

NAME

tcbdb.h – the B+ tree database API of Tokyo Cabinet

DESCRIPTION

B+ tree database is a file containing a B+ tree and is handled with the B+ tree database API.

To use the B+ tree database API, include `'tcutil.h'`, `'tchdb.h'`, `'tcbdb.h'` and related standard header files. Usually, write the following description near the front of a source file.

```
#include <tcutil.h>
#include <tchdb.h>
#include <tcbdb.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>
#include <stdint.h>
```

Objects whose type is pointer to `'TCBDB'` are used to handle B+ tree databases. A B+ tree database object is created with the function `'tcbdbnew'` and is deleted with the function `'tcbdbdel'`. To avoid memory leak, it is important to delete every object when it is no longer in use.

Before operations to store or retrieve records, it is necessary to open a database file and connect the B+ tree database object to it. The function `'tcbdbopen'` is used to open a database file and the function `'tcbdbclose'` is used to close the database file. To avoid data missing or corruption, it is important to close every database file when it is no longer in use.

API

The function `'tcdberrmsg'` is used in order to get the message string corresponding to an error code.

```
const char *tcdberrmsg(int ecode);
'ecode' specifies the error code.
```

The function `'tcbdbnew'` is used in order to create a B+ tree database object.

```
TCBDB *tcbdbnew(void);
The return value is the new B+ tree database object.
```

The function `'tcbdbdel'` is used in order to delete a B+ tree database object.

```
void tcbdbdel(TCBDB *bdb);
'bdb' specifies the B+ tree database object.
If the database is not closed, it is closed implicitly. Note that the deleted object and its derivatives can not be used anymore.
```

The function `'tcdbdecode'` is used in order to get the last happened error code of a B+ tree database object.

```
int tcdbdecode(TCBDB *bdb);
'bdb' specifies the B+ tree database object.
The return value is the last happened error code.
The following error code is defined: 'TCESUCCESS' for success, 'TCETHREAD' for threading error, 'TCEINVALID' for invalid operation, 'TCENOFIL' for file not found, 'TCENOPERM' for no permission, 'TCEMETA' for invalid meta data, 'TCERHEAD' for invalid record header, 'TCEOPEN' for open error, 'TCECLOSE' for close error, 'TCETRUNC' for trunc error, 'TCESYNC' for sync error, 'TCESTAT' for stat error, 'TCESEEK' for seek error, 'TCERREAD' for read error, 'TCEWRITE' for write error, 'TCEMMAP' for mmap error, 'TCELOCK' for lock error, 'TCEUNLINK' for unlink error, 'TCERENAME' for rename error, 'TCEMKDIR' for mkdir error, 'TCERMDIR' for rmdir error, 'TCEKEEP' for existing record, 'TCENOREC' for no record found, and 'TCEMISC' for miscellaneous error.
```

The function `'tcbdbsetmutex'` is used in order to set mutual exclusion control of a B+ tree database object

for threading.

bool tcbdbsetmutex(TCBDB *bdb);

'bdb' specifies the B+ tree database object which is not opened.

If successful, the return value is true, else, it is false.

Note that the mutual exclusion control of the database should be set before the database is opened.

The function 'tcbdbsetcmpfunc' is used in order to set the custom comparison function of a B+ tree database object.

bool tcbdbsetcmpfunc(TCBDB *bdb, BDBCMP cmp, void *cmpop);

'bdb' specifies the B+ tree database object which is not opened.

'cmp' specifies the pointer to the custom comparison function.

'cmpop' specifies an arbitrary pointer to be given as a parameter of the comparison function. If it is not needed, 'NULL' can be specified.

If successful, the return value is true, else, it is false.

The default comparison function compares keys of two records by lexical order. The functions 'tcbdbcmplexical' (default), 'tcbdbcmpdecimal', 'tcbdbcmpint32', and 'tcbdbcmpint64' are built-in. Note that the comparison function should be set before the database is opened. Moreover, user-defined comparison functions should be set every time the database is being opened.

