

RETURN TO MAIN MENU

# **Recital Library**

## **Recital Library**

Recital Corporation,  
100 Cummings Center, Suite 318J  
Beverly, MA 01915

Recital may have patents and/or patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.  
COPYRIGHT ©1988-2004 Recital Corporation. All rights reserved. All Recital products are trademarks or registered trademarks of Recital Corporation, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

Last Updated September, 2004

# INDEX

<b>OVERVIEW</b>	<b>1</b>
<b>SAMPLE FILES</b>	<b>5</b>
<b>INCLUDE FILES</b>	<b>6</b>
<b>RETURN CODE VALUES</b>	<b>7</b>
<b>FUNCTION TABLE</b>	<b>9</b>
<b>FUNCTIONS</b>	
dbakey()	12
dbappend()	14
dbatofld()	16
dbatokeny()	18
dbckey()	21
dbclose()	23
dbcreat()	25
dbcreatx()	28
dbdcache()	30
dbdelete()	32
dbdo()	34
dbfield()	36
dbfilemode()	38
dbfldtoa()	40
dbflush()	42
dbfwd()	44
dbgather()	46
dbgcache()	48
dbgetf()	50
dbgetfx()	52
dbgetm()	54
dbgetnr()	56
dbgetpr()	58
dbgetr()	60
dbgetrk()	62
dbgetseqno()	64
dbicache()	67
dbiclose()	69
dbicreat()	71
dbiflsh()	73
dbiopen()	75
dbkexpr()	77
dbkeytoa()	79
dblockf()	82
dblocki()	84

dblockr()	86
dbmclose()	88
dbmcreat()	90
dbmemoformat()	92
dbmopen()	94
dbnkey()	96
dbonerror()	98
dbopen()	99
dbpkey()	101
dbputm()	103
dbputr()	105
dbputrk()	107
dbrecall()	109
dbrecin()	111
dbrecout()	113
dbrewind()	115
dbrmvkey()	117
dbrmvr()	119
dbrun()	121
dbscatter()	123
dbset()	125
dbsetseqno()	127
dbsize()	130
dbstrcpy()	132
dbstring()	134
dbtkey()	136
dbtlockf()	138
dbtlockr()	140
dbunlockf()	142
dbunlocki()	144
dbunlockr()	146
dbupdm()	148
dbupdr()	150
dbxakey()	152
dbxckey()	155
dbxclose()	158
dbxcreate()	161
dbxdroptag()	161
dbxflsh()	167
dbxfwd()	170
dbxgetnr()	173
dbxgetpr()	176
dbxgetrk()	179
dbxgoto()	182
dbxkexpr()	185

dbxlocki()	189
dbxnkey()	192
dbxnotags()	195
dbxopen()	199
dbxpkey()	203
dbxrewind()	206
dbxrmvkey()	209
dbxtag()	213
dbxtkey()	217
dbxunlocki()	220
dbxupd()	223
<b>DBLSAMP</b>	<b>227</b>
<b>ODBC COMPATIBLE FUNCTIONS</b>	<b>247</b>

# OVERVIEW

---

## OVERVIEW

The RECITAL/Library is a library of functions that provide shared read write access to RECITAL database table (.dbf) files, index (.ndx & .dbx) files and memo (.dbt) files. Applications written in 'C' can use the RECITAL/Library to manipulate data stored in the RECITAL files. On OpenVMS any 3GL that conforms to the OpenVMS procedure calling standard can also use the RECITAL/Library. When using the RECITAL/Library with FORTRAN and COBOL on OpenVMS the dbstring() function should be used to format 'C' strings for use as parameters to these functions.

For performance reasons, all information retrieved and placed into database tables is arranged in character arrays. Functions are provided for converting portions of these character arrays into non-character data. When writing these character arrays to database tables, any portion of the array that is representative of a non-character field is automatically converted to the proper data type.

### Using the RECITAL/Library Functions

The RECITAL/Library works with the following file types:-

3. Database table files (.DBF) that contain fixed length records.
3. Index files (.NDX) and tagged index files (.DBX) that contain keys to the .DBF file enabling fast access to the data records.
3. Memo files (.DBT) that store free formatted text for 'memo' fields in the database table.

This section explains how the RECITAL/Library functions relate to these file types.

## Database Files

A RECITAL database file (.DBF) stores the data of a particular application. The data is divided into units of a fixed size called records. The RECITAL/Library functions operate on these units. For example, the dbappend() or dbputr() functions write a record to a .DBF file, dbgetr() reads a record from a .DBF file and dbupdr() function updates an existing record. Sub-units of records are called fields. At the head of each database table file, RECITAL stores information relating to the layout of the records held in the database table. The database table structure is comprised of:-

PROLOG
FIELD DEFINITIONS
FIELD DESCRIPTIONS
DATABASE RECORDS

Each field/column can be described by the following 'C' structure that is defined in the include file "dbl.h":-

```
typedef struct {
    char    fieldnm[11]; /* Field name          */
    char    type;        /* Type of data in the field:
                        'C' = character
                        'N' = numeric
                        'L' = logical
                        'D' = date
                        'M' = memo          */
    int     width;       /* Field width          */
    int     dec;         /* Number of decimal places
                        (for numeric fields only) */
    int     rpos;        /* Record buffer position */
} dBFIELD;
```

NOTE: rpos is the physical starting position of the column in the record buffer. Position 0 is for the deletion marker, so the first column will start at 1, the next position will be determined by the storage width of the previous column.

When creating a .DBF file, remember the following:-

- 1 The maximum length of a record is 4000 bytes;
- 1 The maximum number of fields in a record is 128;
- 1 The maximum length of a character field is 254 bytes.

The created .DBF file must be opened with the dbopen() function before it can be accessed. After the file is opened, a record can be inserted by using the dbputr() function or appended by using the dbappend().

It is important to consider that when a record is inserted all the records that will follow it in the .DBF file are shifted down to create a space for the new record. To avoid this time consuming shifting operation it is a good idea to insert a new record after the last inserted record. The record number of the record to be inserted will be the number of current records +1. The number of inserted records can be obtained by using the `dbsize()` function. Similarly, when a record is deleted, all the following records are shifted up to close the “gap” between the records. In this case, to avoid shifting, one can delete the record logically not physically. When a record is deleted logically, it is marked deleted but physically remains in the file. The record is considered “INACTIVE”. The record’s status (ACTIVE or INACTIVE) can be obtained by using the `dbgetr()` function. The `dbflush()` function writes the contents of the buffer used for .DBF files onto the disk. It does not have to be used in your application program, but it allows you to control the I/O. The .DBF file must be closed with the `dbcclose()` function before your program exits.

### **Index Files**

Index files contains index keys/record numbers pairs for every record in a database file. Keys and their associated record numbers are organized in a B+ tree structure, and the keys appear sorted in an ascending order. This organization permits fast access to a particular key/record number pair. By supplying a key to the functions `dbtkey()` / `dbxtkey()`, its associated record number can be obtained. This record number can be used to directly access the associated record in a database table.

Index files are created using the `dbicreat()` or `dbxcreate()` functions. When creating an index file, you must name the key type (character (C), date (D), or numeric (N)) and the key expression which is usually the name of one or more record fields (e.g., “SSN” or “NAME” + BIRTHDATE”). The `dbkexpr()` or `dbxkexpr()` functions can be used later to retrieve a key expression from an existing index file. An existing index file must be opened with the `dbiopen()` or `dbxopen()` functions. After the file is opened, you can add a key to it with the `dbakey()` / `dbxakey()` functions or remove a key by using the `dbrmvkey()` / `dbxrmvkey()` functions.

The RECITAL/Library uses an internal access pointer that is positioned at the top of an .NDX file when the file is opened. When the `dbtkey()` / `dbxtkey()` functions translates a key to the record number, they position the pointer to the position where that key is found. An application can obtain the current key by using the `dbckey()` / `dbxckey()` functions, next key by using the `dbnkey()` / `dbxnkey()` functions and previous key by using the `dbpkey()` / `dbxpkey()` functions. The access pointer can be repositioned at the top of an index file by using the `dbrewind()` / `dbxrewind()` functions or moved to the end by using the `dbfwd()` / `dbxfwd()` functions. The `dbgetrk()` / `dbxgetrk()` functions translate a key into record number and then retrieve the record. Given a key, these functions can be used to get a record in one step. The `dbiflsh()` / `dbxflsh()` functions flush the contents of the index buffer cache. Before an application program exits, the index file must be

closed with the `dbfclose()` / `dbxclose()` functions, otherwise the file may not be updated properly.

### **Memo Files**

Some data that needs to be kept in a database can be hard to organize in any of the character, numeric, date or logical fields of a .DBF. In this case, a .DBT file should be created. A .DBT file is used to store free-format text related to 'memo' fields in a database. The memo fields in a .DBF file occupy four bytes which indicate where a particular memo is located in a .DBT file. A memo field of a record can be retrieved by using the `dbgetr()`, `dbgetnr()`, `dbgetpr()` or `dbgetrk()` functions. Having retrieved the memo field, you can read a memo from a .DBT file by using the `dbgetm()` function. A memo can be stored in a .DBT file by using the `dbputm()` function. A memo field can be added or updated in the corresponding .DBF file by using `dbputr()`, `dbputrk()`, or `dbupdr()` functions.



## SAMPLE FILES

---

### SAMPLE FILES

The distribution of the RECITAL/Library in the lib sub-directory of the sdk directory, includes sample files to demonstrate the use of the Library functions and how to link and run applications which use the RECITAL/Library. Depending on the Operating System, the following sample files are included:

#### OpenVMS

File	Description
dblsamp.c	C source code sample showing function usage. A full listing is included at the end of this manual.
dblsamp.com	Run this file to compile and link the sample. Use this file as a reference for compiling and linking your own applications.
dblsamp.obj	C object code.
dblsamp.exe	Run this file to run the sample application. N.B. To ensure that the necessary symbols and logicals are defined, the OpenVMS Developer login.com file must have been run prior to calling the application.

#### UNIX/Linux

File	Description
dblsamp.c	C source code sample showing function usage. A full listing is included at the end of this manual.
makefile	Run this makefile to compile and link the sample. Use this file as a reference for compiling and linking your own applications. # make
dblsamp.o	C object code created by the make.
dblsamp.exe	Executable code created by the make. Run the code using the dblsamp script.
dblsamp	Run this script to run the sample application. N.B. The calls to recitalxxUD.sh and profile.db are used to ensure that the necessary environment variables are defined before running the application.

## INCLUDE FILES

---

### INCLUDE FILES

The following files should be included in applications which use the RECITAL/Library:

The RECITAL/Library C Header file, **dbl.h**

The RECITAL/Library C Function Prototype file, **dblproto.h**.

```
#include "dbl.h"
```

```
#include "dblproto.h"
```

## RETURN CODE VALUES

---

### RETURN CODE VALUES

The include file “dbl.h” contains symbolic definitions for the values that can be returned by RECITAL/Library functions.

An application program should refer to these values when checking return values from RECITAL/Library functions. A list of the symbols and their assigned numeric values follows:-

VALUE	SYMBOLS	DESCRIPTION
0	SUCCESS	Function executed successfully
-1	d_ERROR	A function returns -1 if it failed due to fundamental reasons such as wrong input values or the file to be accessed is closed or disk I/O operations failed because of a System limitation such as “disk full”.
-2	d_NODBF	File does not have a valid .DBF file structure.
-3	d_BADREC	Specified record number is invalid, i.e., smaller than 1 or larger than the number of records currently allocated.
-4	d_WTFAIL	File write operation failed.
-5	d_NONDX	File does not have a valid .NDX file structure.
-6	d_KYLONG	Key expression is too long
-7	d_NOKEY	Needed key is not found in the specified .NDX file or the .NDX file is empty.
-8	d_MAYBE	Needed key not found in the specified .NDX file, but the key that may immediately follow the requested key was located.
-9	d_ENDKEY	Access pointer reached the end of the specified .NDX file.
-10	d_NOSYNC	Key exists in the .NDX file, but its corresponding record does not exist in the specified .DBF file.
-11	d_TOPKEY	Access pointer is positioned at the top of the specified .NDX file.
-12	d_LOCKFAILED	Lock operation failed.
-13	d_UNLOCKFAILED	Unlock operation failed.
-14	d_ALREADYLOCKED	Database (or record) has already been locked by another user.
-15	d_NOLICENCE	User is not licensed to perform the requested operation.
-16	d_NOTLOCKED	Reserved for future use.
-17	d_NODBX	File does not have a valid .DBX structure.

-18	d_TAGERROR	Invalid tag name or number specified.
-19	d_SHARED	Table opened shared, operation requires exclusive use.
-20	d_MEMORY	Insufficient memory.

# FUNCTION TABLE

## FUNCTION TABLE

The following table is a quick reference to all of the functions available in the RECITAL/Library. The file column lists the type of RECITAL files that the function works with, that is .DBF, .DBT, .NDX and/or .DBX files.

FUNCTION	FILE	DESCRIPTION
dbakey()	ndx	Add key to index file
dbappend()	dbf	Append record to end of database file
dbatofld()	dbf	Convert ASCII to numeric field
dbatokay()	ndx	Convert ASCII to numeric/date key
dbckey()	ndx	Obtain current key from index file
dbcclose()	dbf	Close a database
dbcreat()	dbf	Create a new database
dbcreatx()	dbf	Create a new database with field descriptions
dbdcache()	dbf	Specify database cache size
dbdelete()	dbf	Mark a data record deleted
dbdo()		Execute a RECITAL program
dbfield()	dbf	Extract field value
dbfilemode()	dbf/ndx	Specify file operation mode
dbfldtoa()	dbf	Convert numeric field to ASCII
dbflush()	dbf	Flush I/O buffer cache for a database file
dbfwd()	ndx	Position access pointer to the end of index file
dbgather()	dbf	Returns field buffer data to the record buffer
dbgcache()	dbf	Enables the distributed cache manager on OpenVMS
dbgetf()	dbf	Obtain database information
dbgetfx()	dbf	Obtain extra database information
dbgetm()	dbt	Get memo from memo file
dbgetnr()	dbf/ndx	Retrieve next data record by key
dbgetpr()	dbf/ndx	Retrieve previous data record by key
dbgetr()	dbf	Retrieve a data record
dbgetrk()	dbf/ndx	Retrieve data record by key
dbgetseqno()	dbf	Return the next sequence number
dbicache()	ndx	Specify index cache
dbiclose()	ndx	Close index file
dbicreat()	ndx	Create an index file
dbiflsh()	ndx	Flush I/O buffer cache for an index file
dbiopen()	ndx	Open an existing index file
dbkexpr()	ndx	Obtain index file information
dbkeytoa()	ndx	Convert numeric/data key to ASCII
dblockf()	dbf	Lock database file
dblocki()	ndx	Lock index file

dblockr()	dbf	Lock a record in a database file
dbmclose()	dbt	Close memo file
dbmcreat()	dbt	Create a memo file
dbmemoformat()	dbt	Toggles memo input formatting
dbmopen()	dbt	Open an existing memo file
dbnkey()	ndx	Obtain next key
dbonerror()		Enable/disable recovery from I/O errors
dbopen()	dbf	Open an existing database
dbpkey()	ndx	Obtain previous key
dbputm()	dbt	Put memo in a memo file
dbputr()	dbf	Insert record in database file
dbputrk()	dbf/ndx	Add key to index file and insert record into database file
dbrecall()	dbf	Recall deleted data record
dbrecin()	dbf	Place contents of fields into buffers
dbrecout()	dbf	Place contents of buffers into fields
dbrewind()	ndx	Position access pointer to the top of index file
dbrmvkey()	ndx	Remove key from index file
dbrmvr()	dbf	Physically remove a data record
dbrun()		Run an operating system command
dbscatter()	dbf	Splits the record buffer into individual field buffers
dbset()		Issues a Recital SET COMMAND
dbsetseqno()	dbf	Set the sequence number in a table
dbsize()	dbf	Obtain number of records in a database
dbstrcpy()		Format string for fields and keys
dbstring()		Convert non 'C' string to a 'C' null terminated string
dbtkey()	ndx	Translate key to record number
dbtlockf()	dbf	Test to see if a database is locked by another user
dbtlockr()	dbf	Test to see if a record is locked by another user
dbunlockf()	dbf	Unlock a database file
dbunlocki()	ndx	Unlock a index file
dbunlockr()	dbf	Unlock a record in a database
dbupdm()	dbt	Update a memo in a memo file
dbupdr()	dbf	Update a data record
dbxakey()	dbx	Add keys to all tags in a tagged index file
dbxckey()	dbx	Read current key
dbxclose()	dbx	Close an open tagged index file
dbxcreate()	dbx	Create a tag in a tagged index file
dbxdroptag()	dbx	Drop the specified tag from a tagged index file
dbxflsh()	dbx	Flush I/O buffer cache for a tagged index file
dbxfwd()	dbx	Position access pointer to end of active tag
dbxgetnr()	dbx	Get next data record by key in active tag
dbxgetpr()	dbx	Get previous data record by key in active tag
dbxgetrk()	dbx	Get data record by key in active tag
dbxgoto()	dbx	Go to specified record number and reposition keys in tag
dbxkexpr()	dbx	Get the key expression for a specified tag

dbxlocki()	dbx	Lock tagged index file
dbxnkey()	dbx	Read next key in active tag
dbxnotags()	dbx	Return the number of tags in a tagged index file
dbxopen()	dbx	Open a dbx index file
dbxpkey()	dbx	Read previous key in active tag
dbxrewind()	dbx	Position access pointer to beginning of active tag
dbxrmvkey()	dbx	Remove key from all tags in tagged index file
dbxtag()	dbx	Change the active tag
dbxtkey()	dbx	Translate tag key to record number
dbxunlocki()	dbx	Unlock tagged index file
dbxupd()	dbx	Update all tags in the tagged index file for the specified record

## **dbakey()**

---

### **NAME**

**dbakey()**

**add key**

### **SYNOPSIS**

```
#include<dbl.h>
```

```
int    dbakey(ndx, key, recno)
```

```
<input parameters>
```

```
char    *ndx;          /* .NDX file descriptor      */
char    key;           /* Address of a buffer containing the key */
long    recno;         /* Record number            */
```

```
<output parameters>
```

```
none
```

### **RETURN VALUE**

The dbakey() function returns 0 for success, or <0 if an error occurs. See the section on return code values for a detailed list of return codes.

### **DESCRIPTION**

This function adds a key with its associated record number to the specified .NDX file which contains keys in a B-tree structure.



## EXAMPLE

The following example adds a character key “John Smith 5” and its associated record number 5 to the index file whose descriptor is in “char \*ndx”.

```
#include "dbl.h"

char   *ndx;          /* .NDX file descriptor    */
int     rc;           /* Return code          */

rc=dbakey(ndx,"John Smith 5".(long)5);
if (rc == SUCCESS) printf("key added \n");
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbatokey(), dbicreat(), dbiflsh(), dbrmvkey(), dbtkey(), dbupdr()

## **dbappend()**

---

### **NAME**

**dbappend()**                                      **append a record to the end of a database file**

### **SYNOPSIS**

```
#include<dbl.h>

int      dbappend (dbf, record, recno)

<input parameters>
char      *dbf;
char      *record;      /* Address of a variable where the record number
                        is stored by the function */

<output parameters>
long      *recno;      /* Address of a variable where the record number
                        is stored by the function */
```

### **RETURN VALUE**

The dbappend() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### **DESCRIPTIONS**

This function writes the contents of the record supplied in a user defined buffer to a .DBF file. The record is appended to the end of the database file and becomes the last physical record in the database. The contents of the output parameter “recno” indicates the record number of the new record.

DBAPPEND() attempts to lock the database to perform the append operation. Therefore appropriate error trapping should be considered.

## EXAMPLE

This example appends the record whose contents are in the “char record[100];” to the .DBF file whose file descriptor is in “char \*dbf;”.

```
#include "dbl.h"

long   recno;           /* Record number*/
char   *dbf;            /* .DBF file description*/
char   record[1000];    /* Record buffer*/
int     rc;              /* Return code*/

rc = dbappend(dbf, record, &recno);
if (rc == SUCCESS) printf("Record added \n");
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbflush(), dbgetr(), dbputr(), dbputrk(), dbsize(), dbupdr()

### NAME

**dbatofld()**

**ASCII to numeric field**

### SYNOPSIS

```
#include "dbl.h"
```

```
int    dbatofld(ascii, width, decimal, field)
```

```
<input parameters>
```

```
char    *ascii;        /* ASCII character string to be converted*/
```

```
int     width;         /* Width of the field*/
```

```
int     decimal;       /* Number of decimal places in the field*/
```

```
<output parameters>
```

```
char *field;           /* Address of the buffer where the converted  
                        string is returned*/
```

### RETURN VALUE

The dbatofld() function returns 0 for success. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

This function converts the string pointed to be ASCII to the numeric field format. The string must contain a valid floating point number. The number may be terminated by any character that cannot be part of a valid floating point number. This includes white space, punctuation other than periods, and characters other than e or E. The converted string is returned to the caller via the output parameter "field". Values of "width" and "decimal" must be the same as those used when creating the .DBF file. Those values can be obtained by the dbgetf() function.

