NAME

db_log − log management functions

SYNOPSIS

```
#include <db.h>

int
log_open(const char *dir,
        u_int32_t flags, int mode, DB_ENV *dbenv, DB_LOG **regionp);

int
log_close(DB_LOG *logp);

int
log_flush(DB_LOG *logp, const DB_LSN *lsn);

int
log_get(DB_LOG *logp, DB_LSN *lsn, DBT *data, u_int32_t flags);

int
log_compare(const DB_LSN *lsn0, const DB_LSN *lsn1);

int
log_file(DB_LOG *logp, const DB_LSN *lsn, char *namep, size_t len);

int
log_put(DB_LOG *logp, DB_LSN *lsn, const DBT *data, u_int32_t flags);

int
log_unlink(const char *dir, int force, DB_ENV *);

int
log_archive(DB_LOG *logp,
        char **list[], u_int32_t flags, void *(*db_malloc)(size_t));

int
log_register(DB_LOG *logp,
        const DB *dbp, const char *name, DBTYPE type, u_int32_t *fidp);

int
log_unregister(DB_LOG *logp, u_int32_t fid);

int
log_stat(DB_LOG *logp, DB_LOG_STAT **spp, void *(*db_malloc)(size_t));
```

DESCRIPTION

The DB library is a family of groups of functions 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 functional groups (e.g., the file page caching functions) are useful independent of the other DB functions, although some functional groups are explicitly based on other functional groups (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 manager.

These functions 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 *DB_LSN*'s (database log sequence numbers).

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

log_open

The *log_open* function copies a pointer, to the log identified by the **directory** *dir*, into the memory location referenced by *regionp*.

If the *dbenv* argument to *log_open* was initialized using *db_appinit*, *dir* is interpreted as described by *db_appinit*(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

log_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 DB_LOG handle returned by the *log_open* function 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 the *dbenv* argument to *log_open*, which is a pointer to a structure of type DB_ENV (typedef’d in <db.h>). Applications will normally use the same DB_ENV structure (initialized by *db_appinit*(3)), as an argument to all of the subsystems in the DB package.

References to the DB_ENV structure are maintained by DB, so it may not be discarded until the last close function, corresponding to an open function for which it was an argument, has returned. In order to ensure compatibility with future releases of DB, all fields of the DB_ENV 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_ENV structure used by *log_open* are described below. If *dbenv* is NULL or any of its fields are set to 0, defaults appropriate for the system are used where possible.

The following fields in the DB_ENV structure may be initialized before calling *log_open*:

void *(*db_errcall)(char *db_errpfx, char *buffer);
FILE *db_errfile;
const char *db_errpfx;

int db_verbose;

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

u_int32_t lg_max;

The maximum size of a single file in the log. Because DB_LSN 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 *log_open* function returns the value of *errno* on failure and 0 on success.

log_close

The *log_close* function closes the log specified by the *logp* argument.

In addition, if the *dir* argument to *log_open* was NULL and *dbenv* was not initialized using *db_appinit*, all files created for this shared region will be removed, as if *log_unlink* were called.

When multiple threads are using the DB_LOG handle concurrently, only a single thread may call the *log_close* function.

The *log_close* function returns the value of *errno* on failure and 0 on success.

log_flush

The *log_flush* function guarantees that all log records whose LSNs 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 *log_flush* function returns the value of *errno* on failure and 0 on success.

log_get

The *log_get* function 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* structure is set to the record retrieved and the size field indicates the number of bytes in the record. See *db_dbt*(3) for a description of other fields in the *data* structure. **When multiple threads are using the returned DB_LOG 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 *log_put* function is returned in the *data* argument. The *lsn* argument is overwritten with the DB_LSN 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 *log_get* function 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 DB_LSN of the record returned.

If the log is empty the *log_get* function 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 DB_LSN of the record returned.

If the log is empty, the *log_get* function 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 DB_LSN of the record returned.

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

If the log was opened with the DB_THREAD flag set, calls to *log_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 DB_LSN of the record returned.

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

If the log was opened with the DB_THREAD flag set, calls to *log_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, *log_get* will return EINVAL.

DB_SET

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

Otherwise, the *log_get* function returns the value of *errno* on failure and 0 on success.

log_compare

The *log_compare* function allows the caller to compare two DB_LSN's. *Log_compare* returns 0 if the two DB_LSN's are equal, 1 if *lsn0* is greater than *lsn1*, and -1 if *lsn0* is less than *lsn1*.

log_file

The *log_file* function maps DB_LSN's to file names. The *log_file* function copies the name of the file containing the record named by *lsn* into the memory location referenced by *namep*. (This mapping of DB_LSN to file is needed for database administration. For example, a transaction manager typically records the earliest DB_LSN needed for restart, and the database administrator may want to archive log files to tape when they contain only DB_LSN'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, *log_file* will return ENOMEM. Note, as described above, log file names are quite short, on the order of 10 characters.