The function 'tcbdbtune' is used in order to set the tuning parameters of a B+ tree database object.

bool tcbdbtune(TCBDB *bdb, int32_t lmemb, int32_t nmemb, int64_t bnum, int8_t apow, int8_t fpow, uint8_t opts);

'bdb' specifies the B+ tree database object which is not opened.

'lmemb' specifies the number of members in each leaf page. If it is not more than 0, the default value is specified. The default value is 128.

'nmemb' specifies the number of members in each non-leaf page. If it is not more than 0, the default value is specified. The default value is 256.

'bnum' specifies the number of elements of the bucket array. If it is not more than 0, the default value is specified. The default value is 16381.

'apow' specifies the size of record alignment by power of 2. If it is negative, the default value is specified. The default value is 8 standing for $2^8=256$.

'fpow' specifies the maximum number of elements of the free block pool by power of 2. If it is negative, the default value is specified. The default value is 10 standing for $2^{10}=1024$.

'opts' specifies options by bitwise or: 'BDBTLARGE' specifies that the size of the database can be larger than 2GB by using 64-bit bucket array, 'BDBTDEFLATE' specifies that each page is compressed with Deflate encoding, 'BDBTTCBS' specifies that each page is compressed with TCBS encoding.

If successful, the return value is true, else, it is false.

Note that the tuning parameters of the database should be set before the database is opened.

The function 'tcbdbsetcache' is used in order to set the caching parameters of a B+ tree database object.

bool tcbdbsetcache(TCBDB *bdb, int32_t lcnnum, int32_t ncnum);

'bdb' specifies the B+ tree database object which is not opened.

'lcnnum' specifies the maximum number of leaf nodes to be cached. If it is not more than 0, the default value is specified.

'ncnum' specifies the maximum number of non-leaf nodes to be cached. If it is not more than 0, the default value is specified.

If successful, the return value is true, else, it is false.

Note that the tuning parameters of the database should be set before the database is opened.

The function 'tcdbbopen' is used in order to open a database file and connect a B+ tree database object.

bool tcdbbopen(TCBDB *bdb, const char *path, int omode);

'bdb' specifies the B+ tree database object which is not opened.

'path' specifies the path of the database file.

'omode' specifies the connection mode: 'BDBOWRITER' as a writer, 'BDBOREADER' as a reader. If the mode is 'BDBOWRITER', the following may be added by bitwise or: 'BDBOCREAT', which means it creates a new database if not exist, 'BDBOTRUNC', which means it creates a new database regardless if one exists. Both of 'BDBOREADER' and 'BDBOWRITER' can be added to by bitwise or: 'BDBONOLCK', which means it opens the database file without file locking, or 'BDBOLOCKNB', which means locking is performed without blocking.

If successful, the return value is true, else, it is false.

The function 'tcdbbclose' is used in order to close a B+ tree database object.

bool tcdbbclose(TCBDB *bdb);

'bdb' specifies the B+ tree database object.

If successful, the return value is true, else, it is false.

Update of a database is assured to be written when the database is closed. If a writer opens a database but does not close it appropriately, the database will be broken.

The function 'tcdbbput' is used in order to store a record into a B+ tree database object.

bool tcdbbput(TCBDB *bdb, const void *kbuf, int ksiz, const void *vbuf, int vsiz);

'bdb' specifies the B+ tree database object connected as a writer.

'kbuf' specifies the pointer to the region of the key.

'ksiz' specifies the size of the region of the key.

'vbuf' specifies the pointer to the region of the value.

'vsiz' specifies the size of the region of the value.

If successful, the return value is true, else, it is false.

If a record with the same key exists in the database, it is overwritten.

The function 'tcdbbput2' is used in order to store a string record into a B+ tree database object.

bool tcdbbput2(TCBDB *bdb, const char *kstr, const char *vstr);

'bdb' specifies the B+ tree database object connected as a writer.

'kstr' specifies the string of the key.

'vstr' specifies the string of the value.

'over' specifies whether the value of the duplicated record is overwritten or not.

If successful, the return value is true, else, it is false.

If a record with the same key exists in the database, it is overwritten.

