NAME

DbLog − log management class

SYNOPSIS

```
#include <db_cxx.h>

static int
DbLog::open(const char *dir,
        u_int32_t flags, int mode, DbEnv *dbenv, DbLog **regionp);

int
DbLog::close();

int
DbLog::flush(const DbLsn *lsn);

int
DbLog::get(DbLsn *lsn, Dbt *data, u_int32_t flags);

static int
DbLog::compare(const DbLsn *lsn0, const DbLsn *lsn1);

int
DbLog::file(const DbLsn *lsn, char *namep, size_t len);

int
DbLog::put(DbLsn *lsn, const Dbt *data, u_int32_t flags);

static int
DbLog::unlink(const char *dir, int force, DbEnv *);

int
DbLog::archive(char **list[],
        u_int32_t flags, void *(*db_malloc)(size_t));

int
DbLog::db_register(const Db *dbp,
        const char *name, DBTYPE type, u_int32_t *fidp);

int
DbLog::db_unregister(u_int32_t fid);

int
DbLog::stat(DB_LOG_STAT **spp, void *(*db_malloc)(size_t));
```

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 log class.

This class provide a general-purpose logging facility sufficient for transaction management. Logs can be shared by multiple processes.

The DB transaction log is represented by a directory containing a set of files. The log is a record-oriented, append-only file, with records identified and accessed via *DbLsn*'s (database log sequence numbers).

DbLsn's are returned on each *DbLog::put* operation, and only those DbLsn's returned by *DbLog::put* can later be used to retrieve records from the log.

DbLog::open

The *DbLog::open* method copies a pointer, to the log identified by the **directory** *dir*, into the memory location referenced by *regionp*.

If the *dbenv* argument to *DbLog::open* was initialized using *DbEnv::appinit*, *dir* is interpreted as described by *DbEnv*(3).

Otherwise, if *dir* is not NULL, it is interpreted relative to the current working directory of the process. If *dir* is NULL, the following environment variables are checked in order: ‘‘TMPDIR’’, ‘‘TEMP’’, and ‘‘TMP’’. If one of them is set, log files are created relative to the directory it specifies. If none of them are set, the first possible one of the following directories is used: */var/tmp*, */usr/tmp*, */temp*, */tmp*, *C:/temp* and *C:/tmp*.

All files associated with the log are created in this directory. This directory must already exist when DbLog::open is called. If the log already exists, the process must have permission to read and write the existing files. If the log does not already exist, it is optionally created and initialized.

If the log region is being created and log files are already present, the log files are ‘‘recovered’’ and subsequent log writes are appended to the end of the log.

The log is stored in one or more files in the specified directory. Each file is named using the format

```
log.NNNNN
```

where ‘‘NNNNN’’ is the sequence number of the file within the log.

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_THREAD
: Cause the DbLog handle returned by the *DbLog::open* method to be useable by multiple threads within a single address space, i.e., to be ‘‘free-threaded’’.

All files created by the log subsystem 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 logging subsystem is configured based on which set methods have been used. It is expected that applications will use a single DbEnv object as the argument to all of the subsystems in the DB package. The fields of the DbEnv object used by *DbLog::open* are described below. As references to the DbEnv object may be maintained by *DbLog::open*, it is necessary that the DbEnv object and memory it references be valid until the object is destroyed. Any of the DbEnv fields that are not explicitly set will default to appropriate values.

The following fields in the DbEnv object may be initialized, using the appropriate set method, before calling *DbLog::open*:

```
void *(*db_errcall)(char *db_errpfx, char *buffer);
FILE *db_errfile;
const char *db_errpfx;
class ostream *db_error_stream;
int db_verbose;
```

The error fields of the DbEnv behave as described for *DbEnv*(3).

u_int32_t lg_max;

The maximum size of a single file in the log. Because DbLsn file offsets are unsigned 4-byte values, *lg_max* may not be larger than the maximum unsigned 4-byte value.

If *lg_max* is 0, a default value is used.