## EXAMPLE

The following example converts a floating point number 3245.876 in ASCII representation to the .DBF file numeric field format of 12 digits width and 3 decimal locations. The converted number is stored in “char field[13];”.

```
#include "dbl.h"

char  field[13];    /* Record field location    */
int    rc;          /* Return code          */

rc = dbatofld("3245.876", 12, 3, field);
if (rc == SUCCESS) printf("conversion successful\n");
else {
    printf("error number %d\n",rc);
    exit(1);
}
```

## SEE ALSO

dbcreat(), dbfield(), dbfldtoa(), dbgetf(), dbkeytoa(), dbstring(), dbstrcpy()

## NAME

**dbatoken()**

**ASCII to index key value**

## SYNOPSIS

```
#include<dbl.h>

int      dbatoken(ascii, key)

<input parameters>
char      *ascii;          /* ASCII string to be converted      */
char      *key;            /* Key type: 'D' or 'd' is for a date key,
                           'N' or 'n' is for a numeric key      */

<output parameters>
char      *key;            /* Address of the buffer where converted key
                           is returned, thus "key" serves as both
                           the input and output parameter      */
```

## RETURN VALUE

The dbatoken() function returns 0 for success. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

This function converts the string pointed to by ascii to the .NDX file key format. The input ASCII string should represent either a valid floating point number or a date in the following format:-

YYYYMMDD	YYYY -	Year: '1900' through '2999'
	MM -	Month: '01' through '12'
	DD -	Day: '01' through '28'/'29'/'30'/'31'

## EXAMPLE

This example converts the ASCII string “124.52” into the .NDX file key format and adds the numeric key to the index specified by “char \*ndx;” with associated record number, 15.

```
#include "dbl.h"

char   *ndx;          /* .NDX file descriptor      */
int     rc;           /* Return code             */
char    key[8];       /* Numeric key location     */

key[0] = 'N';         /* Indicate numeric key    */

rc = dbatokey("124.52", key);
if (rc == SUCCESS) {
    if (dbakey(ndx, key, (long)15) printf("Key added \n");
    else {
        printf("Error number %n", rc);
        exit(1);
    }
} else {
    printf("Error number %d \n", rc);
    exit (1);
}
```

This example converts a date, 19680408 (April 8, 1968) in ASCII representation into .NDX file key format and adds the new key to the specified .NDX file with associated record number, 420.

```
#include "dbl.h"

char    *ndx;          /* .NDX file descriptor      */
int      rc;           /* Return code              */
char    key[8];        /* Numeric key location     */

key[0] = 'D';          /* Indicate date key       */

rc = dbatoken("19680408",key);
if (rc == SUCCESS) {
    if (dbakey(ndx,key,9long0 420) && rc == SUCCESS)
        Printf("Key added \n");
    Else {
        Printf("Error number %d \n", rc);
        Exit(1);
    }
} else {
    printf("Error number %d \n", rc);
    exit (1);
}
```

NOTE: The buffer to hold the converted key value ("key" in our examples) must be at least eight bytes long.

## SEE ALSO

dbakey(), dbkeytoa(), dbstring(), dbstrcpy()



## NAME

**dbckey()**

**read current key**

## SYNOPSIS

```
#include "dbl.h"

int    dbckey(ndx, key, recno)

<input parameter>
char   *ndx;          /* .NDX file descriptor          */

<output parameters>

char   *key;          /* Address of the buffer where current key will be
                        returned */
long   *recno;        /* Address of the buffer where record number
                        associated with the current key will be returned */
```

## RETURN VALUE

The dbckey() function returns 0 for success, or <0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

This function reads the current key and its associated record number from the specified .NDX file.

## EXAMPLE

Get the current key and its associated record number from the index file and print the retrieved values on the standard output. The key is assumed to be a character.

```
#include "dbl.h"

char   *ndx;           /* .NDX file descriptor      */
int     rc;            /* Return code              */
char    key[12];       /* Current key buffer       */
long    recno;         /* Record number            */

rc = dbckey(ndx, key, &recno);
if (rc == SUCCESS) printf("%s %d \n", key, recno);
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbfwd(), dbgetnr(), dbgetpr(), dbnkey(), dbpkey(), dbrewind()

## dbclose()

---

### NAME

**dbclose()**

**close database file**

### SYNOPSIS

```
#include "dbl.h"

int    dbclose(dbf)

<input parameter>
char   *dbf;          /* Database file descriptor */

<output parameter>
none
```

### RETURN VALUE

The dbclose() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

The dbclose() function is used to close the specified database (.DBF) file in the file descriptor.

### EXAMPLE

The following example closes a .DBF file whose file descriptor is in "char \*dbf;".

```
#include "dbl.h"
char   *dbf;          /* .DBF file descriptor */
int     rc;           /* Return code */

rc = dbclose(dbf);
if (rc == SUCCESS) printf("database closed \n");
else {
    printf("error number %d \n");
    exit (1);
}
```

## **SEE ALSO**

`dbiclose()`, `dbopen()`

## dbcreat()

---

### NAME

**dbcreat()**

**create a database file**

### SYNOPSIS

```
#include "dbl.h"

int      dbcreat(dbname, nfields, fields)

<input parameters>
char    *dbname;      /* Address of the character string containing the
                        name of the .DBF file to be created */
int      nfields;      /* Number of fields per record */
DBFIELD fields[];     /* Address of an array of fields definitions */

<output parameters>
none
```

### RETURN VALUE

The dbcreat() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

This function creates a .DBF file. The structure DBFIELD is defined in the header file <dbl.h> as follows:-

```

typedef struct {
    char    fieldnm[11];    /* Field name          */
    char    type;           /* Type of data in the field:
                             'C'    = character
                             'N'    = numeric
                             'L'    = logical
                             'D'    = date
                             'M'    = memo          */
    int     width;          /* Field width */
    int     dec;            /* Number of decimal places
                             (for numeric fields only) */
} dBFIELD;

```

An array of dBFIELD structure defines the organization of each record in the database. The array must be initialized and a pointer to it must be passed to dbcreat. All the structure's numbers must be set to appropriate values. If the name file already exists, this function will reinitialize the file.

NOTE: When using dbcreat, remember the record specifications of a .DBF file:-

Maximum File Name	=	80 bytes
Maximum Field Size	=	254
Maximum Record Size	=	4000
Maximum Numeric Size	=	25
Maximum Decimal Places	=	14
Maximum Number of Fields	=	128

## EXAMPLE

The following example creates a .DBF file with the name 'TEST.DBF' whose records consist of 6 fields defined in "dBFIELD fields[6]".

```

#include "dbf.h"
dBFIELD fields[6] = {
    "NAME",      'C',  20,  0,
    "SSN",       'N',   9,  0,
    "AMOUNT",    'N',   8,  2,
    "RESIDENCY", 'L',   1,  0,
    "BIRTHDATE", 'D',   8,  0,
    "COMMENT",   'C',  40,  0
}
int    rc;    /* Return code */
rc = dbcreat("test.dbf", 6, fields);
if (rc == SUCCESS) printf("database created \n");

```

```
else {  
    printf("error number %d \n, rc);  
    exit (1)  
}
```

## **SEE ALSO**

dbatofld(), dbcreat(), dbcreatx(), dbfldtoa(), dbgetf(), dbgetr(), dbgetfx(), dbicreat(), dbmcreat()

## dbcreatx()

---

### NAME

**dbcreatx()**                      **create a database, allowing the addition of field descriptions**

### SYNOPSIS

```
#include "dbl.h"
```

```
int     dbcreatx(dbname, nofields, fields, flddes[128][26])
```

<input parameters>

char	*dbname;	/* Database file descriptor	*/
int	nofields;	/* Number of fields	*/
dbFIELD	fields;	/* Field structure array	*/
char	flddes[128][26];	/* Array for field descriptions	*/

<output parameters>

none

### RETURN VALUE

The dbcreatx() function returns 0 upon success. For a description of other possible return error codes, see the Return Code Values section.

### DESCRIPTION

The dbcreatx() function performs the same function as the dbcreat() function with the exception that it allows the addition of field descriptions.



## EXAMPLE

The following example adds field descriptions stored in the two dimensional array 'flddes[128][26]' to the structure of the database.

```
#include<dbf.h>
char      *dbname;          /* Database name to create */
int       nofields;         /* Number of fields        */
dBFIELD   fields;           /* Field structure array    */
char      flddes[128][26 ]; /* Array for field description */

rc = dbcreatx("test.dbf",7,fields, flddes);
if (rc != success ) {
    printf("Error in creating database: %d\n", rc);
    exit(1);
} else {
    printf("Database created.\n");
}
```

## SEE ALSO

dbcreat(), dbgetfx()

## **dbdcache()**

---

### **NAME**

**dbdcache()** specify the database cache size

### **SYNOPSIS**

```
#include "dbl.h"

int    dbdcache(size)

<input parameters>

int    size;          /* Database cache size in records    */

<output parameters>
none
```

### **RETURN VALUE**

The dbdcache() function returns 0 for success. See the section on return code values for a detailed list of return codes.

### **DESCRIPTION**

The dbdcache() function specifies the amount of records to be cached in memory.

## EXAMPLE

This example sets the database cache to the total number of records stored in the database specified in the file descriptor stored in “char \*dbf”.

```
#include <dbl.h>
char    *dbf;          /* .DBF file descriptor    */
long    dbsize;        /* Size of the .DBF file  */
int      rc;           /* Return code            */

rc = dbsize(dbf, &dbsize);
if (rc == SUCCESS) dbdcache(dbsize);
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbflush(), dbgetnr(), dbgetr(), dbgetrk(), dbicache(), dbrecall()

## **dbdelete()**

---

### **NAME**

**dbdelete()**

**logically delete a record**

### **SYNOPSIS**

```
#include "dbl.h"

int    dbdelete(dbf, recno)

<input parameters>

char   *dbf;           /* .DBF file descriptor   */
long   recno;          /* Record number         */

<output parameters>
none
```

### **RETURN VALUE**

The dbdelete() function returns 0 for success, or <0 if an error occurs. See the section on return code values for a detailed list of return codes.

### **DESCRIPTION**

This function deletes the record logically, i.e., it marks the record deleted while physically the record remains in the .DBF file.

### **EXAMPLE**

This example marks 20 records in a .DBF file whose file descriptor is in "char \*dbf" deleted and reports the successful deletion on the standard output.

```

#include "dbl.h"

char    *dbf;          /* File descriptor    */
int     i;             /* Counter           */
int     rc;            /* Return code        */
for (i = 0; i < 20 ; ++i) {
    rc = dbdelete(dbf, i);
    if (rc == SUCCESS) printf("%d record deleted \n", i);
    else {
        printf("error number %d \n", rc);
        exit (1);
    }
}

```

## SEE ALSO

dbflush(), dbgetnr(), dbgetr(), dbgetrk(), dbrecall(), dbrmvkey(), dbrmvr(),

## NAME

**dbdo()** **execute a RECITAL program**

## SYNOPSIS

```
#include "dbl.h"

int    dbdo(program, parameters)

<input parameters>
char   *program;           /* Address of a buffer containing a RECITAL
                           program name to execute          */
char   *parameters;        /* Address of a buffer containing up to nine
                           parameters separated with commas that
                           will be passed to program          */

<output parameters>
none
```

## RETURN VALUE

The dbdo() function returns 0 for success or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

The dbdo() function executes a RECITAL program specified in the buffer. Up to nine parameters may be passed to the RECITAL program. Each parameter must be separated by a comma. The parameters will be defined as public and will be named “\_PARA1” through “\_PARA9” in RECITAL.

## EXAMPLE

The following example runs a RECITAL program called “accounts.prg” passing three parameters.

```
#include "dbl.h"

int    rc;
static char program[] = "accounts.prg";
rc = dbdo(program , "para1, para2, para3");
if (rc != SUCCESS){
    printf("error no %d\ n", rc);
    exit(1);
}
```

## SEE ALSO

dbrun()





## EXAMPLE

The following example extracts the value from the field “char \*name” from the record buffer “char \*record” whose file descriptor is in “char \*dbf” and stores it in “char \*fldvalue” as an ASCII string.

```
#include "dbl.h"

char    *dbf;           /* .DBF file descriptor    */
char    record[1000];   /* Record buffer          */
char    name[10];       /* Name buffer            */
char    fldvalue[10];   /* Field value            */
int      rc;            /* Return code            */

rc = dbfield ( dbf, record, name, fldvalue);
if (rc != SUCCESS){
    printf("error no %d\n", RC);
    exit(1);
}
else printf("Value of field %s is %s\n", name, fldvalue);
```

## SEE ALSO

dbatofld(), dbgather(), dbgetf(), dbputr(), dbscatter()

## dbfilemode()

---

### NAME

**dbfilemode()**

**specify file operation mode**

### SYNOPSIS

```
#include "dbl.h"
```

```
int    dbfilemode(shared, readonly)
```

```
<input parameters>
```

```
int    shared;          /* 1 for shared file access otherwise 0    */
```

```
int    readonly;        /* 1 for readonly file access otherwise 0    */
```

```
<output parameters>
```

```
none
```

### RETURN VALUE

The dbfilemode() function returns 0 for success. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

The dbfilemode() function is used to determine how files will be opened. The first parameter specifies if the file is to be opened shared or for exclusive use. The second parameter specifies if the file will be opened read-only or with update privilege.

<p><b>NOTE:</b> when files are opened in a shareable mode, manual locking using library functions must be implemented.</p>
--

## EXAMPLE

The following example opens a database file with shared access and with read/write privilege.

```
#include "dbf.h"

int    dbf;          /* .DBF file descriptor    */
int    rc;           /* Return code           */

rc = dbfilemode(1,0);
if (rc == SUCCESS) printf("database opened \n");
    rc = dbopen("accounts.dbf", &dbf);
    if (rc == SUCCESS) printf("database opened \n");
    else {
        printf("error number %d\ n", rc);
    }
}
```

## SEE ALSO

dbiopen(), dblockf(), dblockr(), dblocki(), dbopen(), dbtlockf(), dbtlockr(), dbunlockf(), dbunlocki(), dbunlockr()

### NAME

**dbfldtoa()** **convert numeric field to ASCII**

### SYNOPSIS

```
#include "dbl.h"

int    dbfldtoa(field, width, ascii)

<input parameters>
char   *field;      /* Field to be converted          */
int    width;       /* Width of the field           */

<output parameters>
char   *ascii;      /* Address of the buffer where the converted
                        ASCII string is returned          */
```

### RETURN VALUE

The dbfldtoa() function returns 0 for success. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

This function converts a numeric field of a record containing a floating-point number to ASCII format. RECITAL/Library functions manipulate records/fields as ASCII data, therefore functions like this and others which allow data conversion must be utilized. The length of the resulting string equals the width of the field + 1, the string is null terminated.

NOTE: Make sure that the output character buffer is at least 1 byte longer than the "width" input parameter, since a NULL character is added to the output string. The value of "width" must be the same as the value specified when creating the .DBF file. If the value is forgotten, it can be obtained by calling the dbgetf() function.

## EXAMPLE

This example converts a numeric field of 10 digits wide located at “record + 29” into an ASCII character string and stores the result in “char ascii[11]”.

```
#include "dbl.h"

char  record[100];          /* Record buffer          */
char  ascii[11];           /* ASCII form of numeric field */
int    rc;                 /* Return code           */

rc = dbfldtoa(record + 29, 10, ascii);
if (rc == SUCCESS) printf("copy successful \n");
else {
    printf("error message %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbatofld(), dbcreat(), dbgetf(), dbkeytoa()

## NAME

**dbflush()** **flush I/O buffer for .DBF file**

## SYNOPSIS

```
#include "dbl.h"

int    dbflush(dbf)

<input parameter>
char   *dbf;          /* .DBF file descriptor      */

<output parameters>
none
```

## RETURN VALUE

The dbflush() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

This function forces the contents of the I/O buffer of the specified .DBF file to be written to the disk.

## EXAMPLE

This example flushes the I/O buffer of the .DBF file whose file descriptor is in "char \*dbf".

```
#include "dbl.h"
char    *dbf;          /* .DBF file descriptor    */
int      rc;           /* Return code             */

rc = dbflush(dbf);
if (rc == SUCCESS) printf(database buffer flushed \n");
else {
    printf("error number %d \n, rc);
    exit (1);
}
```

## SEE ALSO

dbappend(), dbdcache(), dbdelete(), dbgetr(), dbiflsh(), dbputr(), dbrecall(), dbupdr()





```

#include "dbl.h"
int    fd;          /* .NDX file descriptor      */
char   key[100];    /* Buffer for previous key   */
long   recnum;      /* Record number of previous key */

rc = dbfwd(fd);
if (rc == SUCCESS){
    rc = dbpkey(fd, key, &recnum);
    if (rc == SUCCESS) printf("Last key read \n");
    else {
        printf("error number %d \n", rc);
        exit (1);
    }
} else {
    printf("error number %d \n", rc);
    exit (1);
}

```

## SEE ALSO

dbckey(), dbnkey(), dbpkey(), dbrewind()

### NAME

**dbgather()**            **place individual field buffers back to the record buffer that was “scattered” using dbscatter()**

### SYNOPSIS

```
#include “dbl.h”
```

```
int     dbgather(dbf, record, fldbuf)
```

<input parameters>

```
int     dbf;                       /* Database file descriptor     */
char    record[1000];            /* Record buffer to refill      */
char    fldbuf[128][256];        /* Existing field buffers       */
```

<output parameters>

none

### RETURN VALUE

The dbgather() function returns 0 for success. Refer to the section on return codes for a detailed list of return code definitions.

### DESCRIPTION

The dbgather() function “gathers” individual field values stored in a two dimensional character array (fldbuf[128][256]) back into the record buffer (record[1000]). The field buffers are filled with dbscatter() function. The dbgather() function is similar to the dbrecin() function with the exception that it requires fewer input parameters.

## EXAMPLE

The following example retrieves (gathers) the data stored in the individual field buffers (fldbuf[128][255]) placed there with the dbscatter() function, and returns this information back to the record buffer (buffer[1000]).

```
#include "dbl.h"
int    dbf;           /* Database file descriptor */
char   record[1000];  /* Record buffer to fill   */
char   fldbuf[128][255]; /* Field buffers          */

rc = dbgather(dbf, record, fldbuf);
if (rc != 0) {

    printf("Error gathering record: %d\n", rc);
    exit(1);
}else{
    printf("Record gathered!\n");
}
```

## SEE ALSO

dbfield(), dbgcache(), dbgetr(), dbgetrk(), dbrecin(), dbrecout(), dbscatter()

### NAME

**dbgcache()**                    **enable the distributed cache manager on OpenVMS systems**

### SYNOPSIS

```
#include      "dbl.h"

int      dbgcache(mode)

<input parameters>
int      mode;          /* ON | OFF toggle, 0 – OFF, 1 – On      */

<output parameters>
none
```

### RETURN VALUE

This functions returns 0 upon completion.

### DESCRIPTION

This functions enables/disables the distributed cache manager on OpenVMS systems. Normally, if a database or index is opened shareable, any database or index caches specified are not active for the shared files. With the distributed cache manager activated, shared database and index caches are handled by the manager which allows separate processes access to the global cache. Note that proper ENQLM quotas for the processes instituting global caching are necessary.

## EXAMPLE

The following example enables global caching on a shared database. Note the use of `dbfilemode()` and `dbdcache()`.

```
#include "dbl.h"
int    dbf;          /* Database file descriptor */
int    rc;           /* Return Code variable */

rc = dbfilemode(1,0); /* Shared, Write access */
rc = dbdcache(100);  /* Database cache of 100 records */
rc = dbgcache(1);    /* Enable Global Caching */

if (rc = dbopen("dbctest.dbf", &dbf) != 0) {
    fprintf(stderr, "Database Open Failure: %d\n", rc);
    exit(1);
}
```

## SEE ALSO

`dbfilemode()`, `dbdcache()`, `dbicache()`

## NAME

**dbgetf()**

**get .dbf file information**

## SYNOPSIS

```
#include "dbl.h"
```

```
int    dbgetf(dbf, reclen, month, day, year, nofields, fields)
```

<input parameter>

```
char    *dbf;          /* Address of the .DBF file descriptor*/
```

<output parameters>

```
int      reclen;        /* Record length in bytes          */
```

```
char     *month;        /* File creation/update month     */
```

```
char     *day;          /* File creation/update day       */
```

```
char     *year;         /* File creation/update year      */
```

```
int      *nofields;     /* Number of fields in a record   */
```

```
dbfield  *fields;       /* Array of record fields         */
```

## RETURN VALUE

The dbgetf() functions 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

This function obtains database information such as month, day and year of its creation or last update, record length, number of fields per record and contents of each field. The buffer "fields" must be big enough to hold the maximum number of DBFIELD structures – 128 (maximum number of fields per record).

## EXAMPLE

This example reads database information from the .dbf file and prints record length, update information and number of fields per record on standard output.

