specified for the following DB and library functions: DBmemp->pgin(3), DBmemp->pgout(3), DbLog::compare(3), DbLog::flush(3), close(2), fcntl(2), fflush(3), fsync(2), lseek(2), malloc(3), memcpy(3), memset(3), open(2), qsort(3), realloc(3), sigfillset(3), sigprocmask(2), stat(2), strcpy(3), strdup(3), strerror(3), strlen(3), unlink(2), and write(2).

SEE ALSO

db_archive(1), *db_checkpoint*(1), *db_deadlock*(1), *db_dump*(1), *db_load*(1), *db_recover*(1), *db_stat*(1), *db_intro*(3), *db_internal*(3), *db_thread*(3), *Db*(3), *Dbc*(3), *DbEnv*(3), *DbException*(3), *DbInfo*(3), *DbLock*(3), *DbLockTab*(3), *DbLog*(3), *DbLsn*(3), *DbMpool*(3), *DbMpoolFile*(3), *Dbt*(3), *DbTxn*(3), *DbTxnMgr*(3)