**NAME**

hsearch − hsearch, hcreate, hdestroy

**SYNOPSIS**

```
#define DB_DBM_HSEARCH    1
#include <db.h>

typedef enum {
      FIND, ENTER
} ACTION;

typedef struct entry {
      char *key;
      void *data;
} ENTRY;

ENTRY *
hsearch(ENTRY item, ACTION action);

int
hcreate(unsigned int nelem);

void
hdestroy(void);
```

**DESCRIPTION**

The *hsearch* interface to the DB library is intended to provide source code compatibility for historic applications. It is not recommended for any other purpose.

To compile *hsearch* applications, replace the application's *#include* of the hsearch include file (e.g., "#include <search.h>") with the following two lines:

```
#define DB_DBM_HSEARCH    1
#include <db.h>
```

and recompile.

The *hcreate* function creates an in-memory database. The *nelem* argument is an estimation of the maximum number of key/data pairs that will be stored in the database.

The *hdestroy* function discards the database.

Database elements are structures of type *ENTRY*, which contain at least two fields: *key* and *data*. The field *key* is declared to be of type "char *" and is the key used for storage and retrieval. The field *data* is declared to be of type "void *" and is its associated data.

The *hsearch* function retrieves key/data pairs from, and stores key/data pairs into, the database.

The *action* argument must be set to one of two values:

ENTER
: If the key does not already appear in the database, insert the key/data pair into the database. If the key already appears in the database, return a reference to an *ENTRY* structure referencing the existing key and its associated data element.

FIND
: Retrieve the specified key/data pair from the database.

**COMPATIBILITY NOTES**

Historically, *hsearch* required applications to maintain the keys and data in the application's memory for as long as the *hsearch* database existed. As DB handles key and data management internally, there is no requirement that applications maintain local copies of key and data items, although the only effect of doing so should be the allocation of additional memory.

**DIAGNOSTICS**

The *hcreate* function returns 0 on failure, setting *errno*, and non-zero on success.

The *hsearch* function returns a pointer to an ENTRY structure on success, and NULL, setting *errno*, if the *action* specified was FIND and the item did not appear in the database.

**ERRORS**

The *hcreate* function may fail and return *errno* for any of the errors specified for the following DB and library functions: db_open(3), and memset(3).

The *hsearch* function may fail and return *errno* for any of the errors specified for the following DB and library functions: DB->get(3), DB->put(3), memset(3), and strlen(3).

In addition, the *hsearch* function will fail, setting *errno* to 0, if the *action* specified was FIND and the item did not appear in the database.

**SEE ALSO**

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).

*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)