```
#include<dbf.h>

char      *dbf;           /* .DBF file descriptor      */
int       reclen;         /* Record length            */
char      month, day, year; /* Date of last update      */
int       nofields;       /* Number of fields per record */
dBFIELD   fields[128];    /* Fields buffer            */
int       rc;             /* Return code              */
int       i;             /* For loop counter         */

rc = dbgetf(dbf,
            &reclen, &month,
            &day, &year,
            &nofields, fields);
if (rc == SUCCESS) {
    printf("reclen=%d month=%d day=%d year=%d,,\n", reclen, month, day, year, nofields);
    for (i = 0; i < nofields; ++i)
        printf("%d Field Name: %s", i + 1, fields[i].fieldnm);
} else {
    printf("error number %d\n", rc);
    exit [1];
}
```

## SEE ALSO

dbatofld(), dbcreat(), dbfield(), dbfldtoa(), dbgetfx(), dbgetr()

## NAME

**dbgetfx()**

**get extended .dbf file information**

## SYNOPSIS

```
#include<dbl.h>

int      dbgetfx(dbf, reclen, month, day, year, nofields, fields, fldpos, flddes)

<input parameter>
char      *dbf;                /* Address of the .DBF file descriptor*/

<output parameters>
int      reclen;                /* Record length in bytes          */
char      *month;               /* File creation/update month      */
char      *day;                 /* File creation/update day        */
char      *year;                /* File creation/update year       */
int      *nofields;             /* Number of fields in a record    */
DBFIELD   *fields;              /* Array of record fields          */
int      *fldpos;               /* Array of offsets where each field
                                commences*/
char      flddes[128][26];      /* Array to hold field descriptions of
                                each field*/
```

## RETURN VALUE

The dbgetfx() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

This function obtains database information such as month, day and year of creation or last update, record length, number of fields per record and definitions for each field. The buffer “fields” must be big enough to hold the maximum number of DBFIELD structures: 128 (maximum number of fields per record). The offset for each field position in the record is placed in the buffer “int \*fldpos”. The field description for each field is placed into “char flddes[128][26]”.



## EXAMPLE

This example reads database information from the .dbf file and prints record length, update information and number of fields per record on standard output.

```
#include "dbf.h"

char      *dbf;           /* .DBF file descriptor      */
int       reclen;         /* Record length             */
char      month, day, year; /* Date of last update       */
int       nofields;       /* Number of fields per record */
dBFIELD   fields[128];    /* Fields buffer             */
int       fldpos[128];    /* Fields position buffer     */
char      flddes[128][26]; /* Fields description buffer  */
int       rc;             /* Return code               */

rc = dbgetf (dbf, &reclen, &month, &day, &year,
             &nofields, fields, fldpos, flddes);
if (rc == SUCCESS) {
    printf("reclen=%d month=%d day=%d year=%d,
           nofields=%d \n",reclen, month, day, year,
           nofields);
} else{
    printf("error number %d\n", rc);
    exit (1);
}
```

## SEE ALSO

dbcreat(), dbcreatx(), dbgetf(), dbrecin(), dbrecout()

## NAME

**dbgetm()**

**get memo from memo file**

## SYNOPSIS

```
#include "dbl.h"
```

```
int    dbgetm(fd, field, memo, maxsize)
```

```
<input parameters>
```

```
int    fd;                /* .DBT file descriptor          */
```

```
char   *field;            /* Address of a 4 byte value that represents a  
                           memo field of a record in a .DBF file*/
```

```
int    maxsize;          /* Maximum bytes to read    */
```

```
<output parameter>
```

```
char   *memo;            /* Address of a user supplied memo buffer */
```

## RETURN VALUE

The dbgetm() function returns 0 for success. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

This function reads a memo from a .dbt file into the user buffer pointed to by the output parameter "memo". A user can define the maximum number of bytes that can be read/written to a memo file by the fourth parameter "maxsize". If maxsize is 0, then any size is read.

## EXAMPLE

The following example first reads the 3<sup>rd</sup> record from the database file into the record buffer “char record[1000];”, then reads a memo from the .dbt whose file descriptor is in “char \*dbt” into the memo buffer “char memo[5000]”. The corresponding record is assumed to contain its memo field starting at the 231st byte position.

```
#include "dbl.h"

char    *dbf;           /* Address of .DBF file descriptor */
char    *dbt;           /* Address of .DBT file descriptor */
char    record[1000];   /* Record buffer */
char    status;         /* Record status */
char    memo[5000];     /* Buffer to hold the read memo */
int     rc;             /* Return code */

rc = dbgetr(dbf, 3, record, &status);
if (rc == SUCCESS){
    rc = dbgetm(dbt, &record[231], memo, 0);
    if (rc == SUCCESS) printf("memo read\n");
    else {
        print("error message %\d", rc);
        exit (1);
    }
}
```

## SEE ALSO

dbmopen(), dbputm()

## NAME

**dbgetnr()**

**get next data record by key**

## SYNOPSIS

```
#include "dbl.h"

int    dbgetnr(dbf, ndx, record, status)

<input parameters>
char   *dbf;           /* .DBF file descriptor          */
char   *ndx;           /* .NDX file descriptor          */

<output parameters>
char   *record;        /* Address of the buffer where record is returned */
char   status;         /* Record status: if record is marked for deletion,
                        status is set to 0 (INACTIVE), otherwise
                        it is set to 1 (ACTIVE) */
```

## RETURN VALUE

The dbgetnr() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

This function sequentially reads a record in an ascending order. The first record number is obtained by reading the key to record number association table in the .NDX file. Then direct access of the .DBF file by record number is performed. After each sequential read operation, the access pointer to .NDX file is positioned at the next entry in the key to record number association table.

## EXAMPLE

This example reads a record and its status from a .DBF file that is indexed one position after the current record.

```
#include "dbl.h"

char    *dbf;           /* .DBF file descriptor          */
char    *ndx;           /* .NDX file descriptor          */
char    record[1000];   /* Record buffer                 */
char    status;         /* Record status: ACTIVE or INACTIVE */
int     rc;             /* Return code                    */

rc = dbgetnr(dbf, ndx, record, &status);
if (rc == SUCCESS) printf("next record read \n");
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbckey(), dbdcache(), dbdelete(), dbgetpr(), dbgetr(), dbnkey()

## NAME

**dbgetpr()**

**get previous record by key**

## SYNOPSIS

```
#include "dbl.h"

int    dbgetpr(dbf, ndx, record, status)

<input parameters>
char   *ndx;           /* .NDX file descriptor          */
char   *dbf;           /* .DBF file descriptor          */

<output parameters>
char   *record;         /* User buffer where record contents are returned */
char   status;          /* Status of record: ACTIVE or INACTIVE          */
```

## RETURN VALUE

The dbgetpr() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

This function sequentially reads a record in a descending (or counter alphabetical) order. First, the record number is obtained by reading the key to record number association table in the .NDX file. The direct access of .DBF file by record number is then performed. After each sequential read operation, the access pointer to .NDX file is positioned at the previous entry in the key to the record number association table.

## EXAMPLE

```
#include "dbl.h"

char    *dbf;           /* .DBF file descriptor          */
char    *ndx;           /* .NDX file descriptor          */
char    record[1000];   /* Record buffer                 */
char    status;         /* Record status: ACTIVE or INACTIVE */
int     rc;             /* Return code                    */

rc = dbgetpr(dbf, ndx, record, &status);
if (rc == SUCCESS) printf("previous record read \n");
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbckey(), dbgetnr(), dbgetr(), dbpkey()

<b>dbgetr()</b>	<b>get database record using record number</b>
-----------------	--

```
#include "dbl.h"

int      dbgetr(dbf, recno, record, status)

<input parameters>
char     *dbf;           /* .DBF file descriptor          */
long     recno;          /* Record number                 */

<output parameters>
char     *record;        /* Address of the record buffer   */
char     *status;        /* Status of record: ACTIVE or INACTIVE */
```

This function returns 0 for success, or  $< 0$  if an error occurs. See the section on return code values for a detailed list of return codes.

The function reads a record and its status from a data file. ACTIVE status means the record is not deleted, INACTIVE means that the record has been marked deleted but still exists physically. A disk-write failure (d\_WTFail) can be returned by the function because when this function is called, the buffer contents may be written onto the disk first and then a portion of data containing the requested record is transferred to the main memory.



## EXAMPLE

This example reads the 128th record from a .DBF file, whose file descriptor is in “char \*dbf” and prints its status on the standard outputs.

```
#include "dbl.h"

char    *dbf;          /* .DBF file descriptor      */
char    record[100];   /* Record buffer           */
char    status;        /* Record status           */
int     rc;            /* Return code              */

rc = dbgetr(dbf, 128, record, &status);
if (rc == SUCCESS) {
    switch (status) {
        case ACTIVE:
            printf("record 128 is active \n");
            break;
        case INACTIVE:
            printf("record 128 is marked deleted \n");
            break;
    }
} else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbappend(), dbcreat(), dbdcache(), dbdelete(), dbflush(), dbgather(), dbgetf(), dbgetnr(), dbgetpr(), dbgetrk(), dbputr(), dbrecall(), dbrecout(), dbscatter(), dbupdr()

## NAME

**dbgetrk()**

**get a data record by key**

## SYNOPSIS

```
#include "dbl.h"

int    dbgetrk(dbf, ndx, key, record, status)

<input parameters>
char   *dbf;           /* .DBF file descriptor          */
char   *ndx;           /* .NDX file descriptor          */
char   *key;           /* Address of a key buffer       */

<output parameters>
char   *record;        /* Address of a buffer where record's contents
                        are stored                               */
char   *status;        /* Status of record: ACTIVE or INACTIVE */
```

## RETURN VALUE

The dbgetrk() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of failure codes.

## DESCRIPTION

This function obtains a record from a .DBF file by key. First, the function looks for the specified key in the .NDX file and if found, obtains the corresponding record number. It then accesses the .DBF file directly by the record number and transfers the record to the user supplied buffer. The status of the record is returned in the user supplied variable.

## EXAMPLE

This example reads a record from a .DBF file whose file descriptor is specified in “char \*dbf” that matches the character key “021-70-9045”, stores the contents of the record in the buffer “char record[1000]” and the status of the record in the variable “char status”. The status is printed on the standard output.

```
#include "dbf.h"

char  *dbf;           /* .DBF file descriptor      */
char  *ndx;           /* .NDX file descriptor      */
char  key[20] = "021-70-9045"; /* Key                      */
char  record[1000];   /* Record buffer            */
char  status;         /* Status: ACTIVE or INACTIVE */
int    rc;            /* Return code              */

rc = dbgetrk(dbf, ndx, key, record, &status);
if (rc == SUCCESS) {
    printf("The status of the record matching the key,
           %s is %d \n", key, status);
} else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbdcache(), dbdelete(), dbgather(), dbgetr(), dbscatter(), dbtkey()

## dbgetseqno()

---

### NAME

**dbgetseqno()** **return the next sequence number**

### SYNOPSIS

```
#include "dbl.h"

int    dbgetseqno(dbf, seqno)

<input parameter>
char   *dbf;          /* Table file descriptor      */

<output parameters>
long   *seqno;         /* Address of a buffer in which the sequence
                        number is returned      */
```

### RETURN VALUES

The dbgetseqno() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

The dbgetseqno() function returns the next sequence number for the specified table.

### EXAMPLE

The following example sets the sequence number, then gets the sequence number.

```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file   */

static void errorproc(
    char    *func,
    char    *str,
    int     rc);

main()
{
    int      rc;          /* Return Code for error handling    */
    char     *dbf;        /* File descriptor for table         */
    long     *seqno;      /* Buffer address for sequence number */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbfilemode(1,0);
    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened shared.", rc);

    rc=dbsetseqno(dbf, (long) 49);
    errorproc("dbsetseqno()", "seqno set to 49.", rc);
    rc=dbgetseqno(dbf, seqno);
    errorproc("dbgetseqno()", "seqno retrieved.", rc);
    printf("\t\t\t Seqno: \t%d\n", *seqno);

    rc = dbclose(dbf);
    errorproc("dbclose()", "table closed.", rc);

    exit(0);
}

static void errorproc(func, str, rc)
char    *func;
char    *str;
int     rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## **SEE ALSO**

`dbsetseqno()`

## **dbicache()**

---

### **NAME**

**dbicache()** specify size of the index cache

### **SYNOPSIS**

```
#include "dbl.h"

int    dbicache(size)

<input parameter>
int    size;          /* Index cache size in blocks */

<output parameters>
none
```

### **RETURN VALUE**

The dbicache() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### **DESCRIPTION**

The dbicache() function is used to specify the size of the index cache in 512 byte disk blocks. Index caching improves performance of index operations.

### **EXAMPLE**

The following example sets the icache (index cache) to 100 then creates an .NDX file "person.ndx" whose key expression, length and type are "SSN", 10 and 'C' respectively. "SSN" is a field name of the associated .DBF file.

```

#include "db1.h"
int    rc;    /* Return code */

rc = dbicache(100);
if (rc == SUCCESS){
    rc = dbicreat("person.ndx", "SSN", 10, 'C');
    if (rc == SUCCESS){
        printf("index person created \n");
    } else {
        printf("error number %d \n, rc);
        exit (1);
    }
} else {
    printf("error number %d \n, rc);
    exit (1);
}

```

## SEE ALSO

dbdcache(), dbiclose(), dbicreat(), dbiopen()



## **dbiclose()**

---

### **NAME**

**dbiclose()**

**close index file**

### **SYNOPSIS**

```
#include "dbl.h"

int    dbiclose(ndx)

<input parameter>
char   *ndx;          /* .NDX file to be closed    */

<output parameters>
none
```

### **RETURN VALUE**

The dbiclose() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### **DESCRIPTION**

This function closes the specified index file.

### **EXAMPLE**

The following example closes an .NDX file whose file descriptor is specified in “char \*ndx”.

```
#include "dbl.h"

char    *ndx;          /* .NDX file descriptor    */
int      rc;           /* Return code            */

rc = dbiclose(ndx);
if (rc == SUCCESS) printf("index closed \n");

else (
    printf("error number %d \n");
    exit (1);
}
```

## SEE ALSO

dbclose(), dbicache(), dbiopen()

## NAME

**dbicreat()**

**create an index file**

## SYNOPSIS

```
#include "dbl.h"

int    dbicreat(ndxname, keyexpr, keylen, keytype)

<input parameter>
char   *ndxname;    /* Address of a buffer containing the name of the
                    .NDX file to be created */
char   *keyexpr;    /* Address of a buffer containing the key expression */
int     keylen;      /* Index key length */
char    keytype;     /* Index key data type */

<output parameters>
none
```

## RETURN VALUE

The dbicreat() function returns 0 for success, or < 0 if an error occurs. See the section on return values for a detailed list of return codes.

## DESCRIPTION

This function creates an .NDX file. A key provided by a user is usually a character string that consists of one or more field names of a data record, e.g., "NAME+SSN".

**NOTE:** This function only creates the header of the index file. Other library functions (i.e. dbakey()) must be used to fill the index with the appropriate keys.

## **EXAMPLE**

The following example creates an .NDX file “person.ndx” whose key expression, length and type are “SSN”, 10 and ‘C’ respectively. “SSN” is a field name of the associated .DBF file.

```
#include "dbl.h"

int    rc;    /* Return code */

rc = dbicreat("person.ndx", "SSN", 10, 'C');
if (rc == SUCCESS) printf("index person created \n");
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## **SEE ALSO**

dbakey(), dbcreat(), dbicache(), dbkexpr(), dbputrk()

## NAME

**dbiflsh()**

**flush I/O buffer cache for index file**

## SYNOPSIS

```
#include "dbl.h"

int    dbiflsh(ndx)

<input parameter>
char   *ndx;          /* .NDX file descriptor */

<output parameters>
none
```

## RETURN VALUE

The dbiflsh() function return 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

This function forces the I/O buffer cache of the specified .NDX file to be written to the disk.

## EXAMPLE

The following example flushes the .NDX file whose file descriptor is in “char \*ndx”.

```
#include "dbl.h"
char    *ndx;        /* .NDX file descriptor    */
int     rc;          /* Return code          */

rc = dbiflsh(ndx);
if (rc == SUCCESS) printf("index buffer flushed \n");
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbakey(), dbflush(), dbiopen(), dbputrk()

## **dbiopen()**

---

### **NAME**

**dbiopen()**

**open an existing index file**

### **SYNOPSIS**

```
#include "dbl.h"

int    dbiopen(name, ndx)

<input parameter>
char   *name;          /* Address of a buffer containing the .NDX
                        file name                                */

<output parameters>
char   *ndx;           /* .NDX file descriptor                                */
```

### **RETURN VALUE**

The dbiopen() function returns 0 for success, or < 0 if an error occurs. See the section on return value codes for a detailed list of failure codes.

### **DESCRIPTION**

This function opens an existing .NDX file and returns its file descriptor. The file descriptor is released after the dbiclose() function has been called for this file.

### **EXAMPLE**

The following example opens the .NDX file "person.ndx". The file descriptor is returned by the function dbiopen() in "char \*ndx".

```
#include "dbl.h"

char    *ndx;          /* .NDX file descriptor    */
int      rc;           /* Return code             */

rc = dbiopen("person.ndx", &ndx);
if (rc == SUCCESS) printf("index opened \n");
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbfilemode(), dbicache(), dbiclose(), dbiflsh(), dbopen()



## NAME

**dbkexpr()**

**get key expression**

## SYNOPSIS

```
#include "dbl.h"

int    dbkexpr(ndx, keytype, keyexpr, keyprlen, keylen)

<input parameters>
char   *ndx;           /* .NDX file descriptor          */

<output parameters>
char   keytype;        /* Index key data type:
                        'C' – CHARACTER
                        'N' – NUMERIC
                        'D' – DATE*/
char   *keyexpr;       /* Index key expression          */
int     *keyprlen;     /* Length of index key expression */
int     *keylen;       /* Index key length              */
```

## RETURN VALUE

The dbkexpr() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

This function obtains the index information from the specified .NDX file. This information includes:

- index key data type('C' for CHARACTER keys, 'N' for NUMERIC and 'D' for DATE),
- index key expression (usually a character string consisting of one or more record field names)
- length of the index key expression
- length of the key.

## EXAMPLE

This example obtains the index information from the .NDX file whose file descriptor is in “char \*ndx” and prints this information on standard output.

```
#include "dbl.h"

char    *ndx;           /* .NDX file descriptor    */
char    keytype;        /* Key type              */
char    keyexpr[512];   /* Index key expression   */
int     keyexprlen;     /* Expression length      */
int     keylen;         /* Key length             */
int     rc;            /* Return code            */

rc = dbkexpr(ndx, &keytype, keyexpr, &keyexprlen,
             &keylen);
if (rc == SUCCESS){
    printf("\t Index file information - \n");
    printf("\t \t type = '%c', keyexpression = '%s',
           \t \t exprlen = %d, keylen = %d \n". keytype,
           keyexpr, keyexprlen, keylen);
} else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbicreat()

## dbkeytoa()

---

### NAME

**dbkeytoa()**

**convert numeric/data key value to ASCII**

### SYNOPSIS

```
#include "dbl.h"
```

```
int    dbkeytoa(key, ascii)
```

```
<input parameters>
```

```
char   *key;           /* Key to be converted                */
char   ascii[0];       /* Number of decimal places for numeric key */
char   ascii[1];       /* Key type: 'D' – for DATE, 'N' – for NUMERIC */
```

```
<output parameters>
```

```
char   *ascii;         /* Address of an ASCII string representing the
                        converted key value                */
```

NOTE: "ascii" is used in the input and output parameter.
--

### RETURN VALUE

The dbkeytoa() function returns 0 for success, or < 0 if an error occurs. See the section on return code value for a detailed list of return codes.

### DESCRIPTION

This function converts a key of type NUMERIC or DATE into a null terminated ASCII string. If the key is of NUMERIC type the user must pass the number of decimal places to the function via the parameter ascii[0]. The key type is specified in the parameter ascii[1]. Function returns an error if the specified type is neither NUMERIC ('N' or 'n') nor DATE ('D' or 'd'). This function is particular useful if the user wants to examine a numeric or date key obtained by the dbkey() or dbnkey() functions.