See the section "LOG FILE LIMITS" below, for further information.

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

DbLog::close

The *DbLog::close* method closes the log associated with the DbLog object.

In addition, if the *dir* argument to *DbLog::open* was NULL and *dbenv* was not initialized using *DbEnv::appinit*, all files created for this shared region will be removed, as if *DbLog::unlink* were called.

When multiple threads are using the DbLog handle concurrently, only a single thread may call the *DbLog::close* method.

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

DbLog::flush

The *DbLog::flush* method guarantees that all log records whose DbLsns are less than or equal to the *lsn* parameter have been written to disk. If *lsn* is NULL, all records in the log are flushed.

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

DbLog::get

The *DbLog::get* method implements a cursor inside of the log, retrieving records from the log according to the *lsn* and *flags* parameters.

The data field of the *data* object is set to the record retrieved and the size field indicates the number of bytes in the record. See *Dbt*(3) for a description of other fields in the *data* object. **When multiple threads are using the returned DbLog handle concurrently, either the DB_DBT_MALLOC or DB_DBT_USERMEM flags must be specified for any Dbt used for data retrieval.**

The *flags* parameter must be set to exactly one of the following values:

DB_CHECKPOINT

The last record written with the DB_CHECKPOINT flag specified to the *DbLog::put* method is returned in the *data* argument. The *lsn* argument is overwritten with the DbLsn of the record returned. If no record has been previously written with the DB_CHECKPOINT flag specified, the first record in the log is returned.

If the log is empty the *DbLog::get* method will return DB_NOTFOUND.

DB_FIRST

The first record from any of the log files found in the log directory is returned in the *data* argument. The *lsn* argument is overwritten with the DbLsn of the record returned.

If the log is empty the *DbLog::get* method will return DB_NOTFOUND.

DB_LAST

The last record in the log is returned in the *data* argument. The *lsn* argument is overwritten with the DbLsn of the record returned.

If the log is empty, the *DbLog::get* method will return DB_NOTFOUND.

DB_NEXT

The current log position is advanced to the next record in the log and that record is returned in the *data* argument. The *lsn* argument is overwritten with the DbLsn of the record returned.

If the pointer has not been initialized via DB_FIRST, DB_LAST, DB_SET, DB_NEXT, or DB_PREV, *DbLog::get* will return the first record in the log. If the last log record has already been returned or the log is empty, the *DbLog::get* method will return DB_NOTFOUND.

If the log was opened with the DB_THREAD flag set, calls to *DbLog::get* with the DB_NEXT flag set will return EINVAL.

DB_PREV

The current log position is moved to the previous record in the log and that record is returned in the *data* argument. The *lsn* argument is overwritten with the DbLsn of the record returned.

If the pointer has not been initialized via DB_FIRST, DB_LAST, DB_SET, DB_NEXT, or DB_PREV, *DbLog::get* will return the last record in the log. If the first log record has already been returned or the log is empty, the *DbLog::get* method will return DB_NOTFOUND.

If the log was opened with the DB_THREAD flag set, calls to *DbLog::get* with the DB_PREV flag set will return EINVAL.

DB_CURRENT

Return the log record currently referenced by the log.

If the log pointer has not been initialized via DB_FIRST, DB_LAST, DB_SET, DB_NEXT, or DB_PREV, or if the log was opened with the DB_THREAD flag set, *DbLog::get* will return EINVAL.

DB_SET

Retrieve the record specified by the *lsn* argument. If the specified DbLsn is invalid (e.g., does not appear in the log) *DbLog::get* will return EINVAL.

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

DbLog::compare

The *DbLog::compare* method allows the caller to compare two DbLsn's. *DbLog::compare* returns 0 if the two DbLsn's are equal, 1 if *lsn0* is greater than *lsn1*, and -1 if *lsn0* is less than *lsn1*.