The function 'tcdbbputkeep' is used in order to store a new record into a B+ tree database object.

bool tcdbbputkeep(TCBDB *bdb, const void *kbuf, int ksiz, const void *vbuf, int vsiz);

'bdb' specifies the B+ tree database object connected as a writer.

'kbuf' specifies the pointer to the region of the key.

'ksiz' specifies the size of the region of the key.

'vbuf' specifies the pointer to the region of the value.

'vsiz' specifies the size of the region of the value.

If successful, the return value is true, else, it is false.

If a record with the same key exists in the database, this function has no effect.

The function 'tcdbbputkeep2' is used in order to store a new string record into a B+ tree database object.

bool tcdbbputkeep2(TCBDB *bdb, const char *kstr, const char *vstr);

'bdb' specifies the B+ tree database object connected as a writer.

'kstr' specifies the string of the key.

'vstr' specifies the string of the value.

If successful, the return value is true, else, it is false.

If a record with the same key exists in the database, this function has no effect.

The function 'tcbdbputcat' is used in order to concatenate a value at the end of the existing record in a B+ tree database object.

bool tcbdbputcat(TCBDB *bdb, const void *kbuf, int ksiz, const void *vbuf, int vsiz);

'bdb' specifies the B+ tree database object connected as a writer.

'kbuf' specifies the pointer to the region of the key.

'ksiz' specifies the size of the region of the key.

'vbuf' specifies the pointer to the region of the value.

'vsiz' specifies the size of the region of the value.

If successful, the return value is true, else, it is false.

If there is no corresponding record, a new record is created.

The function 'tcbdbputcat2' is used in order to concatenate a string value at the end of the existing record in a B+ tree database object.

bool tcbdbputcat2(TCBDB *bdb, const char *kstr, const char *vstr);

'bdb' specifies the B+ tree database object connected as a writer.

'kstr' specifies the string of the key.

'vstr' specifies the string of the value.

If successful, the return value is true, else, it is false.

If there is no corresponding record, a new record is created.

The function 'tcbdbputdup' is used in order to store a new record into a B+ tree database object with allowing duplication of keys.

bool tcbdbputdup(TCBDB *bdb, const void *kbuf, int ksiz, const void *vbuf, int vsiz);

'bdb' specifies the B+ tree database object connected as a writer.

'kbuf' specifies the pointer to the region of the key.

'ksiz' specifies the size of the region of the key.

'vbuf' specifies the pointer to the region of the value.

'vsiz' specifies the size of the region of the value.

If successful, the return value is true, else, it is false.

If a record with the same key exists in the database, the new record is placed after the existing one.

The function 'tcbdbputdup2' is used in order to store a new string record into a B+ tree database object with allowing duplication of keys.

bool tcbdbputdup2(TCBDB *bdb, const char *kstr, const char *vstr);

'bdb' specifies the B+ tree database object connected as a writer.

'kstr' specifies the string of the key.

'vstr' specifies the string of the value.

If successful, the return value is true, else, it is false.

If a record with the same key exists in the database, the new record is placed after the existing one.

The function 'tcbdbputdup3' is used in order to store records into a B+ tree database object with allowing duplication of keys.

bool tcbdbputdup3(TCBDB *bdb, const void *kbuf, int ksiz, const TCLIST *vals);

'bdb' specifies the B+ tree database object connected as a writer.

'kbuf' specifies the pointer to the region of the common key.

'ksiz' specifies the size of the region of the common key.

'vals' specifies a list object containing values.

If successful, the return value is true, else, it is false.

If a record with the same key exists in the database, the new records are placed after the existing one.

The function 'tcbdbout' is used in order to remove a record of a B+ tree database object.

bool tcdbbout(TCBDB *bdb, const void *kbuf, int ksiz);

'bdb' specifies the B+ tree database object connected as a writer.

'kbuf' specifies the pointer to the region of the key.

'ksiz' specifies the size of the region of the key.

If successful, the return value is true, else, it is false.

If the key of duplicated records is specified, the value of the first record is selected.