## EXAMPLE

The first example converts an .NDX file numeric key stored in “char key[8]” to an ASCII string which is placed in “char ascii[64]”. The converted ASCII string has a precision of 3 decimal places. The converted key is printed on standard output.

```
#include "dbl.h"

char   key[8];           /* Key to be converted           */
char   ascii[64];        /* Buffer for converted key      */
int     rc;              /* Return code                  */

ascii[0] = 3;            /* Specify number of decimal places */
ascii[1] = 'N';          /* Indicate the key is NUMERIC    */

rc = dbkeytoa(key, ascii);
if (rc == SUCCESS) printf("the key is '%s' \n", ascii);
else {
    printf("error number %d \n", rc);
    exit (0);
}
```

The second example reads the previous date key from the .NDX file specified by “char \*ndx;”, converts it from the .NDX date format to the ASCII date format and prints the key on the standard output.

```
#include "dbl.h"

char   *ndx;             /* .NDX file descriptor         */
char   prevkey[8]        /* Previous key buffer           */
long    recno;           /* Record no. associated with previous key */
char   ascii[10];        /* Ascii string buffer          */
int     rc;              /* Return code                  */
ascii[1] = 'D';          /* Indicate the date key        */
```

```
rc = dbpkey(ndx, prevkey, &recno);
if (rc == SUCCESS){
    rc = dbkeytoa(prevkey, ascii);
    if (rc == SUCCESS)
        printf("The previous key is %s \n", ascii);
    else{
        printf("error number %d \n", rc);
        exit (1);
    }
}
```

## **SEE ALSO**

dbatokey(), dbatofld(), dbfldtoa(), dbpkey()

### NAME

**dblockf()** **lock a database file**

### SYNOPSIS

```
#include "dbl.h"

int    dblock(dbf)

<input parameter>
char   *dbf;          /* File descriptor of a .DBF file to be locked */

<output parameter>
none
```

### RETURN VALUE

The dblockf() function returns 0 for success, or < 0 if an error occurs. See the section on return values for a detailed list of return codes.

### DESCRIPTION

The dblockf() function locks the database file from the given file descriptor "char \*dbf".

### EXAMPLE

The following example locks the .DBF file whose file descriptor is in "char \*dbf".

```
#include "dbl.h"
char    *dbf;          /* Database file descriptor */
int      rc;           /* Return code */

rc = dblockf(dbf);
if (rc = SUCCESS) printf("file is locked \n");
else {
    printf("file could not be locked \n");
    return (ERROR);
}
```

## SEE ALSO

dbfilemode(), dblocki(), dblockr(), dbtlockf(), dbtlockr(), dbunlockf(), dbunlocki(), dbunlockr()

## dblocki()

---

### NAME

**dblocki()**

**lock an index file**

### SYNOPSIS

```
#include "dbl.h"

int    dblocki(ndx)

<input parameter>
char   *ndx;          /* File descriptor of a .NDX file to be locked   */

<output parameter>
none
```

### RETURN VALUE

The dblocki() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

The dblocki() function locks the index file from the given file descriptor "char \*ndx".

### EXAMPLE

The following example locks the .NDX file whose file descriptor is in "char \*ndx".

```
#include "dbl.h"
char   *ndx;          /* Index file descriptor   */
int     rc;           /* Return value            */

rc = dblocki(ndx);
if (rc = SUCCESS) printf("index is locked \n");
else {
    printf("index could not be locked \n");
    return (ERROR);
}
```



**SEE ALSO**

dbfilemode(), dblockf(), dblockr(), dbtlockf(), dbtlockr(), dbunlockf(), dbunlocki(), dbunlockr()

### NAME

**dblockr()**

**lock a record in a database file**

### SYNOPSIS

```
#include "dbl.h"

int      dblockr(dbf, recno)

<input parameter>
char      *dbf;          /* File descriptor of a .DBF file to be locked */
int      recno;          /* Record number to lock */

<output parameter>
none
```

### RETURN VALUE

The `dblockr()` function returns 0 for success, or < 0 if an error occurs. See the section on return code value for a detailed list of return codes.

### DESCRIPTION

The `dblockr()` function locks a record in the database file from the given file descriptor “char \*dbf”. The record locked is the number in “int recno”.

### EXAMPLE

The following example locks record number 5 in the .DBF file whose file descriptor is in “char \*dbf”.

```
#include "dbf.h"
char    *dbf;          /* Database file descriptor */
int     rc;             /* Return code */

rc = dblockr(dbf, 5);
if (rc = SUCCESS) printf("record 5 is locked \n");
else {
    printf("record 5 could not be locked \n");
    return (ERROR);
}
```

## SEE ALSO

dbfilemode(), dblockf(), dblocki(), dbtlockf(), dbtlockr(), dbunlockf(), dbunlocki(), dbunlockr()

## dbmclose()

---

### NAME

**dbmclose()**

**close memo file**

### SYNOPSIS

```
#include "dbl.h"

int    dbmclose(dbt)

<input parameter>
char   *dbt;          /* File descriptor of a .DBT file to be closed */

<output parameter>
none
```

### RETURN VALUE

The dbmclose() function returns 0 for success. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

This function closes a .DBT memo file.

### EXAMPLE

The following example closes a .DBT file whose file descriptor is in “char \*dbt”.

```
#include "dbl.h"
char    *dbt;          /* File descriptor of a .DBT file    */
int      rc;           /* Return code                      */

rc = dbmclose(dbt);
if (rc == SUCCESS)
    printf("the memo has been closed.\n");
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbmopen(), dbmcreat()

## **dbmcreat()**

---

### **NAME**

**dbmcreat()**

**create a memo file**

### **SYNOPSIS**

```
#include "dbl.h"

int    dbmcreat(dbtname)

<input parameter>
char   *dbtname;    /* Address of a buffer containing the name
                    of a memo file to be created          */

<output parameter>
none
```

### **RETURN VALUE**

The dbmcreat() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### **DESCRIPTION**

This function creates a .DBT file. The memo file name must have the same base name as the database table that it belongs to.

### **EXAMPLE**

The following example creates a .DBT file.

```
#include "dbl.h"
rc = dbmcreat("memo.dbt");
if (rc == SUCCESS) printf("memo file created \n");
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## **SEE ALSO**

dbcreat(), dbicreat(), dbmclose(), dbmopen()

## dbmemoformat()

---

### NAME

**dbmemoformat()**

**toggle memo input formatting**

### SYNOPSIS

```
#include "dbl.h"

int      dbmemoformat(format)

<input parameters>
int      format;           /* Memo Format flag, 0 – OFF, 1 – ON      */

<output parameters>
none
```

### RETURN VALUE

The dbmemoformat() function returns 0.

### DESCRIPTION

Memo formatting automatically formats memo field data when placed into the file. Spaces and other control characters are stripped if memo format is ON. This function allows access to this characteristic from within the libraries. If the input parameter is 0, function dbputm() and dbupdm() attempt no formatting of memos. Otherwise, those functions strip spaces and control characters.



## EXAMPLE

This example toggles MEMOFORMAT ON. The buffer that is to be written to the memo field will be stripped of extra spaces and control characters.

```
#include "dbl.h"
int    dbt;                /* .DBT file descriptor    */
char   memo[512];         /* Memo buffer             */
int     rc;                /* Return code variable    */

rc = dbmemoformat(1)
if( rc = dbupdm(dbt, memo,
                &record[53],
                &record[53]) != 0) {
    fprintf(stderr, "Error on dbupdm(): %d\n", rc);
    exit(1);
}
```

## SEE ALSO

dbgetm(), dbmopen(), dbputm(),dbupdm()

## dbmopen()

---

### NAME

**dbmopen()**

**open an existing .DBT file**

### SYNOPSIS

```
#include "db1.h"

int      dbmopen(dbtname, dbt)

<input parameters>
char    *dbtname;    /* Address of a buffer containing the name
                      of a .DBT file to be opened          */

<output parameter>
char    *dbt;        /* The .DBT file descriptor          */
```

### RETURN VALUE

The dbmopen() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

This function opens a .DBT file whose name is passed to the function. The .DBT file descriptor is returned. A call to the dbmclose() function which closes a .DBT file releases the file descriptor for reuse.

### EXAMPLE

This example opens an existing memo file "MEMO.DBT" and returns its file descriptor in "char \*dbt".

```
#include "db1.h"
char    *dbt;          /* .DBT filw descriptor    */
int      rc;           /* Return code            */

rc = dbmopen("memo.dbt", &dbt);
if (rc == SUCCESS) printf("memo opened \n");
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbgetm(), dbmclose(), dbmcreat(), dbopen(), dbputm()

## NAME

**dbnkey()**

**read next key**

## SYNOPSIS

```
#include "dbl.h"

int    dbnkey(ndx, key, recno)

<input parameter>
char   *ndx;           /* .NDX file descriptor          */

<output parameters>
char   *key;           /* Address of a buffer where key is stored
                        by the function                      */
long   *recno;         /* Address of a variable where the record
                        number is stored by the function     */
```

## RETURN VALUE

The dbnkey() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

This function sequentially reads the next in ascending (or alphabetical) order and its corresponding record number from the specified .NDX file. If the function is called immediately after the file is open, it starts reading from the top of the file.

## EXAMPLE

This example reads ten keys and their corresponding record numbers from the .NDX file whose file descriptor is specified in “char \*ndx” and stores the keys and record numbers in “char nextkey[10][8]” and “long recno[10]” respectively. Keys are assumed to be numeric or date.

```
#include "dbl.h"

char   *ndx;           /* .NDX file descriptor    */
char   nextkey[10][8]; /* Keys buffer           */
long   recno[10];      /* Record numbers buffer  */
int     it;            /* Loop control variable  */
int     rc;            /* Return code            */

for (i = 0; i < 10; ++i)
    if (dbnkey(ndx, &nextkey(i), &recno(i)) != SUCCESS)
        return(ERROR);
```

## SEE ALSO

dbckey(), dbfwd(), dbgetnr(), dbpkey(), dbrewind()

## **dbonerror()**

---

### **NAME**

**dbonerror()**                                      **enable/disable recovery from I/O errors**

### **SYNOPSIS**

```
#include "dbl.h"

int    dbonerror(mode)

<input parameters>
int    mode;          /* 0 disables and 1 enables */

<output parameters>
none
```

### **RETURN VALUE**

The dbonerror() function has no return values.

### **DESCRIPTION**

This function enables recovery from I/O errors in all of the RECITAL/Library functions. By disabling error checking, the Library function calls operate faster.

### **EXAMPLE**

The following example disables recovery from I/O errors.

```
#include "dbl.h"

dbonerror(0);
```

### **SEE ALSO**

**dbopen()**

## NAME

**dbopen()** **open an existing .DBF file**

## SYNOPSIS

```
#include "dbl.h"

int    dbopen(dbfname, dbf)

<input parameter>
char   *dbfname;    /* Address of a buffer containing the name of a .DBF
                    file to be opened          */

<output parameter>
char   *dbf;        /* The .DBF file descriptor    */
```

## RETURN VALUE

The dbopen() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

This function opens a .DBF file and returns its file descriptor. A call to the dbclose() function releases the file descriptor for use by some other .DBF file.

## EXAMPLE

The following example opens a .DBF file "LIBRARY.DBF" and stores its file descriptor in "char \*dbf".

```
#include "dbl.h"
char    *dbf;          /* .DBF file descriptor    */
int      rc;           /* Return code             */

rc = dbopen("library.dbf", &dbf);
if (rc == SUCCESS) printf("database file open \n");
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbclose(), dbfilemode(), dbiopen(), dbmopen(), dbonerror()



## NAME

**dbpkey()**

**read previous key**

## SYNOPSIS

```
#include "dbl.h"
```

```
Int    dbpkey(ndx, key, recno)
```

```
<input parameter>
```

```
char    *ndx;          /* Address of the .NDX file descriptor  
                        from which to read a key          */
```

```
<output parameters>
```

```
char    *key;          /* Address of the buffer where the key  
                        is stored by the function          */
```

```
long    *recno;        /* Address of the variable where the corresponding  
                        record number is stored by the function */
```

## RETURN VALUE

The dbpkey() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

This function sequentially reads the previous key in a descending (or counter-alphabetical) order and its associated record number from the .NDX file whose file descriptor is passed to the function.

## EXAMPLE

This example reads the previous key and its associated record number from a .NDX file whose file descriptor is in “char \*ndx”. The key and record number are stored in “char prevkey[18];” and “long recno;” respectively. The key is assumed to be date or numeric.

```
#include    <dbl.h>

char    *ndx;           /* .NDX file descriptor      */
char    prevkey[8];     /* Key buffer              */
long    recno;          /* Record no. associated with key */
int      rc;            /* Return code             */

rc = dbpkey(ndx, prevkey, &recno);
else {
    printf("error number %d \n, rc);
    exit (1);
}
```

## SEE ALSO

dbckey(), dbkeytoa(), dbfwd(), dbgetpr(), dbnkey(), dbrewind()

## NAME

**dbputm()**

**put memo in memo file**

## SYNOPSIS

```
#include "dbl.h"

int    dbputm(dbt, memo, field)

<input parameters>
char   *dbt;           /* .DBT file descriptor          */
char   *memo;          /* Address of a buffer containing the memo
                        to be put into the field          */

<output parameters>
char   *field;         /* Address of the memo field      */
```

## RETURN VALUE

The dbputm() function returns 0 for success. See the section on return value codes for a detailed list of failure codes.

## DESCRIPTION

This function writes a memo from a specified buffer to the specified .DBT file. The contents of the output parameter "field" indicate where the memo is put in the .DBT file. These contents should be identical to the contents of a memo field of a data record when it is written or updated in the .DBT file. This function does not automatically update the memo field of the corresponding .DBF file: the functions dbputr() or dbupdr() must be used for this purpose.

## EXAMPLE

This example writes a memo from the buffer “char memo[512]” to the .DBT file whose file descriptor is in “char \*dbt” and then writes a record, with the record number 200, whose contents are in “char record[1000]” to the .DBF file. The record is assumed to contain its memo field starting from the 235th byte.

```
#include "dbl.h"

char    *dbt;           /* .DBT file descriptor    */
char    *dbf;           /* .DBF file descriptor    */
char    memo[512];      /* Memo buffer            */
char    record[1000];   /* Record buffer          */
int      rc;             /* Return code            */

rc = dbputm(dbt, memo, &record[234]);
if (rc == SUCCESS){
    rc = dbputr(dbf, 200, record);
    if (rc == SUCCESS) printf("memo buffer written \n");
    else {
        printf("error number %d \n", rc);
        exit (1);
    }
}
```

## SEE ALSO

dbgetm(), dbmopen(), dbputr(), dbupdm()

## NAME

**dbputr()**

**put data record into a .DBF file**

## SYNOPSIS

```
#include "dbl.h"
```

```
int    dbputr(dbf, recno, record)
```

```
<input parameters>
```

```
char    *dbf;           /* .DBF file descriptor      */
long    recno;          /* Record number            */
char    *record;        /* Address of a buffer where
                        the record contents are stored */
```

```
<output parameters>
```

```
none
```

## RETURN VALUE

The dbputr() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

This function writes the contents of the record supplied in a user defined buffer to a .DBF file. The position of the record within the file is determined by the record number. All the records existing in the file starting from this position (if any) will be shifted down to create a space for the new record. For example, if the record to be written has the number 5, and 8 records have been previously written to the file, the 5th, 6th, 7th and 8th records will become 6th, 7th, 8th and 9th respectively. The new record becomes the 5th record.

## EXAMPLE

The first example adds the 56th record whose contents are in the “char record[100]” to the .DBF file whose file descriptor is in “char \*dbf”.

```
#include "dbl.h"

char    *dbf;           /* .DBF file descriptor    */
char    record[100];    /* Record buffer          */
int      rc;            /* Return code             */

rc = dbputr(dbf, (long) 56, record);
if (rc == SUCCESS) printf("Record added \n");
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

The second example adds a record to the end of the .DBF file. The function dbsize() is used to get the size of the .DBF file (the number of the last record).

```
#include "dbl.h"

char    *dbf;           /* .DBF file descriptor    */
char    record[100];    /* Rrecord buffer          */
long     recno;          /* Record number           */
int      rc;            /* Return code             */

dbsize(dbf, &recno);
rc = dbputr(dbf, recno + 1, record);
if (rc == SUCCESS) printf("record added \n");
else {
    printf("error number %d\n", rc);
    exit (1);
}
```

## SEE ALSO

dbappend(), dbfield(), dbflush(), dbgetr(), dbputm(), dbputrk(), dbrecin(), dbsize(), dbupdr()

## NAME

**dbputrk()** **put data record into a .DBF file by key**

## SYNOPSIS

```
#include "dbl.h"
```

```
int dbputrk(dbf, ndx, key, record)
```

```
<input parameters>
```

```
char *dbf;          /* .DBF file descriptor */
char *ndx;          /* .NDX file descriptor */
char *key;          /* Key buffer */
char *record;       /* Record buffer */
```

```
<output parameters>
```

```
none
```

## RETURN VALUE

The dbputrk() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

This function adds a key to a .NDX file and its associated record to a .DBF file. First the function checks if the specified key already exists in the .NDX file. If it does, its associated record number is obtained and the record is written to the .DBF file. If not, the key is added to the .NDX file first and then the record is written to the end of the .DBF file.

## EXAMPLE

This example adds a record whose contents are in “char record[100]” to the .DBF file whose file descriptor is in “char \*dbf”. It also indexes the file with the contents of “char key[8]” which is written to the file specified in “char \*ndx”.

```
#include "dbl.h"

char  *dbf;           /* .DBF file descriptor */
char  *ndx;           /* .NDX file descriptor */
char  key[8];         /* Key buffer */
char  record[100];    /* Record buffer */
int    rc;            /* Return code */

rc = dbputrk(dbf, ndx, key, record);
if (rc == SUCCESS) printf("record added \n");
else {
    printf("error number %d \n", rc);
    exit (1)
}
```

## SEE ALSO

dbappend(), dbicreat(), dbiflsh(), dbputr()



### NAME

**dbrecall()** recall a previously deleted record

### SYNOPSIS

```
#include "dbl.h"

int    dbrecall(dbf, recno)

<input parameters>
char    *dbf;          /* .DBF file descriptor    */
long    recno;         /* Record number          */

<output parameter>
none
```

### RETURN VALUE

The dbrecall() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

This function recalls or restores the ACTIVE status of a record that has been previously marked deleted.

### EXAMPLE

This example recalls the 15th record in the .DBF file whose file descriptor is in "char \*dbf"

```
#include "dbl.h"

char    *dbf;          /* .DBF file descriptor    */
int      rc;           /* Return code             */

rc = dbrecall(dbf, 15);
if (rc == SUCCESS) printf("record recalled \n");
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbdcache(), dbdelete(), dbflush(), dbgetr()

## NAME

**dbrecin()**                                      **fill a single record buffer from field buffers**

## SYNOPSIS

```
#include "dbl.h"

int      dbrecin(dbf, fields, fldpos, buffer, fldbuffer)

<input parameters>
char      *dbffd;           /* Database file descriptor      */
DBFIELD   fields;          /* Array of record fields       */
int        *fldpos;         /* Array of field offsets       */
char      *buffer;         /* Address of a buffer where record
                           starts
char      fldbuffer[128][256]; /* Address of a buffer from which to
                           update the record

<output parameters>
none
```

## RETURN VALUE

The dbrecin() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

The dbrecin() function fills component fields in a record from the individual field buffers specified. The “fields” and “fldpos” are provided by the dbgetfx() function. This provides a simple mechanism for transferring data to and from records with the use of dbputr(). If the individual field buffer is shorter than the expected length of the field, dbrecin() will pad with the appropriate blank spaces to the left.

## EXAMPLE

This example fills the buffer record with field data stored in “char fieldbuf”.

```
#include "dbl.h"

int      nfields;           /* Number of fields */
dbFIELD  fields[128];       /* Return code      */
int      fieldpos[128];     /* Field positions   */
char     record[1000];      /* Record buffer     */
char     fieldbuf[128][256]; /* Field buffers     */

rc = dbrecin(nfields, fields, fieldpos, record, fieldbuf);
if (rc == SUCCESS) printf("record filled \n");
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbgather(), dbgetfx(), dbputr(), dbrecout()

### NAME

**dbrecout()**                      **split the record buffer into individual field components**

### SYNOPSIS

```
#include "dbl.h"
```

```
int     dbrecout(dbf, fields, fldpos, buffer, fldbuf)
```

```
<input parameters>
```

```
int           *dbffd;               /* Database file descriptor*/
DBFIELD       *fields;             /* Array of record fields*/
int           *fldpos;             /* Array of field offsets*/
char          *buffer;             /* Address of a buffer where record
                                     starts */
char          fldbuf[128][256];     /* Buffers to update from the
                                     record */
```

```
<output parameters>
```

```
none
```

### RETURN VALUE

The dbrecout() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

The dbrecout() function splits a record buffer into its component fields by placing the individual fields into 'fldbuf'. The 'fields' and 'fldpos' are provided by the dbgetfx() function.

## EXAMPLE

This example splits the record “dBFIELD fields” with ASCII information stored in “char fieldbuf[128][256]”.

