NAME

DbEnv − Db initialization and environment options

SYNOPSIS

```
#include <db_cxx.h>

DbEnv::DbEnv(const char *db_home, char *const *db_config, int flags);
DbEnv::DbEnv();
DbEnv::˜DbEnv();

int DbEnv::appinit(const char *db_home,
        char *const *db_config, int flags);

int DbEnv::get_lorder() const;
void DbEnv::set_lorder(int);

typedef void (*db_errcall_fcn)(const char *, char *);
db_errcall_fcn DbEnv::get_errcall() const;
void DbEnv::set_errcall(db_errcall_fcn);

FILE DbEnv::*get_errfile() const;
void DbEnv::set_errfile(FILE *);

const char DbEnv::*get_errpfx() const;
void DbEnv::set_errpfx(const char *);

int DbEnv::get_verbose() const;
void DbEnv::set_verbose(int);

char DbEnv::*get_home() const;
void DbEnv::set_home(char *);

char DbEnv::*get_log_dir() const;
void DbEnv::set_log_dir(char *);

char DbEnv::*get_tmp_dir() const;
void DbEnv::set_tmp_dir(char *);

char DbEnv::**get_data_dir() const;
void DbEnv::set_data_dir(char **);

DbLockTab DbEnv::*get_lk_info() const;

u_int8_t DbEnv::*get_lk_conflicts() const;
void DbEnv::set_lk_conflicts(u_int8_t *);

int DbEnv::get_lk_modes() const;
void DbEnv::set_lk_modes(int);

unsigned int DbEnv::get_lk_max() const;
void DbEnv::set_lk_max(unsigned int);

u_int32_t DbEnv::get_lk_detect() const;
void DbEnv::set_lk_detect(u_int32_t);

typedef int (*db_yield_fcn) (void);
db_yield_fcn DbEnv::get_yield() const;
void DbEnv::set_yield(db_yield_fcn);

DbLog DbEnv::*get_lg_info() const;
```

u_int32_t DbEnv::get_lg_max() const;
void DbEnv::set_lg_max(u_int32_t);

DbMpool DbEnv::*get_mp_info() const;

size_t DbEnv::get_mp_mmapsize() const;
void DbEnv::set_mp_mmapsize(size_t);

size_t DbEnv::get_mp_size() const;
void DbEnv::set_mp_size(size_t);

DbTxnMgr DbEnv::*get_tx_info() const;

unsigned int DbEnv::get_tx_max() const;
void DbEnv::set_tx_max(unsigned int);

typedef int (*tx_recover_fcn)(DB_LOG *, DBT *, DB_LSN *, int, void *);
tx_recover_fcn DbEnv::get_tx_recover() const;
void DbEnv::set_tx_recover(tx_recover_fcn);

u_int32_t DbEnv::get_flags() const;
void DbEnv::set_flags(u_int32_t);

enum ErrorModel { Exception, ErrorReturn };
void DbEnv::set_error_model(ErrorModel);
ErrorModel DbEnv::get_error_model() const;

class ostream* DbEnv::get_error_stream() const;
void DbEnv::set_error_stream(class ostream*);

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).

The DbEnv class provides simple access to an underlying data structure, whose elements can be examined or changed using the *set_* or *get_* methods. The remainder of the manual page sometimes refers to these accesses using the underlying name, e.g., simply *lorder* instead of *get_lorder* and *set_lorder*. The constructors set all elements of the underlying structure to zero. The constructor with three arguments has the effect of calling *DbEnv::appinit* immediately to initialize the application with default parameters. To delay the initialization, use the default constructor. The various set_ methods can then be used to initialize the DbEnv, and finally, a call to *DbEnv::appinit* should be made to initialize DB.

Once the DB environment has been initialized by a call to *DbEnv::appinit*, no set methods other than *set_errpfx* should be called.

appinit

The *appinit* method provides a simple way to initialize and configure the Db environment. It is not necessary that it be called, but it provides a method of creating a consistent environment for processes using one or more of the features of Db.

The *db_home* and *db_config* arguments to *appinit* are described in the section below entitled ‘‘FILE NAMING’’.

The *flags* argument specifies the subsystems that are initialized and how the environment affects Db file naming, among other things. The *flags* value is specified by **or**’ing together one or more of the following values:

DB_CREATE
: Cause subsystems to create any underlying files, as necessary. (See *Db*(3), *DbLockTab*(3), *DbLog*(3), *DbMpool*(3) and *DbTxnMgr*(3) for more information.)