The function *'tcdbbout2'* is used in order to remove a string record of a B+ tree database object.

bool tcdbbout2(TCBDB *bdb, const char *kstr);

'bdb' specifies the B+ tree database object connected as a writer.

'kstr' specifies the string of the key.

If successful, the return value is true, else, it is false.

If the key of duplicated records is specified, the value of the first record is selected.

The function *'tcdbbout3'* is used in order to remove records of a B+ tree database object.

bool tcdbbout3(TCBDB *bdb, const void *kbuf, int ksiz);

'bdb' specifies the B+ tree database object connected as a writer.

'kbuf' specifies the pointer to the region of the key.

'ksiz' specifies the size of the region of the key.

If successful, the return value is true, else, it is false.

If the key of duplicated records is specified, all of them are removed.

The function *'tcdbbget'* is used in order to retrieve a record in a B+ tree database object.

void *tcdbbget(TCBDB *bdb, const void *kbuf, int ksiz, int *sp);

'bdb' specifies the B+ tree database object.

'kbuf' specifies the pointer to the region of the key.

'ksiz' specifies the size of the region of the key.

'sp' specifies the pointer to the variable into which the size of the region of the return value is assigned.

If successful, the return value is the pointer to the region of the value of the corresponding record. 'NULL' is returned if no record corresponds.

If the key of duplicated records is specified, the value of the first record is selected.

Because an additional zero code is appended at the end of the region of the return value, the return value can be treated as a character string. Because the region of the return value is allocated with the *'malloc'* call, it should be released with the *'free'* call when it is no longer in use.

The function *'tcdbbget2'* is used in order to retrieve a string record in a B+ tree database object.

char *tcdbbget2(TCBDB *bdb, const char *kstr);

'bdb' specifies the B+ tree database object.

'kstr' specifies the string of the key.

If successful, the return value is the string of the value of the corresponding record.

'NULL' is returned if no record corresponds.

If the key of duplicated records is specified, the value of the first record is selected.

Because the region of the return value is allocated with the *'malloc'* call, it should be released with the *'free'* call when it is no longer in use.

The function *'tcdbbget3'* is used in order to retrieve a record in a B+ tree database object as a volatile buffer.

const void *tcdbbget3(TCBDB *bdb, const void *kbuf, int ksiz, int *sp);

'bdb' specifies the B+ tree database object.

'kbuf' specifies the pointer to the region of the key.

'ksiz' specifies the size of the region of the key.

'sp' specifies the pointer to the variable into which the size of the region of the return value is assigned.

If successful, the return value is the pointer to the region of the value of the corresponding record. 'NULL' is returned if no record corresponds.

If the key of duplicated records is specified, the value of the first record is selected. Because an additional zero code is appended at the end of the region of the return value, the return value can be treated as a character string. Because the region of the return value is volatile and it may be spoiled by another operation of the database, the data should be copied into another involatile buffer immediately.

The function 'tcbdbget4' is used in order to retrieve records in a B+ tree database object.

TCLIST *tcbdbget4(TCBDB *bdb, const void *kbuf, int ksiz);

'bdb' specifies the B+ tree database object.

'kbuf' specifies the pointer to the region of the key.

'ksiz' specifies the size of the region of the key.

If successful, the return value is a list object of the values of the corresponding records. 'NULL' is returned if no record corresponds.

Because the object of the return value is created with the function 'tclistnew', it should be deleted with the function 'tclistdel' when it is no longer in use.

The function 'tcbdbvnum' is used in order to get the number of records corresponding a key in a B+ tree database object.

int tcbdbvnum(TCBDB *bdb, const void *kbuf, int ksiz);

'bdb' specifies the B+ tree database object.

'kbuf' specifies the pointer to the region of the key.

'ksiz' specifies the size of the region of the key.

If successful, the return value is the number of the corresponding records, else, it is 0.

The function 'tcbdbvsiz' is used in order to get the size of the value of a record in a B+ tree database object.

int tcbdbvsiz(TCBDB *bdb, const void *kbuf, int ksiz);

'bdb' specifies the B+ tree database object.