```
#include "dbl.h"

char      *dbf;           /* .DBF file descriptor      */
dBFIELD   fields[128];    /* Return code              */
int        fieldpos[128]; /* Field positions          */
char       record[1000];  /* Record buffer            */
char       fieldbuf[128][256]; /* Field buffers          */

rc = dbrecout(dbf, fields, fieldpos, record, fieldbuf);
if (rc == SUCCESS) printf("record filled \n");
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbgather(), dbgetfx(), dbgetr(), dbrecin(), dbscatter()

## dbrewind()

---

### NAME

**dbrewind()**

**rewind index file**

### SYNOPSIS

```
#include "dbl.h"

int    dbrewind(ndx)

<input parameter>
char   *ndx;          /* .NDX file descriptor */

<output parameter>
none
```

### RETURN VALUE

The dbrewind() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

This function positions the access pointer at the beginning of the .NDX file. The subsequent dbnkey() locates the first key.

### EXAMPLE

This example rewinds the .NDX file whose file descriptor is in "char \*ndx" and then reads the first key from the file.

```

#include "dbl.h"
char  *ndx;          /* .NDX file descriptor */
char  key[100];      /* Key buffer */
long  recnum;        /* Record number */
int    rc;           /* Return code */

rc = dbrewind(ndx);
if (rc == SUCCESS){
    rc = dbnkey(nfxfd, key, recnum);
    if (rc == SUCCESS) printf("key read \n");
    else {
        printf("error number %d \n", rc);
        exit(1);
    }
}

```

## SEE ALSO

dbckey(), dbfwd(), dbnkey(), dbpkey()



## **dbrmvkey()**

---

### **NAME**

**dbrmvkey()**

**remove key from index file**

### **SYNOPSIS**

```
#include "dbl.h"
```

```
int    dbrmvkey(ndx(ndx, key, recno)
```

```
<input parameters>
```

```
char    *ndx;          /* .NDX file descriptor          */
```

```
char    *key;          /* Address of a buffer containing  
                        the key to be removed          */
```

```
long    recno;         /* Record number associated with the  
                        key to be removed          */
```

```
<output parameter>
```

```
none
```

### **RETURN VALUE**

The dbrmvkey() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### **DESCRIPTION**

This function physically removes a key and its associated record number from a .NDX file.

## EXAMPLE

This example removes a character key specified in “char key[20]” and its associated record number specified in “long recno” from a .NDX file whose file descriptor is in “char \*ndx”.

```
#include "dbl.h"

char   *ndx;          /* .NDX file descriptor      */
char   key[20];       /* Character key to be removed */
long   recno;         /* Record number              */
int     rc;           /* Return code                 */

rc = dbrmvkey(ndx, key, recno);
if (rc == SUCCESS) printf("key removed \n");
else {
    printf("error number %d \n", rc);
    exit(1);
}
```

## SEE ALSO

dbakey(), dbtkey(), dbupdr()

<b>dbrmvr()</b>	<b>physically remove a record from a data file</b>
-----------------	--

```
#include "dbl.h"

int      dbrmvr(dbf, recno)

<input parameters>
char     *dbf;           /* .DBF file descriptor          */
long     recno;          /* Number of a record to be removed */

<output parameters>
none
```

The `dbmvr()` function returns 0 for success, or  $< 0$  if an error occurs. See the section on return code values for a detailed list of return codes.

This function physically removes a record from a .DBF file. All the records whose record numbers are greater than the record number of the deleted record are shifted up. For example, if the 18th record is requested to be deleted, and 20 records currently exist in the database, the 19th and 20th will become 18th and 19th respectively.

## EXAMPLE

The following example physically removes the 5002nd record from the data file whose file descriptor is in “char \*dbf”

```
#include "dbl.h"

char    *dbf;          /* .DBF file descriptor    */
int      rc;           /* Return code            */

rc = dbrmvr(dbf, (long) 5002);
if (rc == SUCCESS) printf("record removed \n");
else {
    printf("error number %d \n", rc);
    exit (1);
}
```

## SEE ALSO

dbdelete()

**dbrun()** execute an operating system command

```
#include <dbl.h>

int      dbrun(command)

<input parameters>
char     *command;           /* Address of a buffer containing an
                              operating system command */

<output Parameters>
none
```

The `dbrun()` function returns 0 for success or  $< 0$  if an error occurs. See the section on return code values for a detailed list of return codes.

The `dbrun()` function executes an operating system command. Any valid operating system command can be specified.

The first example runs the UNIX “rm” command.

```
#include "dbl.h"

int      rc;      /* Return code */

rc = dbrun("rm *.bak");
if (rc != SUCCESS){
    printf("error no %d\n", rc);
    exit(1);
}
```

The second example runs the OpenVMS “purge” command.

```
#include "dbl.h"

int    rc;    /* Return code */

rc = dbrun("purge/log");
if (rc != SUCCESS){
    printf("error no %d\n", rc);
    exit(1);
}
```

## SEE ALSO

dbdo()

### NAME

**dbscatter()**                      **split the record buffer into individual field buffers**

### SYNOPSIS

```
#include "dbl.h"
```

```
int     dbscatter(dbf, record, fldbuf)
```

<input parameters>

```
char    *dbf;                    /* Database file descriptor    */
char    record[1000];           /* Record buffer               */
char    fldbuf[128][256];       /* Field buffers to fill       */
```

<output parameters>

none

### RETURN VALUE

The dbscatter() function returns 0 for success. Refer to the section on return codes for a detailed list of return code definitions.

### DESCRIPTION

The dbscatter() function “scatters” the specified record, retrieved with dbgetr()/dbgetrk(), into individual field buffers. The command is similar to the dbrecout() function with the exception that it requires fewer input parameters.

## EXAMPLE

The following example “scatters” the record stored in ‘char record[1000]’ into the specified field buffers (fldbuf[128][256]).

```
#include "dbl.h"

char    *dbf;           /* Database file descriptor */
char    record[1000];   /* Record buffer           */
char    fldbuf[128][256]; /* Field buffers to fill   */

rc = dbscatter(dbf, record, fldbuf);
if ( rc != 0 ) {
    printf("Error scattering record; %d\n", rc);
    exit(1);
} else {
    printf("Record scattered!\n");
}
```

## SEE ALSO

dbfield(), dbgather(), dbgetr(), dbgetrk(), dbrecout()



## NAME

**dbset()**

**issue a Recital Set Command**

## SYNOPSIS

```
#include "dbl.h"
```

```
int    dbset(keyword, value)
```

```
<input parameters>
```

```
char    *keyword;    /* Address of buffer containing SET keyword    */  
int     value;       /* Value to set, 1 for on/true, 0 for off/false    */
```

```
<output parameters>
```

```
none
```

## RETURN VALUE

The dbset() function returns 0 for success. Refer to the section on return codes for a detailed list of return code definitions.

## DESCRIPTION

The dbset() function issues a Recital SET command, switching the specified setting ‘on’ or ‘off’.

Note: The dbset() function currently only operates on the SET EXACT ON/OFF command.

## EXAMPLE

The following example issues a SET EXACT ON. With SET EXACT OFF, strings are compared up to the length of the shorter string. With SET EXACT ON, the strings compared must be an exact match in both characters and length.

```
#include "dbl.h"

int    rc;    /* Return code    */

rc = dbset("EXACT",1);
if ( rc != 0 ) {
    printf("Error setting exact on; %d\n", rc);
    exit(1);
} else {
    printf("Exact set on!\n");
}
```

## SEE ALSO

dbrun()

## dbsetseqno()

---

### NAME

**dbsetseqno()**

**set the sequence number in a table**

### SYNOPSIS

```
#include "dbl.h"

int    dbsetseqno(dbf, seqno)

<input parameter>
char   *dbf;          /* .DBF file descriptor    */
long   seqno;         /* Sequence number to set */

<output parameter>
none
```

### RETURN VALUE

The dbsetseqno() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

The dbsetseqno() function sets the sequence number for a specified table (.DBF).

### EXAMPLE

The following example sets the sequence number, then gets the sequence number.

```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file   */

static void errorproc(
    char    *func,
    char    *str,
    int     rc);

main()
{
    int     rc;           /* Return Code for error handling    */
    char    *dbf;         /* File descriptor for table         */
    long    *seqno;       /* Buffer address for sequence number */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbfilemode(1,0);
    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened shared.", rc);

    rc=dbsetseqno(dbf, (long) 49);
    errorproc("dbsetseqno()", "seqno set to 49.", rc);
    rc=dbgetseqno(dbf, seqno);
    errorproc("dbgetseqno()", "seqno retrieved.", rc);
    printf("\t\t\t Seqno: \t%d\n", *seqno);

    rc = dbclose(dbf);
    errorproc("dbclose()", "table closed.", rc);

    exit(0);
}

static void errorproc(func, str, rc)
char    *func;
char    *str;
int     rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## **SEE ALSO**

`dbgetseqno()`

### NAME

**dbsize()**                                      **get number of records in a data file**

### SYNOPSIS

```
#include "dbl.h"

int    dbsize(dbf, size)

<input parameter>
char   *dbf;           /* .DBF file descriptor          */

<output parameter>
long   size;           /* Address of the variable where the number of records
                        allocated in the .DBF file is returned by the function */
```

### RETURN VALUE

The dbsize() function returns 0 for success, < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

This function returns the number of records currently allocated in the specified .DBF file.

### EXAMPLE

This example obtains the number of records existing in the .DBF file whose file descriptor is in "char \*dbf" and prints the number on the standard output.

```

#include "dbl.h"

char    *dbf;          /* .DBF file descriptor    */
long    db_size;       /* Size of the .DBF file  */
int      rc;           /* Return code            */

rc = dbsize(dbf, &db_size);
if (rc == SUCCESS) {
    printf("the number of records in the database:
           %1d \n", db_size);
} else {
    printf("error number %d \n", rc);
    exit (1);
}

```

## SEE ALSO

dbappend(), dbputr()

## NAME

**dbstrcpy()**

**format string for fields and keys**

## SYNOPSIS

```
#include "dbl.h"
```

```
int    dbstrcpy(string, adjmode, length, origstr)
```

```
<input parameters>
```

```
char    adjmode;    /* Adjustment mode: 'r' or 'R' for right adjusted
                    'l' or 'L' for left adjusted          */
int     length;     /* Length of formatted string          */
char    *origstr;   /* Original string to be formatted     */
```

```
<output parameter>
```

```
char    *string;    /* Address of a buffer where the formatted
                    string is placed by the function      */
```

## RETURN VALUE

This function has no return values.

## DESCRIPTION

This function formats a string according to the specified adjustment mode and length of string. If the "adjmode" input parameter is either 'R' or 'r', the resulting string is right adjusted, if it is either 'L' or 'l', the resulting string is left adjusted. Make sure to allocate the character array "string" of at least size "length" to avoid character string overflow.

NOTE: The formatted string is not NULL terminating!
---



**EXAMPLE**

This example formats a string “John Smith” to a left adjusted 10 character long string.

```
#include "dbl.h"

char    string[10];

dbstrcpy(string, 'L', 10, "John Smith");
```

**SEE ALSO**

dbatofld(), dbatokey(), dbstring()

### NAME

**dbstring()** **format character string into a ‘C’ string**

### SYNOPSIS

```
#include "dbl.h"

int    dbstring(charbuf, length, Cstringbuf)

    <input parameters>
char    *charbuf;    /* Address of a buffer containing string
                      to be formatted */
int     length;      /* Length of formatted string */

    <output parameter>
char    *Cstringbuf; /* Address of a buffer where the formatted
                      string is placed by the function */
```

### RETURN VALUE

This function has no return values.

### DESCRIPTION

This function formats a string in ‘charbuf’ of length ‘length’ into a ‘C’ null terminated string. This function is only of use to 3GL programs written in languages other than ‘C’ which do not support null terminated strings.

## EXAMPLE

This first example formats the string “John Smith”.

```
#include “dbl.h”

char  string[];          /* String buffer      */

dbstring(“John Smith”, 10, string);
```

The second, VMS/COBOL, example formats a string using dbstring() before calling the dbopen() function.

```
IDENTIFICATION DIVISION.
PROGRAM – ID.      COBOL_OPEN.
.
.
.
01  DATABASE_NAME      PIC  X(13).
01  DBF_FD             PIC  9(09) COMP.
01  RETURN_STAT        PIC  S9(04).
.
.
.
START-01
      CALL-DBOPEN.
      CALL “dbstring” USING BY
          VALUE “DATABASE.DBF”
          13
          REFERENCE DATABASE_NAME.
      CALL “dbopen” USING BY
          CONTENT DATABASE_NAME
          REFERENCE DBF_FD
          GIVING RETURN_STAT
.
.
.
```

## SEE ALSO

dbatofld(), dbatokey(), dbstrcpy()

## NAME

**dbtkey()**

**translate key into record number**

## SYNOPSIS

```
#include "dbl.h"

int    dbtkey(ndx, key, recno)

<input parameters>
char   *ndx;           /* .NDX file descriptor          */
char   *key;           /* Address of a buffer containing the key */

<output parameter>
long   *recno          /* Address of the variable where the record
                        number is returned by the function */
```

## RETURN VALUE

The dbtkey() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

This function translates a key into its associated record number by searching for the specified key in the .NDX file and reading the corresponding record number. If the key is numeric and it is not found in the .NDX file, the function returns d\_NOKEY. If the key is not found, but it is of type character, the function keeps searching for similar keys (each time one character shorter than the key in the previous search). If such a key is located, the function returns semi-successful return code d\_MAYBE.

## EXAMPLE

This example translates a character key “char key[10]” to a record number which is returned in “long recno”. The .NDX file descriptor is in “char \*ndx”.

```
#include "dbl.h"

char   *ndx;           /* .NDX file descriptor    */
char   key[10];        /* Key buffer            */
long   recno;          /* Record number         */
int     rc;            /* Return code           */

rc = dbtkey(ndx, key, &recno);
if (rc != SUCCESS && rc != d_MAYBE) {
    printf("Key not found\n");
    return(ERROR);
}
```

## SEE ALSO

dbakey(), dbgetrk(), dbrmvkey()

### NAME

**dbtlockf()**

**test for a lock on a database file**

### SYNOPSIS

```
#include "dbl.h"

int    dbtlock(dbf)

<input parameter>
char   *dbf;          /* File descriptor of a .DBF file to be locked */

<output parameter>
none
```

### RETURN VALUE

The dbtlockf() function returns 0 for success, or < 0 if an error occurs. See the section on return values for a detailed list of return codes.

### DESCRIPTION

The dbtlockf() function tests to see if a database file is already locked by another user.

### EXAMPLE

The following example test locks the .DBF file whose file descriptor is in "char \*dbf".

```
#include "dbl.h"

char    *dbf;          /* Database file descriptor */
int     rc;            /* Return code */

rc = dbtlockf(dbf);
if (rc = SUCCESS) printf("file is locked \n");
else {
    printf("file could not be locked \n");
    return (ERROR);
}
```

## SEE ALSO

dbfilemode(), dblockf(), dblocki(), dblockr(), dbunlockf(), dbunlocki(), dbunlockr()

### NAME

**dbtlockr()**                                      **test for a lock on a record in a database table file**

### SYNOPSIS

```
#include "dbl.h"

int      dbtlockr(dbf, recno)

<input parameter>
char      *dbf;          /* File descriptor of a .DBF file to be locked      */
int      recno;          /* Record number to lock                                */

<output parameter>
none
```

### RETURN VALUE

The dblockr() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

The dblockr() function tests to see if a record is locked by another user.

### EXAMPLE

The following example test locks record number 5 in the .DBF file whose file descriptor is in "char \*dbf".



```
#include "dbl.h"

char    *dbf;          /* Database file descriptor */
int     rc;            /* Return code */

rc = dbtlockr(dbf, 5);
if (rc = SUCCESS) printf("record 5 is locked \n");
else {
    printf("record 5 could not be locked \n");
    return (ERROR);
}
```

## SEE ALSO

dbfilemode(), dblockf(), dblocki(), dblockr(), dbunlockf(), dbunlocki(), dbunlockr()

## **dbunlockf()**

---

### **NAME**

**dbunlockf()**

**unlock a database file**

### **SYNOPSIS**

```
#include "dbl.h"

int      dbunlockf(dbf)

<input parameter>
char    *dbf;          /* File descriptor of a .DBF file to be unlocked    */

<output parameter>
none
```

### **RETURN VALUE**

The dbunlockf() function returns 0 for success, or < 0. See the section on return code values for a detailed list of return codes.

### **DESCRIPTION**

The dbunlockf() function unlocks the database file from the given file descriptor "char \*dbf".

### **EXAMPLE**

The following example unlocks the .DBF file whose file descriptor is in "char \*dbf".

```
#include "dbl.h"

char    *dbf;          /* Database file descriptor */
int     rc;            /* Return code */

rc = dbunlockf(dbf);
if (rc = SUCCESS) printf("file is unlocked \n!");
else {
    printf("error number %d \n", rc);
    return (ERROR);
}
```

## SEE ALSO

dbfilemode(), dblockf(), dblocki(), dblockr(), dbtlockf(), dbtlockr(), dbunlocki(), dbunlockr()

## dbunlocki()

---

### NAME

**dbunlocki()**

**unlock an index file**

### SYNOPSIS

```
#include "dbl.h"

int      dbunlocki(ndx)

<input parameter>
char    *ndx;          /* File descriptor of a .NDX file to be unlocked    */

<output parameter>
none
```

### RETURN VALUE

The dbunlocki() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

The dbunlocki() function unlocks the index file from the given file descriptor "char \*ndx".

### EXAMPLE

The following example unlocks the .NDX file whose file descriptor is in "char \*ndx".

```
#include "dbl.h"

char    *ndx;          /* Index file descriptor    */
int      rc;           /* Return code              */

rc = dbunlock(ndx);
if (rc = SUCCESS) printf("index is unlocked \n");
else {
    printf("error number %d \n", rc);
    return (ERROR);
}
```

## SEE ALSO

dbfilemode(), dblockf(), dblocki(), dblockr(), dbtlockf(), dbtlockr(), dbunlockf(), dbunlockr()

<b>dbunlockr()</b>	<b>unlock a record in a database table file</b>
--------------------	---

```
#include "dbl.h"

int dbunlockr(dbf, recno)

<input parameter>
char *dbf;          /* File descriptor of a .DBF file to be unlocked */
int recno;          /* Record number to unlock */

<output parameter>
none
```

The `dbunlockr()` function returns 0 for success, or  $< 0$  if an error occurs. See the section on return code values for a detailed list of return codes.

The `dbunlockr()` function unlocks a record in the database file from the given file descriptor “char \*dbf”. The record unlocked is the number in “int recno”.

The following example unlocks record number 5 in the .DBF file whose file descriptor is in “char \*dbf”.

```
#include "dbl.h"

char    *dbf;          /* Database file descriptor */
int      rc;            /* Return code */

rc = dbunlockr(dbf, 5);
if (rc = SUCCESS) printf("record 5 in unlocked \n");
else {
    printf("error number %d \n", rc);
    return (ERROR);
}
```

## SEE ALSO

dbfilemode(), dblockf(), dblocki(), dblockr(), dbtlockf(), dbtlockr(), dbunlockf(), dbunlocki()

## NAME

**dbupdm()**

**update memo in a memo file**

## SYNOPSIS

```
#include "dbl.h"

int    dbpudm(dbt, memo, field, oldfield)

<input parameters>
char    *dbt;           /* .DBT file descriptor          */
char    *memo           /* Address of a buffer containing the memo
                        to be put into the file          */
char    *oldfield;      /* Address of old memo field which should
                        be released, 0 if no reclaim of space
                        is to be made                      */

<output parameters>
char    *field;         /* Address in the record buffer where
                        the memo is stored                */
```

## RETURN VALUE

The dbupdm() function returns 0 for success. See the section on return value codes for a detailed list of failure codes.

## DESCRIPTION

This function updates a memo in the specified .dbt file. The memo to be updated must first be acquired by using dbgetr() or by other means. The last parameter of this function controls the reallocation of existing memo entries. If the address is specified, the function adds the existing memo entry to the free block list which controls allocation of space in the .dbt file. The address will usually be the same as the third parameter. If no reallocation is desired, the fourth parameter is null (0). This function does not automatically update the memo pointer in the record. The function dbputr() or others that commit records must be used.



## EXAMPLE

The following example adds more information to the end of an existing memo entry. Note the use of `dbgetr()` and `dbgetm()`. It is assumed that the 4 byte memo pointer starts at the 54<sup>th</sup> array element (`&record[53]`).