DB_INIT_LOCK
: Initialize the lock subsystem; see *DbLockTab*(3). This subsystem should be used when multiple processes or threads are going to be reading and writing a Db database, so that they do not interfere with each other. If all threads are accessing the database(s) read-only, then locking is unnecessary. When the DB_INIT_LOCK flag is specified, it is usually necessary to run the deadlock detector, *db_deadlock*(1), as well.

DB_INIT_LOG
: Initialize the log subsystem; see *DbLog*(3). This subsystem is used when recovery from application or system failure is important.

DB_INIT_MPOOL
: Initialize the mpool subsystem; see *DbMpool*(3). This subsystem is used whenever the application is using the Db access methods for any purpose.

DB_INIT_TXN
: Initialize the transaction subsystem; see *DbTxn*(3). This subsystem is used when atomicity of multiple operations and recovery are important. The DB_INIT_TXN flag implies the DB_INIT_LOG flag.

DB_MPOOL_PRIVATE
: Create a private memory pool (see *DbMpool*(3) for further information). Ignored unless DB_INIT_MPOOL is also specified.

DB_NOMMAP
: Do not map any files within this environment (see *DbMpool*(3) for further information). Ignored unless DB_INIT_MPOOL is also specified.

DB_RECOVER
: Run normal recovery on this environment before opening it for normal use. If this flag is set, the DB_CREATE, DB_INIT_TXN, and DB_INIT_LOG flags must also be set since the regions will be removed and recreated.

  The *DbEnv::appinit* function returns EINVAL if DB_RECOVER is specified and no log files exist. For this reason, applications should first verify that recovery is needed before specifying the DB_RECOVER flag. For further information, consult the man page for *db_recover*(1).

DB_RECOVER_FATAL
: Run catastrophic recovery on this environment before opening it for normal use. If this flag is set, the DB_CREATE, DB_INIT_TXN, and DB_INIT_LOG flags must also be set since the regions will be removed and recreated.

  The *DbEnv::appinit* function returns EINVAL if DB_RECOVER_FATAL is specified and no log files exist. For this reason, applications should first verify that recovery is needed before specifying the DB_RECOVER_FATAL flag. For further information, consult the man page for *db_recover*(1).

DB_THREAD
: Ensure that handles returned by the Db subsystems are useable by multiple threads within a single process, i.e., that the system is "free-threaded". (See *DbLockTab*(3), *DbLog*(3), *DbMpool*(3), *Db::open*(3) and *DbTxn*(3) for more information.)

DB_TXN_NOSYNC
: On transaction commit, do not synchronously flush the log (see *DbTxn*(3) for further information). Ignored unless DB_INIT_TXN is also specified.

DB_USE_ENVIRON
: The Db process' environment may be permitted to specify information to be used when naming files (see the section entitled "FILE NAMING" below). As permitting users to specify which files are used can create security problems, environment information will be used in file naming for all users only if the DB_USE_ENVIRON flag is set.

DB_USE_ENVIRON_ROOT
: The Db process' environment may be permitted to specify information to be used when naming files (see the section entitled "FILE NAMING" below). As permitting users to specify which files are used can create security problems, if the DB_USE_ENVIRON_ROOT flag is set, environment information will be used for file naming only for users with a user-ID matching that of the superuser (specifically, users for whom the *getuid* system call returns the user-ID 0).

The Db environment 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 *appinit* are described below. As references to the DbEnv object may be maintained by *appinit*, it is necessary that the DbEnv object and memory it references be valid until the object is destroyed. The *dbenv* argument may not be NULL. If any of the fields of the *dbenv* are set to 0, defaults appropriate for the system are used where possible.

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

void (*db_errcall)(char *db_errpfx, char *buffer);
: When an error occurs in the DB package, an *errno* value is returned by the method. In some cases, however, the *errno* value may be insufficient to completely describe the cause of the error.

  If *db_errcall* is non-NULL, it may be called with additional error information. The *db_errpfx* argument is the current environment's *db_errpfx* field. The *buffer* argument contains a nul-terminated string with the additional information.

  This error logging facility should not be required for normal operation, but may be useful in debugging applications.