'kbuf' specifies the pointer to the region of the key.

'ksiz' specifies the size of the region of the key.

If successful, the return value is the size of the value of the corresponding record, else, it is -1.

If the key of duplicated records is specified, the value of the first record is selected.

The function 'tcbdbsync' is used in order to synchronize updated contents of a B+ tree database object with the file and the device.

bool tcbdbsync(TCBDB *bdb);

'bdb' specifies the B+ tree database object connected as a writer.

If successful, the return value is true, else, it is false.

This function is useful when another process connects the same database file.

The function 'tcbdboptimize' is used in order to optimize the file of a B+ tree database object.

bool tcbdboptimize(TCBDB *bdb, int32_t lmemb, int32_t nmemb, int64_t bnum, int8_t apow, int8_t fpow, uint8_t opts);

'bdb' specifies the B+ tree database object connected as a writer.

'lmemb' specifies the number of members in each leaf page. If it is not more than 0, the current setting is not changed.

'nmemb' specifies the number of members in each non-leaf page. If it is not more than 0, the current setting is not changed.

'bnum' specifies the number of elements of the bucket array. If it is not more than 0, the current setting is not changed.

'apow' specifies the size of record alignment by power of 2. If it is negative, the current setting is not changed.

'fpow' specifies the maximum number of elements of the free block pool by power of 2.

If it is negative, the current setting is not changed.

'*opts*' specifies options by bitwise or: 'BDBTLARGE' specifies that the size of the database can be larger than 2GB by using 64-bit bucket array, 'BDBTDEFLATE' specifies that each record is compressed with Deflate encoding, 'BDBTTCBS' specifies that each page is compressed with TCBS encoding. If it is 'UINT8_MAX', the default setting is not changed.

If successful, the return value is true, else, it is false.

This function is useful to reduce the size of the database file with data fragmentation by successive updating.

The function 'tcbdbtranbegin' is used in order to begin the transaction of a B+ tree database object.

bool tcbdbtranbegin(TCBDB *bdb);

'*bdb*' specifies the B+ tree database object connected as a writer.

If successful, the return value is true, else, it is false.

The database is locked by the thread while the transaction so that only one transaction can be activated with a database object at the same time. Thus, the serializable isolation level is assumed if every database operation is performed in the transaction.

The function 'tcbdbtrancommit' is used in order to commit the transaction of a B+ tree database object.

bool tcbdbtrancommit(TCBDB *bdb);

'*bdb*' specifies the B+ tree database object connected as a writer.

If successful, the return value is true, else, it is false.

Update in the transaction is fixed when it is committed successfully.

The function 'tcbdbtranabort' is used in order to abort the transaction of a B+ tree database object.

bool tcbdbtranabort(TCBDB *bdb);

'*bdb*' specifies the B+ tree database object connected as a writer.

If successful, the return value is true, else, it is false.

Update in the transaction is discarded when it is aborted. The state of the database is rolled-back to before transaction.

The function 'tcbdbpath' is used in order to get the file path of a B+ tree database object.

const char *tcbdbpath(TCBDB *bdb);

'*bdb*' specifies the B+ tree database object.

The return value is the path of the database file or 'NULL' if the object does not connect to any database file.

The function 'tcdbbrnum' is used in order to get the number of records of a B+ tree database object.

uint64_t tcdbbrnum(TCBDB *bdb);

'*bdb*' specifies the B+ tree database object.

The return value is the number of records or 0 if the object does not connect to any database file.

The function 'tcdbfsiz' is used in order to get the size of the database file of a B+ tree database object.

uint64_t tcdbfsiz(TCBDB *bdb);

'*bdb*' specifies the B+ tree database object.

The return value is the size of the database file or 0 if the object does not connect to any database file.

The function 'tcdbbcurnew' is used in order to create a cursor object.

BDBCUR *tcdbbcurnew(TCBDB *bdb);

'*bdb*' specifies the B+ tree database object.

The return value is the new cursor object.