```
#include "dbl.h"

int    dbf, dbt;
char   record[1000];
char   status;
int    rc;
char   memobuffer[512];

if(rc = dbgetr, (long) 2, record, &status) !=0) {
    printf("Get Memo Failure: %d\n", rc);
    exit(1);
}

if(rc = dbgetm( dbt, &record[53], memobuffer, 0) !=0) {
    printf("Get Memo Failure: %d\n", rc);
    exit(1);
}

strcat(memobuffer,
        "This string is appended to the end of the retrieved
        memo");

if(rc = dbupdm( dbt, memobuffer,
                &record[53],
                &record[53]) !=0) {
    printf("Memo Update Failure: %d\n",rc);
    exit(1);
}

if(rc = dbupdr(dbf, (long)2, record) !=0) {
    printf("Record Update failure: %d\n", rc);
    exit(1);
}
```

## SEE ALSO

`dbgetm()`, `dbgetr()`, `dbmopen()`, `dbputm()`, `dbputr()`

## NAME

**dbupdr()**

**update record using record number**

## SYNOPSIS

```
#include "dbl.h"
```

```
int      dbupdr(dbf, recno, record)
```

```
<input parameters>
```

```
char    *dbf;           /* .DBF file descriptor      */
long    recno;          /* Record number           */
char *record;           /* Address of the record buffer */
```

```
<output parameters>
```

```
none
```

## RETURN VALUE

The dbupdr() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

This function updates a record in a .DBF file, based on the given record number. The updated record is recognized as active. The corresponding .NDX file (if any) can be updated by dbrmvkey() followed by dbakey().

## EXAMPLE

This example retrieves 20 records from a .DBF file whose file descriptor is in "char \*dbf" and updates them with the contents of the character buffers specified in "char new\_records[20][100]".

```

#include "dbl.h"

char    *dbf;                /* .DBF file descriptor*/
char    new_records[20][100]; /* Contents of updated records*/
char    old_record[100];      /* Record to be updated*/
char    *status;             /* Status of record*/

int      i;                  /* Loop control variable*/
int      rc;                 /* Return code*/

for (i = 1; i <= 20; ++i) {
    rc = dbgetr(dbf, i, record, &status);    /* Get a record      */
    if (rc != SUCCESS) break;
    rc = dbupdr(dbf, i, new_records[i]);     /* Update the record  */
    if (rc != SUCCESS) break;
}

```

## SEE ALSO

dbakey(), dbappend(), dbflush(), dbgetr(), dbputr(), dbrmvkey()

### NAME

**dbxakey()**                                      **add keys to all tags in the tagged index file**

### SYNOPSIS

```
#include "dbl.h"

int      dbxakey(dbx, recno);

<input parameters>
char      *dbx;          /* Tagged index file descriptor      */
long      recno;         /* Record number                    */

<output parameters>
none
```

### RETURN VALUE

The dbxakey() function returns 0 for success, or -1 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

The dbxakey() function adds a key to each index tag in the tagged index file, .DBX for the record number specified.

### EXAMPLE

The following example opens the table and tag index, locks the table and tag index, adds in records, flushes the buffers, then unlocks and closes the files.

```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file */

static  DBFIELD fields[6] = {
    "VESSEL",  'C',   18,   0,   0,
    "LAT",     'N',   4,    1,   0,
    "LONG",    'N',   4,    1,   0,
    "AMOUNT",  'N',  10,    0,   0,
    "DATE",    'D',   8,    0,   0,
    "FLAG",    'C',   9,    0,   0
};

static void errorproc(
    char *func,
    char *str,
    int rc);

static char *text_table[] = {
    "Santa Margarita 80.027.5 7000000095/01/01Spanish ",
    "Santa Rosa      81.824.8 3000000032/01/01Spanish ",
    "Unknown galleon 83.123.0 015/01/01Spanish ",
    "Jessie          97.127.4 10000075/01/01U.S.  ",
    "Lea             96.227.8 100000080/01/01U.S.  ",
    "S.J. Lee        96.926.9 20000075/01/01U.S.  ",
    "Genovase        78.418.4 185000030/01/01Spanish ",
    "Unknown Vessel  77.517.9 075/01/01U.S.  "
};

main()
{
    int rc;          /* Return Code for error handling */
    char *dbf;       /* File descriptor for table      */
    char *dbx;       /* File descriptor for tagged index */
    int i;           /* Loop control variable          */
    long recno;      /* Table record variable          */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbicache(100);
    errorproc("dbicache()", "index cache specified.", rc);
    rc = dbfilemode(1,0);
    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened in shared mode.", rc);

```

```

rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
errorproc("dbxopen()", "tagged index opened.", rc);

for( i = 1; i <= 7; i++) {
    rc = dbappend(dbf, text_table[i], &recno);
    errorproc("dbappend()", "record appended.", rc);
    rc = dbxlocki(dbx);
    errorproc("dbxlocki()", "tagged index file locked.", rc);
    rc = dbxakey(dbx, recno);
    errorproc("dbxakey()", "key added.", rc);
    rc = dbxunlocki(dbx);
    errorproc("dbxunlocki()", "tagged index file unlocked.", rc);
}

rc = dbflush(dbf);
errorproc("dbflush()", "table cache flushed.", rc);
rc = dbxflsh(dbx);
errorproc("dbxflsh()", "Tagged index cache flushed.", rc);

rc = dbxclose(dbx);
errorproc("dbxclose()", "tagged index closed.", rc);
rc = dbclose(dbf);
errorproc("dbclose()", "table closed.", rc);

exit(0);
}

static void errorproc(func, str, rc)
char    *func;
char    *str;
int      rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## SEE ALSO

dbappend(), dbxclose(), dbxflsh(), dbxlocki(), dbxopen(), dbxunlocki()

### NAME

**dbxckey()**

**read current key**

### SYNOPSIS

```
#include "dbl.h"

int    dbxckey(dbx, key, recno)

<input parameters>
char   *dbx;          /* Tagged index file descriptor          */

<output parameters>
char   *key;           /* Address of a buffer where the current key is          */
                        returned                               */
long   *recno;         /* Address of a buffer where the current record number    */
                        associated with the key will be returned */
```

### RETURN VALUE

The `dbxckey()` function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

The `dbxckey()` function reads the current key and its associated record number for the active tag from the specified tagged index file, `.DBX`.

### EXAMPLE

The following example reads the current key and its associated record number from the `VESSEL` tag of the `shipwreck` table.

```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file   */

static  dBFIELD fields[6] = {
    "VESSEL",  'C',   18,   0,   0,
    "LAT",     'N',   4,    1,   0,
    "LONG",    'N',   4,    1,   0,
    "AMOUNT",  'N',  10,    0,   0,
    "DATE",    'D',   8,    0,   0,
    "FLAG",    'C',   9,    0,   0
};

static void errorproc(
    char  *func,
    char  *str,
    int   rc);

main()
{
    int    rc;          /* Return Code for error handling */
    char   *dbf;        /* File descriptor for table      */
    char   *dbx;        /* File descriptor for tagged index */
    char   record[58];  /* Array of char for records      */
    char   status;      /* Deleted record flag            */
    long   recno;       /* Table record variable          */
    char   key[21];     /* Storage location for keys      */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbicache(100);
    errorproc("dbicache()", "index cache specified.", rc);
    rc = dbfilemode(1,0);
    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened in shared mode.", rc);
    rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
    errorproc("dbxopen()", "tagged index opened.", rc);

    rc = dbxtkey(dbx, "Santa Rosa", &recno, 0);
    errorproc("dbxtkey()", "record number found by key.", rc);
    printf("\t\trecord number: \t%d\n", recno);

    rc = dbxgetrk(dbf, dbx, "Santa Rosa", record, &status);
    errorproc("dbxgetrk()", "record retrieved by key.", rc);
    printf("\t\t%s\n", record);

```



```

    rc = dbxckey(dbx, key, &recno);
    errorproc("dbxckey()", "current key read.", rc);
    printf("\t\tcurrent key: \t%s\n", key);

    rc = dbxclose(dbx);
    errorproc("dbxclose()", "tagged index closed.", rc);
    rc = dbfclose(dbf);
    errorproc("dbfclose()", "table closed.", rc);

    exit(0);
}

static void errorproc(func, str, rc)
char    *func;
char    *str;
int     rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## SEE ALSO

dbxclose(), dbxgetrk(), dbxopen(), dbxtkey()

## **dbxclose()**

---

### **NAME**

**dbxclose()**

**close an open tagged index file**

### **SYNOPSIS**

```
#include "dbl.h"

int    dbxclose(dbx);

<input parameters>
char   *dbx;          /* Tagged index file descriptor */

<output parameters>
none
```

### **RETURN VALUE**

The dbxclose() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### **DESCRIPTION**

The dbxclose() function is used to close the tagged index file, .DBX specified in the file descriptor.

### **EXAMPLE**

The following example creates a tag on the VESSEL field for the shipwreck.dbf table, then closes the index and the table.

```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file */

static  dBFIELD fields[6] = {
    "VESSEL",  'C',   18,   0,   0,
    "LAT",     'N',   4,    1,   0,
    "LONG",    'N',   4,    1,   0,
    "AMOUNT",  'N',  10,    0,   0,
    "DATE",    'D',   8,    0,   0,
    "FLAG",    'C',   9,    0,   0
};

static void errorproc(
    char  *func,
    char  *str,
    int    rc);

main()
{
    int    rc;    /* Return Code for error handling */
    char  *dbf;   /* File descriptor for table      */
    char  *dbx;   /* File descriptor for tagged index */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbicache(100);
    errorproc("dbicache()", "index cache specified.", rc);
    rc = dbfilemode(0,0);
    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened in exclusive mode.", rc);

    rc = dbxcreate( dbf,
                    fields[0].fieldnm,      /* "VESSEL", */
                    &dbx,
                    fields[0].fieldnm,      /* "VESSEL", */
                    NULL,
                    0,
                    0);
    errorproc("dbxcreate()", "tag created", rc);

    rc = dbxclose(dbx);
    errorproc("dbxclose()", "tagged index closed.", rc);
    rc = dbclose(dbf);
    errorproc("dbclose()", "table closed.", rc);

```

```

        exit(0);
    }

    static void errorproc(func, str, rc)
    char    *func;
    char    *str;
    int      rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## SEE ALSO

dbxflsh(), dbxopen()

## NAME \_\_\_\_\_

## dbxcreate()

## create an index tag in a tagged index file

## SYNOPSIS

```
#include "dbl.h"
```

```
int dbxcreate(dbf, tagname, dbx, keyexpr, forexpr, keyunique, keydescend)
```

&lt;input parameters&gt;

```

char    *dbf;           /* Table .DBF file descriptor          */
char    *tagname;       /* Name of the tag to be created       */
char    *keyexpr;       /* Address of a buffer containing key expression */
char    *forexpr;       /* Address of a buffer containing for expression */
int      keyunique;      /* If >= 1, tag is created unique      */
int      keydescend;     /* If >= 1, tag is created in descending order */

```

&lt;output parameters&gt;

```
char    *dbx;          /* Open tagged index file descriptor          */
```

## RETURN VALUE

The `dbxcreate()` function returns 0 for success, or  $< 0$  if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

The `dbxcreate()` function will create an index tag for the specified table, `.DBF` in a tagged index file, `.DBX`. If the tagged file does not already exist, it is created with the same name as the table.

### EXAMPLE

The following example creates a tag on the VESSEL field for the shipwreck.dbf table.

```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file */

static  dBFIELD fields[6] = {
    "VESSEL",  'C',   18,   0,   0,
    "LAT",     'N',   4,    1,   0,
    "LONG",    'N',   4,    1,   0,
    "AMOUNT",  'N',  10,    0,   0,
    "DATE",    'D',   8,    0,   0,
    "FLAG",    'C',   9,    0,   0
};

static void errorproc(
    char  *func,
    char  *str,
    int    rc);

main()
{
    int    rc;    /* Return Code for error handling */
    char  *dbf;   /* File descriptor for table      */
    char  *dbx;   /* File descriptor for tagged index */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbicache(100);
    errorproc("dbicache()", "index cache specified.", rc);
    rc = dbfilemode(0,0);
    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened in exclusive mode.", rc);

    rc = dbxcreate( dbf,
                    fields[0].fieldnm,      /* "VESSEL", */
                    &dbx,
                    fields[0].fieldnm,      /* "VESSEL", */
                    NULL,
                    0,
                    0);
    errorproc("dbxcreate()", "tag created", rc);

    rc = dbxclose(dbx);
    errorproc("dbxclose()", "tagged index closed.", rc);
    rc = dbclose(dbf);
    errorproc("dbclose()", "table closed.", rc);

```

```

        exit(0);
    }

    static void errorproc(func, str, rc)
    char    *func;
    char    *str;
    int      rc;
    {
        if ( rc != SUCCESS ) {
            printf("\n Error performing function %s -> %d\n", func, rc);
            exit(1);
        }
        printf("Function: \t%s, \t%s - Ok\n", func, str);
        return;
    }
}

```

## SEE ALSO

dbcreat(), dbcreatx(), dbxkexpr()

## **dbxdroptag()**

---

### **NAME**

**dbxdroptag()**                      **drop the specified tag from the tagged index file**

### **SYNOPSIS**

```
#include "dbl.h"

int    dbxdroptag(dbx, tagname)

<input parameters>
char   *dbx;           /* Tagged index file descriptor      */
char   *tagname;       /* Address of a buffer containing the name
                        of the tag to be dropped          */

<output parameters>
none
```

### **RETURN VALUE**

The dbxdroptag() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### **DESCRIPTION**

The dbxdroptag() function is used to drop tag indexes from a specified tagged index file, .DBX. If the dropped tag is the last tag in the file then the .DBX file is removed, as there must be at least one tag in the file.

### **EXAMPLE**

The following example creates a tag on the FLAG field for the shipwreck.dbf table, then drops the tag.



```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file */

static  DBFIELD fields[6] = {
    "VESSEL", 'C', 18, 0, 0,
    "LAT",    'N', 4, 1, 0,
    "LONG",   'N', 4, 1, 0,
    "AMOUNT", 'N', 10, 0, 0,
    "DATE",   'D', 8, 0, 0,
    "FLAG",   'C', 9, 0, 0
};

static void errorproc(
    char *func,
    char *str,
    int rc);

main()
{
    int rc; /* Return Code for error handling */
    char *dbf; /* File descriptor for table */
    char *dbx; /* File descriptor for tagged index */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbicache(100);
    errorproc("dbicache()", "index cache specified.", rc);
    rc = dbfilemode(0,0);
    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened in exclusive mode.", rc);

    rc = dbxcreate( dbf,
                    fields[5].fieldnm, /* "FLAG", */
                    &dbx,
                    fields[5].fieldnm, /* "FLAG", */
                    NULL,
                    0,
                    0);
    errorproc("dbxcreate()", "tag created", rc);

    rc = dbxdroptag(dbx, fields[5].fieldnm);
    errorproc("dbxdroptag()", "tagged index 'FLAG' dropped.", rc);

    rc = dbxclose(dbx);

```

```

    errorproc("dbxclose()", "tagged index closed.", rc);
    rc = dbclose(dbf);
    errorproc("dbclose()", "table closed.", rc);

    exit(0);
}

static void errorproc(func, str, rc)
char    *func;
char    *str;
int     rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## SEE ALSO

dbxclose(), dbxcreate(), dbxtag()

## NAME

**dbxflsh()** flush I/O buffer cache for tagged index file

## SYNOPSIS

```
#include "dbl.h"

int    dbxflsh(dbx)

<input parameters>
char   *dbx;          /* Tagged index file descriptor */

<output parameters>
none
```

## RETURN VALUE

The dbxflsh() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

The dbxflsh() function forces the I/O buffer cache of the specified .DBX file to be written to disk.

## EXAMPLE

The following example opens the table and tag index, locks the table and tag index, adds in records, flushes the buffers, then unlocks and closes the files.

```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file */

static  DBFIELD fields[6] = {
    "VESSEL",  'C',   18,   0,   0,
    "LAT",     'N',   4,    1,   0,
    "LONG",    'N',   4,    1,   0,
    "AMOUNT",  'N',  10,    0,   0,
    "DATE",    'D',   8,    0,   0,
    "FLAG",    'C',   9,    0,   0
};

static void errorproc(
    char *func,
    char *str,
    int rc);

static char *text_table[] = {
    "Santa Margarita 80.027.5 7000000095/01/01Spanish ",
    "Santa Rosa      81.824.8 3000000032/01/01Spanish ",
    "Unknown galleon 83.123.0 015/01/01Spanish ",
    "Jessie          97.127.4 10000075/01/01U.S.  ",
    "Lea             96.227.8 100000080/01/01U.S.  ",
    "S.J. Lee        96.926.9 20000075/01/01U.S.  ",
    "Genovase        78.418.4 185000030/01/01Spanish ",
    "Unknown Vessel  77.517.9 075/01/01U.S.  "
};

main()
{
    int rc;          /* Return Code for error handling */
    char *dbf;       /* File descriptor for table      */
    char *dbx;       /* File descriptor for tagged index */
    int i;           /* Loop control variable          */
    long recno;      /* Table record variable          */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbicache(100);
    errorproc("dbicache()", "index cache specified.", rc);
    rc = dbfilemode(1,0);
    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened in shared mode.", rc);

```

```

rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
errorproc("dbxopen()", "tagged index opened.", rc);

for( i = 1; i <= 7; i++) {
    rc = dbappend(dbf, text_table[i], &recno);
    errorproc("dbappend()", "record appended.", rc);
    rc = dbxlocki(dbx);
    errorproc("dbxlocki()", "tagged index file locked.", rc);
    rc = dbxakey(dbx, recno);
    errorproc("dbxakey()", "key added.", rc);
    rc = dbxunlocki(dbx);
    errorproc("dbxunlocki()", "tagged index file unlocked.", rc);
}

rc = dbflush(dbf);
errorproc("dbflush()", "table cache flushed.", rc);
rc = dbxflsh(dbx);
errorproc("dbxflsh()", "Tagged index cache flushed.", rc);

rc = dbxclose(dbx);
errorproc("dbxclose()", "tagged index closed.", rc);
rc = dbclose(dbf);
errorproc("dbclose()", "table closed.", rc);

exit(0);
}

static void errorproc(func, str, rc)
char    *func;
char    *str;
int      rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## SEE ALSO

dbappend(), dbxakey(), dbxclose(), dbxlocki(), dbxopen(), dbxunlocki()

<b>dbxfwd()</b>	<b>position access pointer to end of active tag</b>
-----------------	---

```
#include "dbl.h"

int      dbxfwd(dbx)

<input parameters>
char     *dbx;          /* Tagged index file descriptor      */

<output parameters>
none
```

The `dbxfwd()` function returns to 0 for success, or  $< 0$  if an error occurs. See the section on return code values for a detailed list of return codes.

The `dbxfwd()` function positions the access pointer at the end of the active tag in the specified tagged index file, `.DBX`.

The following example opens the table and tag index, goes to the bottom of the index, reads the previous key by position then by key, rewinds the index, reads the next 6 records in the index, reads the next record by key and then closes the files.

```
#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file */
#include "dblproto.h"     /* Recital/Library prototype file */

static  dbFIELD fields[6] = {
    "VESSEL",  'C',    18,    0,    0,
    "LAT",     'N',     4,     1,    0,
    "LONG",    'N',     4,     1,    0,
    "AMOUNT",  'N',    10,     0,    0,
    "DATE",    'D',     8,     0,    0,
    "FLAG",    'C',     9,     0,    0
};

static void errorproc(
    char    *func,
    char    *str,
    int     rc);

main()
{
int     rc;                /* Return Code for error handling */
char    *dbf;             /* File descriptor for table */
char    *dbx;             /* File descriptor for tagged index */
int     i;               /* Loop control variable */
char    record[58];       /* Array of char for records */
char    nextkey[10][20];  /* Key buffers */
char    status;           /* Deleted record flag */
long    rec_no[10];       /* Record number array */

rc = dbdcache(100);
errorproc("dbdcache()", "table cache specified.", rc);
rc = dbicache(100);
errorproc("dbicache()", "index cache specified.", rc);
rc = dbfilemode(1,0);
rc = dbopen("shipwreck.dbf", &dbf);
errorproc("dbopen()", "table opened in shared mode.", rc);
rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
errorproc("dbxopen()", "tagged index opened.", rc);

rc = dbxfwd(dbx);
errorproc("dbxfwd()", "index pointer placed at bottom.", rc);

rc = dbxpkey(dbx, nextkey[0], rec_no);
errorproc("dbxpkey()", "previous key found in tagged index.", rc);
printf("\t\tprev key: \t%s\n", nextkey[0]);
```





## get next data record by key in active tag

## SYNOPSIS

```
char    *status;      /* Record status, if marked for deletion = 0,
                        otherwise 1 */
```

## RETURN VALUE

## DESCRIPTION

The `dbxgetnr()` reads the next record in descending order in the active tag and returns the record data.