FILE *db_errfile;
: The *db_errfile* field behaves similarly to the *db_errcall* field, except that the error message is written to the file stream represented by *db_errfile*.

  If *db_errpfx* is non-NULL, the message will be preceded by the string referenced by *db_errpfx*, a colon (":") and a space. The message will be followed by a newline character.

const char *db_errpfx;
: A prefix to prepend to error messages. Because Db does not copy the memory referenced by the *db_errpfx* field, the application may modify the error message prefix at any time.

ostream* error_stream;
: The error_stream functions like the error file, allowing errors to be redirected to a C++ error stream. It is unwise to use both error_stream with nonzero values of either errcall or errfile.

int db_verbose;
Include informational and debugging messages as well as error messages in the *db_errcall* and *db_errfile* output.

int (*db_yield)(void);
The Db library makes a system call to pause for some number of microseconds whenever it is necessary to wait on a lock. This may not be optimal, especially in a thread-only environment where it will be more efficient to explicitly yield the processor to another thread. If *db_yield* returns 0, indicating success, the lock will be re-tried, otherwise, the default descheduling method will be called and then the lock will be re-tried. Note, it is probably incorrect to supply a thread *db_yield* method if more than a single process is operating in the Db environment. This is because most thread-yield functions will not allow other processes to run, and the lock may be held by another process, not by another thread.

Solaris architecture note: Because of bugs in versions of Solaris before version 5.6, Db uses the *sema_wait*(3T) call instead of the *sema_trywait*(3T) call. For this reason, setting the *db_yield* field of the DbEnv object will have no effect on Solaris.

Each of the open functions that *appinit* may call (*DbLockTab::open*, *DbLog::open*, *DbMpool::open* and *DbTxnMgr::open*) is called as follows, where the DB_CREATE flag is optional:

```
XXX::open(NULL, DB_CREATE,
        S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP, dbenv)
```

This call will cause each subsystem to construct pathnames as described in the section on ‘‘FILE NAMING’’. The subsystem has permission to read and write underlying files as necessary, and optionally to create files. (All created files will be created readable and writeable by the owner and the group. The group ownership of created files is based on the system and directory defaults, and is not further specified by Db.)

In addition, the *dbenv* argument is passed to the open functions of any subsystems initialized by *appinit*. For this reason the fields of the DbEnv object relevant to the subsystems being initialized must themselves be initialized before *appinit* is called. See the manual page for each subsystem for a list of these fields and their uses.

The return value from each of these calls is placed in the appropriate field of the DbEnv object:

DbLockTab *lk_info;
The return value of the *DbLockTab::open*(3) call.

DbLog *lg_info;
The return value of the *DbLog::open*(3) call.

DbMpool *mp_info;
The return value of the *DbMpool::open*(3) call.

DbTxnMgr *tx_info;
The return value of the *DbTxnMgr::open*(3) call.

In general, these fields are not directly used by applications; subsystems of Db that use these fields will simply reference them using the DbEnv argument passed to the subsystem.

For example, an application using the Db hash access method functions to access a database will first call *Db::open* passing it the DbEnv argument filled in by the initial call to *appinit*. Then, all future calls to the hash access method functions for that database will automatically use the underlying shared memory buffer pool that was specified by the *mp_info* field of that DbEnv argument.

The single exception to this rule is the *tx_info* field, which applications must explicitly specify to the *DbTxnMgr::begin*, *DbTxnMgr::checkpoint* and *DbTxnMgr::close* functions.

The *error_model* field of DbEnv allows the user to configure the way errors are treated in DB. It can be changed at any time (e.g., after the call to *DbEnv::appinit*). The error model is described in *DbException*(3).

FILE NAMING

The most important task of *appinit* is to structure file naming within Db.

Each of the locking, logging, memory pool and transaction subsystems of Db require shared memory regions, backed by the filesystem. Further, cooperating applications (or multiple invocations of the same application) must agree on the location of the shared memory regions and other files used by the Db subsystems, the log files used by the logging subsystem, and, of course, the data files.

Although it is possible to specify full pathnames to all Db functions, this is cumbersome and requires that applications be recompiled when database files are moved. The *appinit* method makes it possible to place database files in a single directory, or in multiple directories, grouped by their method within the database.