DbLog::file

The *DbLog::file* method maps DbLsn's to file names. The *DbLog::file* method copies the name of the file containing the record named by *lsn* into the memory location referenced by *namep*. (This mapping of DbLsn to file is needed for database administration. For example, a transaction manager typically records the earliest DbLsn needed for restart, and the database administrator may want to archive log files to tape when they contain only DbLsn's before the earliest one needed for restart.)

The *len* argument is the length of the *namep* buffer in bytes. If *namep* is too short to hold the file name, *DbLog::file* will return ENOMEM. Note, as described above, log file names are quite short, on the order of 10 characters.

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

DbLog::put

The *DbLog::put* method appends records to the log. The DbLsn of the put record is returned in the *lsn* parameter. The *flags* parameter may be set to one of the following values:

DB_CHECKPOINT

The log should write a checkpoint record, recording any information necessary to make the log structures recoverable after a crash.

DB_CURLSN
The DB_LSN of the next record to be put is returned in the *lsn* parameter.

DB_FLUSH
The log is forced to disk after this record is written, guaranteeing that all records with DbLsns less than or equal to the one being put are on disk before this method returns (this function is most often used for a transaction commit, see *Db::Txn* in Db (3)).

The caller is responsible for providing any necessary structure to *data*. (For example, in a write-ahead logging protocol, the application must understand what part of *data* is an operation code, what part is redo information, and what part is undo information. In addition, most transaction managers will store in *data* the DbLsn of the previous log record for the same transaction, to support chaining back through the transaction's log records during undo.)

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

DbLog::unlink

The *DbLog::unlink* method destroys the log region identified by the directory *dir*, removing all files used to implement the log region. (The directory *dir* is not removed.) If there are processes that have called *DbLog::open* without calling *DbLog::close* (i.e., there are processes currently using the log region), *DbLog::unlink* will fail without further action, unless the force flag is set, in which case *DbLog::unlink* will attempt to remove the log region files regardless of any processes still using the log region.

The result of attempting to forcibly destroy the region when a process has the region open is unspecified. Processes using a shared memory region maintain an open file descriptor for it. On UNIX systems, the region removal should succeed and processes that have already joined the region should continue to run in the region without change, however processes attempting to join the log region will either fail or attempt to create a new region. On other systems, e.g., WNT, where the *unlink*(2) system call will fail if any process has an open file descriptor for the file, the region removal will fail.

In the case of catastrophic or system failure, database recovery must be performed (see *db_recover*(1) or the DB_RECOVER and DB_RECOVER_FATAL flags to *DbEnv::appinit*(3)). Alternatively, if recovery is not required because no database state is maintained across failures, it is possible to clean up a log region by removing all of the files in the directory specified to the *DbLog::open* method, as log region files are never created in any directory other than the one specified to *DbLog::open*. Note, however, that this has the potential to remove files created by the other DB subsystems in this database environment.

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

DbLog::archive

The *DbLog::archive* method creates a NULL-terminated array of log or database file names and copies a pointer to them into the user-specified memory location *list*.

By default, *DbLog::archive* returns the names of all of the log files that are no longer in use (e.g., no longer involved in active transactions), and that may be archived for catastrophic recovery and then removed from the system. If there were no file names to return, *list* will be set to NULL.

Arrays of log file names are created in allocated memory. If *db_malloc* is non-NULL, it is called to allocate the memory, otherwise, the library function *malloc*(3) is used. The function *db_malloc* must match the calling conventions of the *malloc*(3) library routine. Regardless, the caller is responsible for deallocating the returned memory. To deallocate the returned memory, free each returned memory pointer; pointers inside the memory do not need to be individually freed.

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

DB_ARCH_ABS
All pathnames are returned as absolute pathnames, instead of relative to the database home directory.

DB_ARCH_DATA
Return the database files that need to be archived in order to recover the database from catastrophic failure. If any of the database files have not been accessed during the lifetime of the current log files, *DbLog::archive* will not include them in this list. It is also possible that some of the files referenced in the log have since been deleted from the system.