### EXAMPLE

173

```
#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file */
#include "dblproto.h"     /* Recital/Library prototype file */

static  dbFIELD fields[6] = {
    "VESSEL",  'C',   18,   0,   0,
    "LAT",     'N',    4,   1,   0,
    "LONG",    'N',    4,   1,   0,
    "AMOUNT",  'N',   10,   0,   0,
    "DATE",    'D',    8,   0,   0,
    "FLAG",    'C',    9,   0,   0
};

static void errorproc(
    char    *func,
    char    *str,
    int     rc);

main()
{
int     rc;                /* Return Code for error handling */
char    *dbf;             /* File descriptor for table */
char    *dbx;             /* File descriptor for tagged index */
int     i;               /* Loop control variable */
char    record[58];       /* Array of char for records */
char    nextkey[10][20];  /* Key buffers */
char    status;           /* Deleted record flag */
long    rec_no[10];       /* Record number array */

rc = dbdcache(100);
errorproc("dbdcache()", "table cache specified.", rc);
rc = dbicache(100);
errorproc("dbicache()", "index cache specified.", rc);
rc = dbfilemode(1,0);
rc = dbopen("shipwreck.dbf", &dbf);
errorproc("dbopen()", "table opened in shared mode.", rc);
rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
errorproc("dbxopen()", "tagged index opened.", rc);

rc = dbxfwd(dbx);
errorproc("dbxfwd()", "index pointer placed at bottom.", rc);

rc = dbxpkey(dbx, nextkey[0], rec_no);
errorproc("dbxpkey()", "previous key found in tagged index.", rc);
printf("\t\tprev key: \t%s\n", nextkey[0]);
```



## NAME

**dbxgetpr()**                                      **get previous data record by key in active tag**

## SYNOPSIS

```
#include "dbl.h"

int      dbxgetpr(dbf, dbx, record, status)

<input parameters>
char      *dbf;           /* Table file descriptor          */
char      *dbx;           /* Tagged index file descriptor   */

<output parameters>
char      *record;        /* Address of a buffer where the record data
                           is returned                      */
char      *status;        /* Record status, if marked for deletion = 0,
                           otherwise 1                      */
```

## RETURN VALUE

The dbxgetpr() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

The dbxgetpr() reads the previous record in descending order in the active tag and returns the record data.

## EXAMPLE

The following example opens the table and tag index, goes to the bottom of the index, reads the previous key by position then by key, rewinds the index, reads the next 6 records in the index, reads the next record by key and then closes the files.

```
#include <stdio.h>                                /* Recital/Library include file */
#include "dblproto.h"                             /* Recital/Library prototype file */

static  dbFIELD fields[6] = {
    "VESSEL",   'C',   18,   0,   0,
    "LAT",      'N',    4,   1,   0,
    "LONG",     'N',    4,   1,   0,
    "AMOUNT",   'N',   10,   0,   0,
    "DATE",     'D',    8,   0,   0,
    "FLAG",     'C',    9,   0,   0
};

static void errorproc(
    char    *func,
    char    *str,
    int     rc);

main()
{
int     rc;                                /* Return Code for error handling */
char    *dbf;                             /* File descriptor for table */
char    *dbx;                             /* File descriptor for tagged index */
int     i;                                /* Loop control variable */
char    record[58];                       /* Array of char for records */
char    nextkey[10][20];                  /* Key buffers */
char    status;                           /* Deleted record flag */
long    rec_no[10];                       /* Record number array */

rc = dbdcache(100);
errorproc("dbdcache()", "table cache specified.", rc);
rc = dbicache(100);
errorproc("dbicache()", "index cache specified.", rc);
rc = dbfilemode(1,0);
rc = dbopen("shipwreck.dbf", &dbf);
errorproc("dbopen()", "table opened in shared mode.", rc);
rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
errorproc("dbxopen()", "tagged index opened.", rc);

rc = dbxfwd(dbx);
errorproc("dbxfwd()", "index pointer placed at bottom.", rc);

rc = dbxpkey(dbx, nextkey[0], rec_no);
errorproc("dbxpkey()", "previous key found in tagged index.", rc);
printf("\t\tprev key: \t%s\n", nextkey[0]);
```



### NAME

**dbxgetrk()**

**get data record by key in tag**

### SYNOPSIS

```
#include "dbl.h"
```

```
int    dbxgetrk(dbf, dbx, key, record, status)
```

```
<input parameters>
```

```
char    *dbf;           /* Table file descriptor          */
char    *dbx;           /* Tagged index file descriptor   */
char    *key;           /* Address of the buffer containing a key to
                        search for                                */
```

```
<output parameters>
```

```
char    *record;        /* Address of a buffer where the record data
                        is returned                                */
char    *status;        /* Record status, if marked for deletion = 0,
                        otherwise 1                                */
```

### RETURN VALUE

The `dbxgetrk()` function returns 0 for success, or < 0 if an error occurs. See the section on return code vales for a detailed list of return codes.

### DESCRIPTION

The `dbxgetrk()` function searches for the specified key in the active tag of the .DBX file. If found, the record data is returned.

### EXAMPLE

The following example searches for “Santa Rosa” in the VESSEL tag of the shipwreck table and returns the record.

```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file   */

static  dbFIELD fields[6] = {
    "VESSEL",  'C',   18,   0,   0,
    "LAT",     'N',    4,   1,   0,
    "LONG",    'N',    4,   1,   0,
    "AMOUNT",  'N',   10,   0,   0,
    "DATE",    'D',    8,   0,   0,
    "FLAG",    'C',    9,   0,   0
};

static void errorproc(
    char  *func,
    char  *str,
    int    rc);

main()
{
    int    rc;          /* Return Code for error handling */
    char  *dbf;         /* File descriptor for table      */
    char  *dbx;         /* File descriptor for tagged index */
    char  record[58];    /* Array of char for records      */
    char  status;        /* Deleted record flag            */
    long  recno;         /* Table record variable          */
    char  key[21];       /* Storage location for keys      */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbicache(100);
    errorproc("dbicache()", "index cache specified.", rc);
    rc = dbfilemode(1,0);
    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened in shared mode.", rc);
    rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
    errorproc("dbxopen()", "tagged index opened.", rc);

    rc = dbxtkey(dbx, "Santa Rosa", &recno, 0);
    errorproc("dbxtkey()", "record number found by key.", rc);
    printf("\t\trecord number: \t%d\n", recno);

    rc = dbxgetrk(dbf, dbx, "Santa Rosa", record, &status);
    errorproc("dbxgetrk()", "record retrieved by key.", rc);
    printf("\t\t%s\n", record);
}

```



```

    rc = dbxckey(dbx, key, &recno);
    errorproc("dbxckey()", "current key read.", rc);
    printf("\t\tcurrent key: \t%s\n", key);

    rc = dbxclose(dbx);
    errorproc("dbxclose()", "tagged index closed.", rc);
    rc = dbfclose(dbf);
    errorproc("dbfclose()", "table closed.", rc);

    exit(0);
}

static void errorproc(func, str, rc)
char    *func;
char    *str;
int      rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## SEE ALSO

dbxckey(), dbxclose(), dbxopen(), dbxtkey()

## NAME

**dbxgoto()**                    go to specified record number and reposition keys in index tag

## SYNOPSIS

```
#include "dbl.h"

int    dbxgoto(dbx, recno)

<input parameters>
char   *dbx;           /* Tagged index file descriptor   */
long   recno;          /* Record number to go to       */

<output parameters>
none
```

## RETURN VALUE

The dbxgoto() function returns for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

The dbxgoto() function will go to the specified record number in the table associated with the tagged file. The record number specified becomes the current record and the index tag is repositioned.

## EXAMPLE

The following example checks how many records are in the table, then goes to the last physical record in the table.

```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file   */

static  DBFIELD fields[6] = {
    "VESSEL",  'C',   18,   0,   0,
    "LAT",     'N',   4,    1,   0,
    "LONG",    'N',   4,    1,   0,
    "AMOUNT",  'N',  10,   0,   0,
    "DATE",    'D',   8,    0,   0,
    "FLAG",    'C',   9,    0,   0
};

static void errorproc(
    char  *func,
    char  *str,
    int   rc);

main()
{
    int    rc;          /* Return Code for error handling */
    char   *dbf;         /* File descriptor for table      */
    char   *dbx;         /* File descriptor for tagged index */
    long   db_size;      /* Size of table variable         */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbicache(100);
    errorproc("dbicache()", "index cache specified.", rc);
    rc = dbfilemode(1,0);
    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened in shared mode.", rc);

    rc = dbsize(dbf, &db_size);
    errorproc("dbsize()", "size retrieved.", rc);
    printf("\t\t No. of Records: \t%d\n", db_size);

    rc = dbxgoto(dbx, db_size);
    errorproc("dbxgoto()", "gone to last physical record in table.", rc);

    rc = dbxclose(dbx);
    errorproc("dbxclose()", "tagged index closed.", rc);
    rc = dbclose(dbf);
    errorproc("dbclose()", "table closed.", rc);
}

```

```
        exit(0);
    }

    static void errorproc(func, str, rc)
    char    *func;
    char    *str;
    int      rc;
    {
        if ( rc != SUCCESS ) {
            printf("\n Error performing function %s -> %d\n", func, rc);
            exit(1);
        }
        printf("Function: \t%s, \t%s - Ok\n", func, str);
        return;
    }
}
```

## **SEE ALSO**

dbxfwd(), dbxrewind()

## NAME

**dbxkexpr()** **get the key expression for a specified tag**

## SYNOPSIS

```
#include "dbl.h"

int    dbxkexpr(dbx, tagno, tagname, keytype, keyexpr, forexpr,
               keyexprlen, keylen, keyunique, keydescend);

<input parameters>
char   *dbx;           /* Tagged index file descriptor          */

<input/output parameters>
int     tagno;          /* If tagno > 0, then it is used as input as the tag number,
                        otherwise the tag number corresponding to the
                        specified tagname is returned.          */
char   *tagname;       /* If tagno <= 0, then it is used as input as the tagname,
                        otherwise the tagname corresponding to the
                        specific tagno is returned.            */

<output parameters>
char   keytype;        /* Tag key data type:
                        'C' - Character
                        'N' - Numeric
                        'D' - Date                               */
char   *keyexpr;       /* Tag key expression                               */
char   *forexpr;       /* Tag for expression                               */
int     *keyexprlen;   /* Length of the tag key expression                 */
int     *keylen;       /* Tag key length                                   */
int     *keyunique;    /* 1 if tag is unique, 0 otherwise                   */
int     *keydescend;   /* 1 if tag is in descending order, 0 otherwise      */
```

## RETURN VALUE

The `dbxkexpr()` function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

The `dbxkexpr()` function obtains the index tag information from the specified tagged index file, `.DBX` for a tag number or name.

## EXAMPLE

The following example opens the index for the `shipwreck.dbf` table, checks the number of tags, then returns detailed information about the tags

```
#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file */

static  DBFIELD fields[6] = {
    "VESSEL", 'C', 18, 0, 0,
    "LAT",    'N', 4, 1, 0,
    "LONG",   'N', 4, 1, 0,
    "AMOUNT", 'N', 10, 0, 0,
    "DATE",   'D', 8, 0, 0,
    "FLAG",   'C', 9, 0, 0
};

static void errorproc(
    char *func,
    char *str,
    int rc);

main()
{
    int rc;          /* Return Code for error handling */
    char *dbf;       /* File descriptor for table      */
    char *dbx;       /* File descriptor for tagged index */
    char keytype;     /* Variable for varying keytype  */
    char tagname[21]; /* Storage location for tag name  */
    char keyexpr[512]; /* Storage location for key expr  */
    char forexpr[512]; /* Storage location for for expr  */
    int kexprlen;     /* The key expression length      */
    int keylen;       /* The key length                 */
    int i;            /* Loop control variable          */
    int keyunique;     /* Boolean for unqiue tag key     */
    int keydescend;    /* Boolean for descending tag key */
    int notags;       /* Number of tags in a dbx file   */
```

```

rc = dbdcache(100);
errorproc("dbdcache()", "table cache specified.", rc);
rc = dbicache(100);
errorproc("dbicache()", "index cache specified.", rc);
rc = dbfilemode(0,0);
rc = dbopen("shipwreck.dbf", &dbf);
errorproc("dbopen()", "table opened in exclusive mode.", rc);
rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
errorproc("dbxopen()", "tagged index opened.", rc);
rc = dbxnotags(dbx, &notags);
errorproc("dbxnotags()", "tagged index count obtained.", rc);
printf("\t\t\t\t\t Key count: \t%d\n", notags );

for (i = 1; i <= notags; i++) {
rc = dbxkexpr(dbx,
               &i,
               tagname,
               &keytype,
               keyexpr,
               forexpr,
               &kexprlen,
               &keylen,
               &keyunique,
               &keydescend);
errorproc("dbxkexpr()", "key information retrieved.", rc);
printf("\t\t\t\t\t Key number: \t%d\n", i );
printf("\t\t\t\t\t Key name: \t%s\n", tagname );
printf("\t\t\t\t\t Key type: \t%c\n", keytype );
printf("\t\t\t\t\t Key expression: %s\n", keyexpr);
printf("\t\t\t\t\t Key length: %d\t Expression length: %d\n", kexprlen, keylen);
printf("\t\t\t\t\t For expression: %s\n", forexpr);
printf("\t\t\t\t\t Key unique: \t%s\n", keyunique ? "Yes" : "No" );
printf("\t\t\t\t\t Key descending: \t%s\n", keydescend ? "Yes" : "No" );
}

rc = dbxclose(dbx);
errorproc("dbxclose()", "tagged index closed.", rc);
rc = dbclose(dbf);
errorproc("dbclose()", "table closed.", rc);
exit(0);
}

static void errorproc(func, str, rc)
char *func;
char *str;
int rc;

```

```
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}
```

## **SEE ALSO**

dbxclose(), dbxcreate(), dbxopen()



### NAME

**dbxlocki()**

**lock tagged index file**

### SYNOPSIS

```
#include "dbl.h"

int    dbxlocki(dbx)

<input parameters>
char   *dbx;          /* Tagged index file pointer */

<output parameters>
none
```

### RETURN VALUE

The dbxlocki() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

The dbxlocki() function locks the specified tagged index file, .DBX.

### EXAMPLE

The following example opens the table and tag index, locks the table and tag index, adds in records, flushes the buffers, then unlocks and closes the files.

```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file   */

static  DBFIELD fields[6] = {
    "VESSEL",  'C',   18,   0,   0,
    "LAT",     'N',   4,    1,   0,
    "LONG",    'N',   4,    1,   0,
    "AMOUNT",  'N',  10,    0,   0,
    "DATE",    'D',   8,    0,   0,
    "FLAG",    'C',   9,    0,   0
};

static void errorproc(
    char *func,
    char *str,
    int rc);

static char *text_table[] = {
    "Santa Margarita 80.027.5 700000095/01/01Spanish ",
    "Santa Rosa 81.824.8 3000000032/01/01Spanish ",
    "Unknown galleon 83.123.0 015/01/01Spanish ",
    "Jessie 97.127.4 10000075/01/01U.S. ",
    "Lea 96.227.8 100000080/01/01U.S. ",
    "S.J. Lee 96.926.9 20000075/01/01U.S. ",
    "Genovase 78.418.4 185000030/01/01Spanish ",
    "Unknown Vessel 77.517.9 075/01/01U.S. "
};

main()
{
    int rc;          /* Return Code for error handling */
    char *dbf;       /* File descriptor for table      */
    char *dbx;       /* File descriptor for tagged index */
    int i;           /* Loop control variable          */
    long recno;      /* Table record variable          */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbicache(100);
    errorproc("dbicache()", "index cache specified.", rc);
    rc = dbfilemode(1,0);
    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened in shared mode.", rc);

```

```

rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
errorproc("dbxopen()", "tagged index opened.", rc);

for( i = 1; i <= 7; i++) {
    rc = dbappend(dbf, text_table[i], &recno);
    errorproc("dbappend()", "record appended.", rc);
    rc = dbxlocki(dbx);
    errorproc("dbxlocki()", "tagged index file locked.", rc);
    rc = dbxakey(dbx, recno);
    errorproc("dbxakey()", "key added.", rc);
    rc = dbxunlocki(dbx);
    errorproc("dbxunlocki()", "tagged index file unlocked.", rc);
}

rc = dbflush(dbf);
errorproc("dbflush()", "table cache flushed.", rc);
rc = dbxflsh(dbx);
errorproc("dbxflsh()", "Tagged index cache flushed.", rc);

rc = dbxclose(dbx);
errorproc("dbxclose()", "tagged index closed.", rc);
rc = dbclose(dbf);
errorproc("dbclose()", "table closed.", rc);

exit(0);
}

static void errorproc(func, str, rc)
char    *func;
char    *str;
int      rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## SEE ALSO

dbappend(), dbxakey(), dbxclose(), dbxflsh(), dbxopen(), dbxunlocki()

## NAME

**dbxnkey()**

**read next key in active tag**

## SYNOPSIS

```
#include "dbl.h"

int    dbxnkey(dbx, key, recno)

<input parameters>
char   *dbx;          /* Tagged index file descriptor    */

<output parameters>
char   *key;           /* Address of a buffer where the key
                        is returned                      */
long   *recno;         /* Address of a buffer where the record
                        number is returned                 */
```

## RETURN VALUE

The `dbxnkey()` function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

The `dbxnkey()` function reads the next key in ascending order from the active tag in the .DBX file. The key and record number for the next key is returned.

## EXAMPLE

The following example opens the table and tag index, goes to the bottom of the index, reads the previous key by position then by key, rewinds the index, reads the next 6 records in the index, reads the next record by key and then closes the files.

```
#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file */
#include "dblproto.h"     /* Recital/Library prototype file */

static  dbFIELD fields[6] = {
    "VESSEL",  'C',   18,   0,   0,
    "LAT",     'N',    4,   1,   0,
    "LONG",    'N',    4,   1,   0,
    "AMOUNT",  'N',   10,   0,   0,
    "DATE",    'D',    8,   0,   0,
    "FLAG",    'C',    9,   0,   0
};

static void errorproc(
    char    *func,
    char    *str,
    int     rc);

main()
{
int     rc;                /* Return Code for error handling */
char    *dbf;             /* File descriptor for table */
char    *dbx;             /* File descriptor for tagged index */
int     i;               /* Loop control variable */
char    record[58];       /* Array of char for records */
char    nextkey[10][20];  /* Key buffers */
char    status;           /* Deleted record flag */
long    rec_no[10];       /* Record number array */

rc = dbdcache(100);
errorproc("dbdcache()", "table cache specified.", rc);
rc = dbicache(100);
errorproc("dbicache()", "index cache specified.", rc);
rc = dbfilemode(1,0);
rc = dbopen("shipwreck.dbf", &dbf);
errorproc("dbopen()", "table opened in shared mode.", rc);
rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
errorproc("dbxopen()", "tagged index opened.", rc);

rc = dbxfwd(dbx);
errorproc("dbxfwd()", "index pointer placed at bottom.", rc);

rc = dbxpkey(dbx, nextkey[0], rec_no);
errorproc("dbxpkey()", "previous key found in tagged index.", rc);
printf("\t\tprev key: \t%s\n", nextkey[0]);
```

```

    rc = dbxgetpr(dbf, dbx, record, &status);
    errorproc("dbxgetpr()", "previous record (by key) retrieved.", rc);
    printf("Previous: \t%s\n", record);

    rc = dbxrewind(dbx);
    errorproc("dbxrewind()", "Tagged index rewound.", rc);

    for( i = 1; i <= 6; i++) {
        rc = dbxgetnr(dbf, dbx, record, &status);
        printf("\t\t%d: %s\n", i, record);
    }
    errorproc("dbxgetnr()", "6 records retrieved by tag key.", rc);

    rc = dbxnkey(dbx, nextkey[0], rec_no);
    errorproc("dbxnkey()", "next key retrieved in tagged index.", rc);
    printf("\t\tnext key: \t%s\n", nextkey[0]);

    rc = dbxclose(dbx);
    errorproc("dbxclose()", "tagged index closed.", rc);
    rc = dbclose(dbf);
    errorproc("dbclose()", "table closed.", rc);

    exit(0);
}

static void errorproc(func, str, rc)
char    *func;
char    *str;
int     rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## SEE ALSO

dbxfwd(), dbxgetnr(), dbxgetpr(), dbxpkey(), dbxrewind()

### NAME

**dbxnotags()**                      **return the number of index tags in a tagged index file**

### SYNOPSIS

```
#include "dbl.h"

int    dbxnotags(dbx, notags)

<input parameters>
char   *dbx;           /* Tagged index file descriptor   */

<output parameters>
int     *notags;        /* Number of tags in the .DBX file   */
```

### RETURN VALUE

The dbxnotags() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

The dbxnotags() function is used to find out the number of tag in the specified index. There must be at least one tag in the index file, up to a maximum of 128.

### EXAMPLE

The following example checks the number of tags on the shipwreck.dbf table, then returns detailed information about the tags