Applications are normally expected to specify a single directory home for their database. This can be done easily in the call to *appinit* by specifying a value for the *db_home* argument. There are more complex configurations where it may be desirable to override *db_home* or provide supplementary path information.

The following describes the possible ways in which file naming information may be specified to the Db library. The specific circumstances and order in which these ways are applied are described in a subsequent paragraph.

db_home

If the *db_home* argument to *appinit* is non-NULL, its value may be used as the database home, and files named relative to its path.

DB_HOME

If the DB_HOME environment variable is set when *appinit* is called, its value may be used as the database home, and files named relative to its path.

db_config

The *db_config* argument to *appinit* may be used to specify an array of character strings of the format ‘‘NAME VALUE’’, that specify file name information for the process’ Db environment. The whitespace delimiting the two parts of the entry may be one or more <space> or <tab> characters. (Leading or trailing <space> and <tab> characters are discarded.) Each entry must specify both the NAME and the VALUE of the pair. All entries with unrecognized NAME values will be ignored. The *db_config* array must be NULL terminated.

DB_CONFIG

The same information specified to the *db_config* argument to *appinit* may be specified using a configuration file. If a database home directory has been specified (either by the application specifying a non-NULL *db_home* argument to *appinit*, or by the application setting the DB_USE_ENVIRON or DB_USE_ENVIRON_ROOT flags and the DB_HOME environment variable being set), any file named ‘‘DB_CONFIG’’ in the database home directory will be read for lines of the format ‘‘NAME VALUE’’. The whitespace delimiting the two parts of the line may be one or more <space> or <tab> characters. (Leading or trailing <space> and <tab> characters are discarded.) All empty lines or lines whose first non-whitespace character is a hash character (‘‘#’’) will be ignored. Each line must specify both the NAME and the VALUE of the pair. All lines with unrecognized NAME values will be ignored.

The following ‘‘NAME VALUE’’ pairs in the *db_config* argument and the DB_CONFIG file are currently supported by Db.

DB_DATA_DIR

The path of a directory to be used as the location of the access method data files, e.g., paths specified to the *Db::open*(3) method will be relative to this path.

The DB_DATA_DIR paths are additive, and specifying more than one will result in each specified directory being searched for database data files. If multiple paths are specified, created data files will always be created in the **first** directory specified.

DB_LOG_DIR

The path of a directory to be used as the location of logging files, e.g., files created by the *DbLog*(3) subsystem will be relative to this directory. If specified, this is the directory name that will be passed to *DbLog::open*(3).

DB_TMP_DIR
: The path of a directory to be used as the location of temporary files, e.g., files created to back in-memory access method databases will be created relative to this path. Note, these temporary files can potentially be quite large, depending on the size of the database.

  If DB_TMP_DIR is not specified, the following environment variables are checked in order: ‘‘TMPDIR’’, ‘‘TEMP’’, ‘‘TMP’’ and ‘‘TempFolder’’. If one of them is set, temporary files are created relative to the directory it specifies.

  If DB_TMP_DIR is not specified and none of the above environment variables are set, the first possible one of the following directories is used: */var/tmp*, */usr/tmp*, */temp*, */tmp*, *C:/temp* and *C:/tmp*.

The following describes the specific circumstances and order in which the different ways of specifying file naming information are applied. Specifically, Db file name processing proceeds sequentially through the following steps:

‘‘/’’
: If any file name specified to any Db method begins with a leading slash, that file name is used without modification by Db.

DB_CONFIG
: If a relevant configuration string (e.g., DB_DATA_DIR), is specified in the DB_CONFIG configuration file, the VALUE from the ‘‘NAME VALUE’’ pair is prepended to the current file name. If the resulting file name begins with a leading slash, the file name is used without further modification by Db.

  The DB_CONFIG configuration file is intended to permit systems to customize file location for a database independent of applications using that database. For example, a database administrator can move the database log and data files to a different location without application recompilation.

db_config
: If a relevant configuration string (e.g., DB_DATA_DIR), is specified in the *db_config* argument and is not specified in the DB_CONFIG file, the VALUE from the ‘‘NAME VALUE’’ pair is prepended to the current file name. If the resulting file name begins with a leading slash, the file name is used without further modification by Db.

  The *db_config* argument is intended to permit applications to customize file location for a database. For example, an application writer can place data files and log files in different directories, or instantiate a new log directory each time the application runs.