DB_ARCH_LOG
Return all the log file names regardless of whether or not they are in use.

The DB_ARCH_DATA and DB_ARCH_LOG flags are mutually exclusive.

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

The *DbLog::archive* method is based on the C *log_archive* function, which is the underlying function used by the *db_archive*(1) utility. See the source code for the *db_archive* utility for an example of using *log_archive* in a UNIX environment. See the *db_archive*(1) manual page for more information on database archival procedures.

DbLog::db_register
The *DbLog::db_register* method registers a file name with the log manager and copies a file identification number into the memory location referenced by *fidp*. This file identification number should be used in all subsequent log messages that refer to operations on this file. The log manager records all file name to file identification number mappings at each checkpoint so that a recovery process can identify the file to which a record in the log refers.

The *DbLog::db_register* method is called when an access method registers the open of a file. The *dbp* parameter should be a pointer to the Db object which is being returned by the access method.

The *type* parameter should be one of the Db types specified in *Db::open (see*Db*(3)),* e.g., DB_HASH.

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

DbLog::db_unregister
The *DbLog::db_unregister* method disassociates the file name to file identification number mapping for the file identification number specified by the *fid* parameter. The file identification number may then be reused.

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

DbLog::stat
The *DbLog::stat* function creates a statistical structure and copies a pointer to it into the user-specified memory location.

Statistical structures are created in allocated memory. If *db_malloc* is non-NULL, it is called to allocate the memory, otherwise, the library function *malloc*(3) is used. The function *db_malloc* must match the calling conventions of the *malloc*(3) library routine. Regardless, the caller is responsible for deallocating the returned memory. To deallocate the returned memory, free each returned memory pointer; pointers inside the memory do not need to be individually freed. The log statistics are stored in a structure of type DB_LOG_STAT (typedef'd in <db.h>). The following DB_LOG_STAT fields will be filled in:

u_int32_t st_magic;
The magic number that identifies a file as a log file.

u_int32_t st_version;
The version of the log file type.

u_int32_t st_refcnt;
The number of references to the region.

u_int32_t st_regsize;
The size of the region.

u_int32_t st_mode;
: The mode of any created log files.

u_int32_t st_lg_max;
: The maximum size of any individual file comprising the log.

u_int32_t st_written;
: The total number of bytes written to this log.

u_int32_t st_written_chkpt;
: The number of bytes written since the last checkpoint.

u_int32_t st_region_wait;
: The number of times that a process was able to obtain the region lock without waiting.

u_int32_t st_region_nowait;
: The number of times that a process was forced to wait before obtaining the region lock.

LOG FILE LIMITS

Log file sizes impose a time limit on the length of time a database may be accessed under transaction protection, before it needs to be dumped and reloaded (see *db_dump(3)* and *db_load(3)).* Unfortunately, the limits are potentially difficult to calculate.

The log file name consists of "log." followed by 5 digits, resulting in a maximum of 99,999 log files. Consider an application performing 600 transactions per second, for 15 hours a day, logged into 10Mb log files, where each transaction is logging approximately 100 bytes of data. The calculation:

```
(10 * 2^20 * 99999) /
        (600 * 60 * 60 * 15 * 100) = 323.63
```

indicates that the system will run out of log file space in roughly 324 days. If we increase the maximum size of the files from 10Mb to 100Mb, the same calculation indicates that the application will run out of log file space in roughly 9 years.

There is no way to reset the log file name space in Berkeley DB. If your application is reaching the end of its log file name space, you should:

1. Archive your databases as if to prepare for catastrophic failure (see *db_archive*(1) for more information).

2. Dump and re-load **all** your databases (see *db_dump*(1) and *db_load*(1) for more information).

3. Remove all of the log files from the database environment (see *db_archive*(1) for more information).

4. Restart your applications.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of *db_log*:

DB_HOME
: If the *dbenv* argument to *DbLog::open* was initialized using *db_appinit*, the environment variable DB_HOME may be used as the path of the database home for the interpretation of the *dir* argument to *DbLog::open*, as described in *db_appinit*(3). Specifically, *DbLog::open* is affected by the configuration string value of DB_LOG_DIR.