The *log_file* function returns the value of *errno* on failure and 0 on success.

log_put

The *log_put* function appends records to the log. The DB_LSN 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 DB_LSNs less than or equal to the one being put are on disk before this function returns (this function is most often used for a transaction commit, see *db_txn*(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 DB_LSN of the previous log record for the same transaction, to support chaining back through the transaction's log records during undo.)

The *log_put* function returns the value of *errno* on failure and 0 on success.

log_unlink

The *log_unlink* function destroys the log region identified by the directory *dir*, removing all files used to implement the log region. (The log files themselves and the directory *dir* are not removed.) If there are processes that have called *log_open* without calling *log_close* (i.e., there are processes currently using the log region), *log_unlink* will fail without further action, unless the force flag is set, in which case *log_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 *db_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 *log_open* function, as log region files are never created in any directory other than the one specified to *log_open*. Note, however, that this has the potential to remove files created by the other DB subsystems in this database environment.

The *log_unlink* function returns the value of *errno* on failure and 0 on success.

log_archive

The *log_archive* function 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, *log_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, *log_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 *log_archive* function returns the value of *errno* on failure and 0 on success.

The *log_archive* function 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.

log_register

The *log_register* function 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 *log_register* function is called when an access method registers the open of a file. The *dbp* parameter should be a pointer to the DB structure which is being returned by the access method.

The *type* parameter should be one of the DB types specified in *db_open*(3), e.g., DB_HASH.

The *log_register* function returns the value of *errno* on failure and 0 on success.

log_unregister

The *log_unregister* function 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 *log_unregister* function returns the value of *errno* on failure and 0 on success.

log_stat

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

Statistical structure 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 region 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.

int 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_w_mbytes;
  The number of megabytes written to this log.

u_int32_t st_w_bytes;
  The number of bytes over and above *st_w_mbytes* written to this log.

u_int32_t st_wc_mbytes;
  The number of megabytes written to this log since the last checkpoint.

u_int32_t st_wc_bytes;
  The number of bytes over and above *st_wc_mbytes* written to this log since the last checkpoint.

u_int32_t st_cur_file;
: The current log file number.

u_int32_t st_cur_offset;
: The byte offset in the current log file.

u_int32_t st_region_wait;
: The number of times that a thread of control was forced to wait before obtaining the region lock.

u_int32_t st_region_nowait;
: The number of times that a thread of control was able to obtain the region lock without waiting.

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 *log_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 *log_open*, as described in *db_appinit*(3). Specifically, *log_open* is affected by the configuration string value of DB_LOG_DIR.

TMPDIR
: If the *dbenv* argument to *log_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 *log_open* section above.

ERRORS

The *log_open* function may fail and return *errno* for any of the errors specified for the following DB and library functions: atoi(3), close(2), db_version(3), fcntl(2), fflush(3), log_close(3), log_unlink(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 *log_open* function may fail and 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 *log_close* function may fail and 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 *log_flush* function may fail and 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 *log_flush* function may fail and return *errno* for the following conditions:

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

The *log_get* function may fail and 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 *log_get* function may fail and 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 *log_file* function may fail and 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 *log_file* function may fail and return *errno* for the following conditions:

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

The *log_put* function may fail and 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 *log_put* function may fail and 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 *log_unlink* function may fail and 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 *log_unlink* function may fail and return *errno* for the following conditions:

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

The *log_archive* function may fail and return *errno* for any of the errors specified for the following DB and library functions: close(2), fcntl(2), fflush(3), getcwd(3), log_compare(3), log_get(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 *log_archive* function may fail and return *errno* for the following conditions:

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

The log was corrupted.

The *log_register* function may fail and 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 *log_register* function may fail and return *errno* for the following conditions:

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

The *log_unregister* function may fail and 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 *log_unregister* function may fail and return *errno* for the following conditions:

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

The *log_stat* function may fail and return *errno* for any of the errors specified for the following DB and library functions: fcntl(2), and malloc(3).

BUGS

The log files are not machine architecture independent. Specifically, log file metadata is 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_appinit*(3), *db_cursor*(3), *db_dbm*(3), *db_internal*(3), *db_lock*(3), *db_log*(3), *db_mpool*(3), *db_open*(3), *db_thread*(3), *db_txn*(3)