DB_HOME
: If the DB_HOME environment variable was set, (and the application has set the appropriate DB_USE_ENVIRON or DB_USE_ENVIRON_ROOT environment variable), its value is prepended to the current file name. If the resulting file name begins with a leading slash, the file name is used without further modification by Db.

  The DB_HOME environment variable is intended to permit users and system administrators to override application and installation defaults, e.g.,

```
env DB_HOME=/database/my_home application
```

  Alternatively, application writers are encouraged to support the **−h** option found in the supporting Db utilities to let users specify a database home.

db_home
: If the application specified a non-NULL *db_home* argument to *appinit* (and the database home was not already specified using the DB_HOME environment variable) its value is prepended to the current file name. If the resulting file name begins with a leading slash, the file name is used without

further modification by Db.

(nothing)
Finally, all file names are interpreted relative to the current working directory of the process.

The common model for a Db environment is one where only the DB_HOME environment variable, or the *db_home* argument, is specified. In this case, all data files will be presumed to be relative to that directory, and all files created by the Db subsystems will be created in that directory.

The more complex model for a transaction environment might be one where a database home is specified, using either the DB_HOME environment variable or the *db_home* argument to *appinit*, and then DB_DATA_DIR and DB_LOG_DIR are set to the relative path names of directories underneath the home directory using the *db_config* argument to *appinit* or the DB_CONFIG file.

EXAMPLES

Store all files in the directory */a/database*:

```
DbEnv::appinit("/a/database", NULL, ...);
```

Create temporary backing files in */b/temporary*, and all other files in */a/database*:

```
char *config[] = {
   "DB_TMP_DIR /b/temporary",
   NULL
};

DbEnv::appinit("/a/database", config, ...);
```

Store data files in */a/database/datadir*, log files in */a/database/logdir*, and all other files in the directory */a/database*:

```
char *config[] = {
   "DB_DATA_DIR datadir",
   "DB_LOG_DIR logdir",
   NULL
};

DbEnv::appinit("/a/database", config, ...);
```

Store data files in */a/database/data1* and */b/data2*, and all other files in the directory */a/database*. Any data files that are created will be created in */b/data2*:

```
char *config[] = {
   "DB_DATA_DIR /b/data2",
   "DB_DATA_DIR data1",
   NULL
};

DbEnv::appinit("/a/database", config, ...);
```

See the file *examples_cxx/AppinitExample.cpp* in the Db source distribution for a C++ language code example of how an application might use *appinit* to configure its Db environment.

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 *appinit* method may fail and throw a *DbException*(3)

or return *errno* for any of the errors specified for the following DB and library functions: Db::close(3), DbEnv::appexit(3), DbLock::unlink(3), DbLockTab::open(3), DbLog::compare(3), DbLog::get(3), DbLog::open(3), DbLog::unlink(3), DbMpool::open(3), DbMpool::unlink(3), DbTxnMgr::checkpoint(3), DbTxnMgr::open(3), DbTxnMgr::unlink(3), calloc(3), fclose(3), fcntl(2), fflush(3), fgets(3), fopen(3), malloc(3), memcpy(3), memset(3), realloc(3), stat(2), strchr(3), strcmp(3), strcpy(3), strdup(3), strerror(3), strlen(3), strsep(3), and time(3).

In addition, the *appinit* 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_THREAD flag was specified and spinlocks are not implemented for this architecture.

  The DB_RECOVER or DB_RECOVER_FATAL flag was specified, and no log files were found.

  The DB_HOME or TMPDIR environment variables were set but empty.

  An incorrectly formatted ``NAME VALUE'' entry or line was found.

[ENOSPC]
: HP-UX only: a previously created Db environment for this process still exists.

The *DbEnv::appexit* method may fail and throw a *DbException*(3)

or return *errno* for any of the errors specified for the following DB and library functions: DbLockTab::close(3), DbLog::close(3), DbMpool::close(3), and DbTxnMgr::close(3).

BUGS

Due to the constraints of the PA-RISC memory architecture, HP-UX does not allow a process to map a file into its address space multiple times. For this reason, each DB environment may be opened only once by a process on HP-UX, i.e., calls to *appinit* will fail if the specified Db environment has been opened and not subsequently closed.

Because of bugs in versions of Solaris before version 5.6, setting the *db_yield* field of the DbEnv object will have no effect on Solaris.

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)