TMPDIR
: If the *dbenv* argument to *DbLog::open* was NULL or not initialized using *db_appinit*, the environment variable TMPDIR may be used as the directory in which to create the log, as described in the *DbLog::open* section above.

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 *DbLog::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: DbLog::close(3), DbLog::unlink(3), atoi(3), close(2), db_version(3), fcntl(2), fflush(3), lseek(2), malloc(3), memcpy(3), memset(3), mmap(2), munmap(2), open(2), opendir(3), read(2), readdir(3), realloc(3), sigfillset(3), sigprocmask(2), stat(2), strchr(3), strcpy(3), strdup(3), strerror(3), strlen(3), strncmp(3), unlink(2), and write(2).

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

[EAGAIN]
The shared memory region was locked and (repeatedly) unavailable.

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

The DB_THREAD flag was specified and spinlocks are not implemented for this architecture.

The specified file size was too large.

The *DbLog::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 *DbLog::flush* 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), fsync(2), lseek(2), malloc(3), memcpy(3), memset(3), open(2), sigfillset(3), sigprocmask(2), stat(2), strcpy(3), strdup(3), strerror(3), strlen(3), unlink(2), and write(2).

In addition, the *DbLog::flush* 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 *DbLog::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: atoi(3), close(2), fcntl(2), fflush(3), lseek(2), malloc(3), memcpy(3), memset(3), open(2), opendir(3), read(2), readdir(3), realloc(3), sigfillset(3), sigprocmask(2), stat(2), strchr(3), strcpy(3), strdup(3), strerror(3), strlen(3), strncmp(3), and unlink(2).

In addition, the *DbLog::get* 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 DB_FIRST flag was specified and no log files were found.

The *DbLog::file* 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), malloc(3), memcpy(3), memset(3), open(2), sigfillset(3), sigprocmask(2), stat(2), strcpy(3), strdup(3), strerror(3), strlen(3), and unlink(2).

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

[ENOMEM]
The supplied buffer was too small to hold the log file name.

The *DbLog::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: close(2), fcntl(2), fflush(3), fsync(2), lseek(2), malloc(3), memcpy(3), memset(3), 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 *DbLog::put* 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 record to be logged is larger than the maximum log record.

The *DbLog::unlink* 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), malloc(3), memcpy(3), memset(3), mmap(2), munmap(2), open(2), sigfillset(3), sigprocmask(2), stat(2), strcpy(3), strdup(3), strerror(3), strlen(3), and unlink(2).

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

[EBUSY]
The shared memory region was in use and the force flag was not set.

The *DbLog::archive* method may fail and throw a *DbException*(3)

or return *errno* for any of the errors specified for the following DB and library functions: DbLog::compare(3), DbLog::get(3), close(2), fcntl(2), fflush(3), getcwd(3), malloc(3), memcpy(3), memset(3), open(2), qsort(3), realloc(3), sigfillset(3), sigprocmask(2), stat(2), strchr(3), strcmp(3), strcpy(3), strdup(3), strerror(3), strlen(3), and unlink(2).

In addition, the *DbLog::archive* 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 log was corrupted.

The *DbLog::db_register* 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), fsync(2), lseek(2), malloc(3), memcmp(3), memcpy(3), memset(3), open(2), realloc(3), sigfillset(3), sigprocmask(2), stat(2), strcpy(3), strdup(3), strerror(3), strlen(3), time(3), unlink(2), and write(2).

In addition, the *DbLog::db_register* 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 *DbLog::db_unregister* 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), fsync(2), lseek(2), malloc(3), memcpy(3), memset(3), 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 *DbLog::db_unregister* method may fail and throw a *DbException*(3) or return *errno* for the following conditions:

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

BUGS

The log files are not machine architecture independent. Specifically, their metadata are not stored in a fixed byte order.

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)