Note that the cursor is available only after initialization with the 'tcdbbcurfirst' or the 'tcdbbcurjump' functions and so on. Moreover, the position of the cursor will be indefinite when the database is updated after the initialization of the cursor.

The function ‘tcdbcurdel’ is used in order to delete a cursor object.

```
void tcdbcurdel(BDBCUR *cur);  
    ‘cur’ specifies the cursor object.
```

The function ‘tcdbcurfirst’ is used in order to move a cursor object to the first record.

```
bool tcdbcurfirst(BDBCUR *cur);  
    ‘cur’ specifies the cursor object.  
    If successful, the return value is true, else, it is false. False is returned if there is no  
    record in the database.
```

The function ‘tcdbcurlast’ is used in order to move a cursor object to the last record.

```
bool tcdbcurlast(BDBCUR *cur);  
    ‘cur’ specifies the cursor object.  
    If successful, the return value is true, else, it is false. False is returned if there is no  
    record in the database.
```

The function ‘tcdbcurjump’ is used in order to move a cursor object to the front of records corresponding a key.

```
bool tcdbcurjump(BDBCUR *cur, const void *kbuf, int ksiz);  
    ‘cur’ specifies the cursor object.  
    ‘kbuf’ specifies the pointer to the region of the key.  
    ‘ksiz’ specifies the size of the region of the key.  
    If successful, the return value is true, else, it is false. False is returned if there is no  
    record corresponding the condition.  
    The cursor is set to the first record corresponding the key or the next substitute if com-  
    pletely matching record does not exist.
```

The function ‘tcdbcurjump2’ is used in order to move a cursor object to the front of records corresponding a key string.

```
bool tcdbcurjump2(BDBCUR *cur, const char *kstr);  
    ‘cur’ specifies the cursor object.  
    ‘kstr’ specifies the string of the key.  
    If successful, the return value is true, else, it is false. False is returned if there is no  
    record corresponding the condition.  
    The cursor is set to the first record corresponding the key or the next substitute if com-  
    pletely matching record does not exist.
```

The function ‘tcdbcurprev’ is used in order to move a cursor object to the previous record.

```
bool tcdbcurprev(BDBCUR *cur);  
    ‘cur’ specifies the cursor object.  
    If successful, the return value is true, else, it is false. False is returned if there is no previ-  
    ous record.
```

The function ‘tcdbcurnext’ is used in order to move a cursor object to the next record.

```
bool tcdbcurnext(BDBCUR *cur);  
    ‘cur’ specifies the cursor object.  
    If successful, the return value is true, else, it is false. False is returned if there is no next  
    record.
```

The function ‘tcdbcurput’ is used in order to insert a record around a cursor object.

```
bool tcdbcurput(BDBCUR *cur, const void *vbuf, int vsiz, int cpmode);  
    ‘cur’ specifies the cursor object of writer connection.  
    ‘vbuf’ specifies the pointer to the region of the value.  
    ‘vsiz’ specifies the size of the region of the value.  
    ‘cpmode’ specifies detail adjustment: ‘BDBCPCURRENT’, which means that the value
```

of the current record is overwritten, 'BDBCPBEFORE', which means that the new record is inserted before the current record, 'BDBCPAFTER', which means that the new record is inserted after the current record.

If successful, the return value is true, else, it is false. False is returned when the cursor is at invalid position.

After insertion, the cursor is moved to the inserted record.

The function 'tcdbcurput2' is used in order to insert a string record around a cursor object.

bool tcdbcurput2(BDBCUR *cur, const char *vstr, int cpmode);

'cur' specifies the cursor object of writer connection.

'vstr' specifies the string of the value.

'cpmode' specifies detail adjustment: 'BDBPCURRENT', which means that the value of the current record is overwritten, 'BDBCPBEFORE', which means that the new record is inserted before the current record, 'BDBCPAFTER', which means that the new record is inserted after the current record.

If successful, the return value is true, else, it is false. False is returned when the cursor is at invalid position.

After insertion, the cursor is moved to the inserted record.

The function 'tcdbcurout' is used in order to delete the record where a cursor object is.

bool tcdbcurout(BDBCUR *cur);

'cur' specifies the cursor object of writer connection.

If successful, the return value is true, else, it is false. False is returned when the cursor is at invalid position.

After deletion, the cursor is moved to the next record if possible.

The function 'tcdbcurkey' is used in order to get the key of the record where the cursor object is.

char *tcdbcurkey(BDBCUR *cur, int *sp);

'cur' specifies the cursor object.

'sp' specifies the pointer to the variable into which the size of the region of the return value is assigned.

If successful, the return value is the pointer to the region of the key, else, it is 'NULL'. 'NULL' is returned when the cursor is at invalid position.

Because an additional zero code is appended at the end of the region of the return value, the return value can be treated as a character string. Because the region of the return value is allocated with the 'malloc' call, it should be released with the 'free' call when it is no longer in use.

The function 'tcdbcurkey2' is used in order to get the key string of the record where the cursor object is.

char *tcdbcurkey2(BDBCUR *cur);

'cur' specifies the cursor object.

If successful, the return value is the string of the key, else, it is 'NULL'. 'NULL' is returned when the cursor is at invalid position.

Because the region of the return value is allocated with the 'malloc' call, it should be released with the 'free' call when it is no longer in use.

The function 'tcdbcurkey3' is used in order to get the key of the record where the cursor object is, as a volatile buffer.

const char *tcdbcurkey3(BDBCUR *cur, int *sp);

'cur' specifies the cursor object.

'sp' specifies the pointer to the variable into which the size of the region of the return value is assigned.

If successful, the return value is the pointer to the region of the key, else, it is 'NULL'. 'NULL' is returned when the cursor is at invalid position.

Because an additional zero code is appended at the end of the region of the return value,

the return value can be treated as a character string. Because the region of the return value is volatile and it may be spoiled by another operation of the database, the data should be copied into another involatile buffer immediately.

The function 'tcdbbcurl' is used in order to get the value of the record where the cursor object is.

```
char *tcdbbcurl(BDBCUR *cur, int *sp);
```

'cur' specifies the cursor object.

'sp' specifies the pointer to the variable into which the size of the region of the return value is assigned.

If successful, the return value is the pointer to the region of the value, else, it is 'NULL'. 'NULL' is returned when the cursor is at invalid position.

Because an additional zero code is appended at the end of the region of the return value, the return value can be treated as a character string. Because the region of the return value is allocated with the 'malloc' call, it should be released with the 'free' call when it is no longer in use.

The function 'tcdbbcurl2' is used in order to get the value string of the record where the cursor object is.

```
char *tcdbbcurl2(BDBCUR *cur);
```

'cur' specifies the cursor object.

If successful, the return value is the string of the value, else, it is 'NULL'. 'NULL' is returned when the cursor is at invalid position.

Because the region of the return value is allocated with the 'malloc' call, it should be released with the 'free' call when it is no longer in use.

The function 'tcdbbcurl3' is used in order to get the value of the record where the cursor object is, as a volatile buffer.

```
const char *tcdbbcurl3(BDBCUR *cur, int *sp);
```

'cur' specifies the cursor object.

'sp' specifies the pointer to the variable into which the size of the region of the return value is assigned.

If successful, the return value is the pointer to the region of the value, else, it is 'NULL'. 'NULL' is returned when the cursor is at invalid position.

Because an additional zero code is appended at the end of the region of the return value, the return value can be treated as a character string. Because the region of the return value is volatile and it may be spoiled by another operation of the database, the data should be copied into another involatile buffer immediately.

The function 'tcdbbcurrrec' is used in order to get the key and the value of the record where the cursor object is.

```
bool tcdbbcurrrec(BDBCUR *cur, TCXSTR *kxstr, TCXSTR *vxstr);
```

'cur' specifies the cursor object.

'kxstr' specifies the object into which the key is wrote down.

'vxstr' specifies the object into which the value is wrote down.

If successful, the return value is true, else, it is false. False is returned when the cursor is at invalid position.

SEE ALSO

tcctest(1), tcbmttest(1), tcbmgr(1), tokyocabinet(3)