```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file   */

static  DBFIELD fields[6] = {
    "VESSEL",  'C',  18,  0,  0,
    "LAT",     'N',   4,  1,  0,
    "LONG",    'N',   4,  1,  0,
    "AMOUNT",  'N',  10,  0,  0,
    "DATE",    'D',   8,  0,  0,
    "FLAG",    'C',   9,  0,  0
};

static void errorproc(
    char  *func,
    char  *str,
    int    rc);

main()
{
    int    rc;                /* Return Code for error handling */
    char  *dbf;              /* File descriptor for table      */
    char  *dbx;              /* File descriptor for tagged index */
    char  keytype;            /* Variable for varying keytype   */
    char  tagname[21];        /* Storage location for tag name   */
    char  keyexpr[512];       /* Storage location for key expr   */
    char  forexpr[512];       /* Storage location for for expr   */
    int    kexprlen;          /* The key expression length       */
    int    keylen;            /* The key length                  */
    int    i;                 /* Loop control variable          */
    int    keyunique;         /* Boolean for unique tag key      */
    int    keydescend;        /* Boolean for descending tag key  */
    int    notags;            /* Number of tags in a dbx file    */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbicache(100);
    errorproc("dbicache()", "index cache specified.", rc);
    rc = dbfilemode(0,0);
    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened in exclusive mode.", rc);
    rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
    errorproc("dbxopen()", "tagged index opened.", rc);
    rc = dbxnotags(dbx, &notags);
    errorproc("dbxnotags()", "tagged index count obtained.", rc);

```





**SEE ALSO**

`dbxclose()`, `dbxcreate()`, `dbxkexpr()`, `dbxopen()`

## dbxopen()

---

### NAME

**dbxopen()**

**open a .dbx index file**

### SYNOPSIS

```
#include "dbl.h"

int    dbxopen(dbf, filename, dbx);

<input parameters>
char   *dbf;           /* .DBF file descriptor          */
char   *filename;      /* Address of a buffer containing the name of
                        a .DBX file to be opened          */

<output parameters>
char   *dbx;           /* The .DBX file descriptor      */
```

### RETURN VALUE

The dbxopen() function returns 0 for success , or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

This function opens a .DBX file and returns its file descriptor. A call to the dbxclose() function releases the file descriptor.

### EXAMPLE

The following example opens the index for the shipwreck.dbf table, checks the number of tags, then returns detailed information about the tags

```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file   */

static  DBFIELD fields[6] = {
    "VESSEL",  'C',  18,  0,  0,
    "LAT",     'N',  4,  1,  0,
    "LONG",    'N',  4,  1,  0,
    "AMOUNT",  'N', 10,  0,  0,
    "DATE",    'D',  8,  0,  0,
    "FLAG",    'C',  9,  0,  0
};

static void errorproc(
    char *func,
    char *str,
    int rc);

main()
{
    int rc;          /* Return Code for error handling */
    char *dbf;       /* File descriptor for table      */
    char *dbx;       /* File descriptor for tagged index */
    char keytype;     /* Variable for varying keytype   */
    char tagname[21]; /* Storage location for tag name   */
    char keyexpr[512]; /* Storage location for key expr   */
    char forexpr[512]; /* Storage location for for expr   */
    int kexprlen;     /* The key expression length       */
    int keylen;       /* The key length                  */
    int i;            /* Loop control variable           */
    int keyunique;     /* Boolean for unique tag key      */
    int keydescend;    /* Boolean for descending tag key  */
    int notags;        /* Number of tags in a dbx file    */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbicache(100);
    errorproc("dbicache()", "index cache specified.", rc);
    rc = dbfilemode(0,0);
    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened in exclusive mode.", rc);
    rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
    errorproc("dbxopen()", "tagged index opened.", rc);
    rc = dbxnotags(dbx, &notags);
    errorproc("dbxnotags()", "tagged index count obtained.", rc);

```



## **SEE ALSO**

`dbxclose()`, `dbxcreate()`, `dbxkexpr()`

## NAME

**dbxpkey()**

**read previous key in active tag**

## SYNOPSIS

```
#include "dbl.h"

int    dbxpkey(dbx, key, recno)

<input parameters>
char   *dbx;          /* Tagged index file descriptor    */

<output parameters>
char   *key;          /* Address of a buffer where the key
                        is returned                               */
long   *recno ;       /* Address of a buffer where the record number
                        is returned                               */
```

## RETURN VALUE

The dbxpkey() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

The dbxpkey() function locates the key in descending order in the active tag for the .DBX file. The key and record number is returned.

## EXAMPLE

The following example opens the table and tag index, goes to the bottom of the index, reads the previous key by position then by key, rewinds the index, reads the next 6 records in the index, reads the next record by key and then closes the files.

```
#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file */
#include "dblproto.h"     /* Recital/Library prototype file */

static  dbFIELD fields[6] = {
    "VESSEL",  'C',   18,   0,   0,
    "LAT",     'N',    4,   1,   0,
    "LONG",    'N',    4,   1,   0,
    "AMOUNT",  'N',   10,   0,   0,
    "DATE",    'D',    8,   0,   0,
    "FLAG",    'C',    9,   0,   0
};

static void errorproc(
    char    *func,
    char    *str,
    int     rc);

main()
{
int     rc;                /* Return Code for error handling */
char    *dbf;             /* File descriptor for table */
char    *dbx;             /* File descriptor for tagged index */
int     i;                /* Loop control variable */
char    record[58];       /* Array of char for records */
char    nextkey[10][20];  /* Key buffers */
char    status;           /* Deleted record flag */
long    rec_no[10];       /* Record number array */

rc = dbdcache(100);
errorproc("dbdcache()", "table cache specified.", rc);
rc = dbicache(100);
errorproc("dbicache()", "index cache specified.", rc);
rc = dbfilemode(1,0);
rc = dbopen("shipwreck.dbf", &dbf);
errorproc("dbopen()", "table opened in shared mode.", rc);
rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
errorproc("dbxopen()", "tagged index opened.", rc);

rc = dbxfwd(dbx);
errorproc("dbxfwd()", "index pointer placed at bottom.", rc);

rc = dbxpkey(dbx, nextkey[0], rec_no);
errorproc("dbxpkey()", "previous key found in tagged index.", rc);
printf("\t\tprev key: \t%s\n", nextkey[0]);
```



```

    rc = dbxgetpr(dbf, dbx, record, &status);
    errorproc("dbxgetpr()", "previous record (by key) retrieved.", rc);
    printf("Previous: \t%s\n", record);

    rc = dbxrewind(dbx);
    errorproc("dbxrewind()", "Tagged index rewound.", rc);

    for( i = 1; i <= 6; i++) {
        rc = dbxgetnr(dbf, dbx, record, &status);
        printf("\t\t%d: %s\n", i, record);
    }
    errorproc("dbxgetnr()", "6 records retrieved by tag key.", rc);

    rc = dbxnkey(dbx, nextkey[0], rec_no);
    errorproc("dbxnkey()", "next key retrieved in tagged index.", rc);
    printf("\t\tnext key: \t%s\n", nextkey[0]);

    rc = dbxclose(dbx);
    errorproc("dbxclose()", "tagged index closed.", rc);
    rc = dbclose(dbf);
    errorproc("dbclose()", "table closed.", rc);

    exit(0);
}

static void errorproc(func, str, rc)
char    *func;
char    *str;
int     rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## SEE ALSO

dbxfwd(), dbxgetnr(), dbxgetpr(), dbxnkey(), dbxrewind()

---

<b>dbxrewind()</b>	<b>position</b> access pointer to beginning of tag
--------------------	--

```
#include "dbl.h"

int dbxrewind(dbx)

<input parameters>
char *dbx;          /* Tagged index file descriptor */

<output parameters>
none
```

The `dbxrewind()` function returns 0 for success, or  $< 0$  if an error occurs. See the section on return code values for a detailed list of return codes.

The `dbxrewind()` function positions the access pointer at the beginning of the active tag in the specified tagged index file, `.DBX`.

The following example opens the table and tag index, goes to the bottom of the index, reads the previous key by position then by key, rewinds the index, reads the next 6 records in the index, reads the next record by key and then closes the files.

```
#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file */
#include "dblproto.h"     /* Recital/Library prototype file */

static  dbFIELD fields[6] = {
    "VESSEL",  'C',   18,   0,   0,
    "LAT",     'N',    4,   1,   0,
    "LONG",    'N',    4,   1,   0,
    "AMOUNT",  'N',   10,   0,   0,
    "DATE",    'D',    8,   0,   0,
    "FLAG",    'C',    9,   0,   0
};

static void errorproc(
    char    *func,
    char    *str,
    int     rc);

main()
{
int     rc;                /* Return Code for error handling */
char    *dbf;             /* File descriptor for table */
char    *dbx;             /* File descriptor for tagged index */
int     i;               /* Loop control variable */
char    record[58];       /* Array of char for records */
char    nextkey[10][20];  /* Key buffers */
char    status;           /* Deleted record flag */
long    rec_no[10];       /* Record number array */

rc = dbdcache(100);
errorproc("dbdcache()", "table cache specified.", rc);
rc = dbicache(100);
errorproc("dbicache()", "index cache specified.", rc);
rc = dbfilemode(1,0);
rc = dbopen("shipwreck.dbf", &dbf);
errorproc("dbopen()", "table opened in shared mode.", rc);
rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
errorproc("dbxopen()", "tagged index opened.", rc);

rc = dbxfwd(dbx);
errorproc("dbxfwd()", "index pointer placed at bottom.", rc);

rc = dbxpkey(dbx, nextkey[0], rec_no);
errorproc("dbxpkey()", "previous key found in tagged index.", rc);
printf("\t\tprev key: \t%s\n", nextkey[0]);
```

```

    rc = dbxgetpr(dbf, dbx, record, &status);
    errorproc("dbxgetpr()", "previous record (by key) retrieved.", rc);
    printf("Previous: \t%s\n", record);

    rc = dbxrewind(dbx);
    errorproc("dbxrewind()", "Tagged index rewound.", rc);

    for( i = 1; i <= 6; i++) {
        rc = dbxgetnr(dbf, dbx, record, &status);
        printf("\t\t%d: %s\n", i, record);
    }
    errorproc("dbxgetnr()", "6 records retrieved by tag key.", rc);

    rc = dbxnkey(dbx, nextkey[0], rec_no);
    errorproc("dbxnkey()", "next key retrieved in tagged index.", rc);
    printf("\t\tnext key: \t%s\n", nextkey[0]);

    rc = dbxclose(dbx);
    errorproc("dbxclose()", "tagged index closed.", rc);
    rc = dbclose(dbf);
    errorproc("dbclose()", "table closed.", rc);

    exit(0);
}

static void errorproc(func, str, rc)
char    *func;
char    *str;
int     rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## SEE ALSO

dbxfwd(), dbxgetnr(), dbxgetpr(), dbxnkey(), dbxpkey()

## **dbxrmvkey()**

---

### **NAME**

**dbxrmvkey()**                      **remove key from all tags in tagged index file**

### **SYNOPSIS**

```
#include "dbl.h"
```

```
int    dbxrmvkey(dbx, key, recno)
```

```
<input parameters>
```

```
char    *dbx;           /* Tagged index file descriptor    */
char    *key;           /* Address of a buffer containing the key to be
                        removed                                */
long    recno;          /* Record number associated with key to be
                        removed                                */
```

```
<output parameters>
```

```
none
```

### **RETURN VALUE**

The `dbxrmvkey()` function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### **DESCRIPTION**

The `dbxrmvkey()` function searches for the specified key in the active tag, and then removes all keys in the tagged index file, `.DBX`, for the matching record number.

### **EXAMPLE**

The following example creates the shipwreck table and tags on the `VESSEL` and `FLAG` fields. It appends a record, adds the key, then removes and updates the key.

```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file */

static  dBFIELD fields[6] = {
    "VESSEL",  'C',   18,   0,   0,
    "LAT",     'N',   4,    1,   0,
    "LONG",    'N',   4,    1,   0,
    "AMOUNT",  'N',  10,    0,   0,
    "DATE",    'D',   8,    0,   0,
    "FLAG",    'C',   9,    0,   0
};

static void errorproc(
    char *func,
    char *str,
    int rc);

static char *text_table[] = {
    "Santa Margarita 80.027.5 700000095/01/01Spanish "
};

main()
{
    int rc;          /* Return Code for error handling */
    char *dbf;       /* File descriptor for table      */
    char *dbx;       /* File descriptor for tagged index */
    FLDNUM fieldcount; /* Number of fields to create    */
    int i;           /* Loop control variable          */
    long recno;      /* Table record variable          */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbicache(100);
    errorproc("dbicache()", "index cache specified.", rc);
    rc = dbfilemode(0,0);

    fieldcount = 6;

    rc = dbcreat("shipwreck.dbf", fieldcount, fields);
    errorproc("dbcreat()", "database created.", rc);

    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened.", rc);

```

```

rc = dbxcreate( dbf,
                fields[0].fieldnm,          /* "VESSEL", */
                &dbx,
                fields[0].fieldnm,          /* "VESSEL", */
                NULL,
                0,
                0);
errorproc("dbxcreate()", "tag created", rc);

rc = dbxcreate( dbf,
                fields[5].fieldnm,          /* "FLAG", */
                &dbx,
                fields[5].fieldnm,          /* "FLAG", */
                NULL,
                0,
                0);
errorproc("dbxcreate()", "tag created", rc);
rc = dbxclose(dbx);
errorproc("dbxclose()", "tagged index closed.", rc);

rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
errorproc("dbxopen()", "tagged index opened.", rc);

rc = dbappend(dbf, text_table[0], &recno);
errorproc("dbappend()", "record appended.", rc);
rc = dbxakey(dbx, recno);
errorproc("dbxakey()", "key added.", rc);

rc = dbflush(dbf);
errorproc("dbflush()", "table cache flushed.", rc);
rc = dbxflsh(dbx);
errorproc("dbxflsh()", "tagged index cache flushed.", rc);

rc = dbxrmvkey(dbx, "Santa Margarita ", recno);
errorproc("dbxrmvkey()", "key physically removed.", rc);

rc = dbxakey(dbx, recno);
errorproc("dbxakey()", "key added back in.", rc);

rc = dbxupd(dbx, recno);
errorproc("dbxupd()", "update key.", rc);

rc = dbxclose(dbx);
errorproc("dbxclose()", "tagged index closed.", rc);
rc = dbclose(dbf);
errorproc("dbclose()", "table closed.", rc);

```

```

        exit(0);
    }

    static void errorproc(func, str, rc)
    char    *func;
    char    *str;
    int      rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## SEE ALSO

dbxakey(), dbxupd()



## NAME

**dbxtag()**

**change the active tag**

## SYNOPSIS

```
#include "dbl.h"
```

```
int    dbxtag(dbx, tagname)
```

```
<input parameters>
```

```
char    *dbx          /* Tagged index file descriptor    */
char    *tagname       /* Address of a buffer containing the tag
                        to make active                      */
```

```
<output parameters>
```

```
none
```

## RETURN VALUE

The `dbxtag()` function returns 0 for success, or <0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

The `dbxtag()` function changes the active tag in the specified tag file, `.DBX`. All `dbx` functions in this library perform their operations on the active tag.

## EXAMPLE

The following example creates the shipwreck table and tags on the `VESSEL` and `FLAG` fields. It appends a record, reads the current key using the default `VESSEL` tag then changes the active tag to `FLAG` and reads the current key.

```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file   */

static  dBFIELD fields[6] = {
    "VESSEL",  'C',   18,   0,   0,
    "LAT",     'N',   4,    1,   0,
    "LONG",    'N',   4,    1,   0,
    "AMOUNT",  'N',  10,   0,   0,
    "DATE",    'D',   8,    0,   0,
    "FLAG",    'C',   9,    0,   0
};

static void errorproc(
    char *func,
    char *str,
    int rc);

static char *text_table[] = {
    "Santa Margarita 80.027.5 700000095/01/01Spanish "
};

main()
{
    int rc;          /* Return Code for error handling */
    char *dbf;       /* File descriptor for table      */
    char *dbx;       /* File descriptor for tagged index */
    FLDNUM fieldcount; /* Number of fields to create    */
    int i;           /* Loop control variable          */
    long recno;      /* Table record variable          */
    char key[21];    /* Storage location for keys      */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbicache(100);
    errorproc("dbicache()", "index cache specified.", rc);
    rc = dbfilemode(0,0);

    fieldcount = 6;
    rc = dbcreat("shipwreck.dbf", fieldcount, fields);
    errorproc("dbcreat()", "database created.", rc);

    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened.", rc);

```

```

rc = dbxcreate( dbf,
                fields[0].fieldnm,          /* "VESSEL", */
                &dbx,
                fields[0].fieldnm,          /* "VESSEL", */
                NULL,
                0,
                0);
errorproc("dbxcreate()", "tag created", rc);

rc = dbxcreate( dbf,
                fields[5].fieldnm,          /* "FLAG", */
                &dbx,
                fields[5].fieldnm,          /* "FLAG", */
                NULL,
                0,
                0);
errorproc("dbxcreate()", "tag created", rc);
rc = dbxclose(dbx);
errorproc("dbxclose()", "tagged index closed.", rc);

rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
errorproc("dbxopen()", "tagged index opened.", rc);

rc = dbappend(dbf, text_table[0], &recno);
errorproc("dbappend()", "record appended.", rc);
rc = dbxakey(dbx, recno);
errorproc("dbxakey()", "key added.", rc);

rc = dbflush(dbf);
errorproc("dbflush()", "table cache flushed.", rc);
rc = dbxflsh(dbx);
errorproc("dbxflsh()", "tagged index cache flushed.", rc);

rc = dbxckey(dbx, key, &recno);
errorproc("dbxckey()", "current key read.", rc);
printf("\t\tcurrent key: \t%s\n", key);

rc = dbxtag(dbx, fields[5].fieldnm);
errorproc("dbxtag()", "current tag changed to flag.", rc);

rc = dbxckey(dbx, key, &recno);
errorproc("dbxckey()", "current key read.", rc);
printf("\t\tcurrent key: \t%s\n", key);

rc = dbxclose(dbx);
errorproc("dbxclose()", "tagged index closed.", rc);

```

```

    rc = dbclose(dbf);
    errorproc("dbclose()", "table closed.", rc);

    exit(0);
}

static void errorproc(func, str, rc)
char    *func;
char    *str;
int      rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## SEE ALSO

dbxckey(), dbxclose(), dbxopen()

## NAME

**dbxtkey()** **translate tag key to record number**

## SYNOPSIS

```
#include "dbl.h"

int    dbxtkey(dbx, key, recno, softseek)

<input parameters>
char   *dbx;           /* Tagged index file pointer      */
char   key;            /* Address of a buffer containing the key */
int     softseek;       /* If 1, then perform a softseek      */

<output parameters>
long   *recno;          /* Address of a buffer where record number
                        is returned */
```

## RETURN VALUE

The dbxtkey() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

## DESCRIPTION

The dbxtkey() function translates a key into its associated record number by searching for the specified key in the active tag.

## EXAMPLE

The following example searches for “Santa Rosa” in the VESSEL tag of the shipwreck table and returns the record number.

```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file   */

static  dbFIELD fields[6] = {
    "VESSEL",  'C',   18,   0,   0,
    "LAT",     'N',    4,   1,   0,
    "LONG",    'N',    4,   1,   0,
    "AMOUNT",  'N',   10,   0,   0,
    "DATE",    'D',    8,   0,   0,
    "FLAG",    'C',    9,   0,   0
};

static void errorproc(
    char  *func,
    char  *str,
    int    rc);

main()
{
    int    rc;          /* Return Code for error handling */
    char  *dbf;         /* File descriptor for table      */
    char  *dbx;         /* File descriptor for tagged index */
    char  record[58];   /* Array of char for records      */
    char  status;       /* Deleted record flag            */
    long  recno;        /* Table record variable          */
    char  key[21];      /* Storage location for keys      */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbicache(100);
    errorproc("dbicache()", "index cache specified.", rc);
    rc = dbfilemode(1,0);
    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened in shared mode.", rc);
    rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
    errorproc("dbxopen()", "tagged index opened.", rc);

    rc = dbxtkey(dbx, "Santa Rosa", &recno, 0);
    errorproc("dbxtkey()", "record number found by key.", rc);
    printf("\t\trecord number: \t%d\n", recno);

    rc = dbxgetrk(dbf, dbx, "Santa Rosa", record, &status);
    errorproc("dbxgetrk()", "record retrieved by key.", rc);
    printf("\t\t%s\n", record);
}

```

```

    rc = dbxckey(dbx, key, &recno);
    errorproc("dbxckey()", "current key read.", rc);
    printf("\t\tcurrent key: \t%s\n", key);

    rc = dbxclose(dbx);
    errorproc("dbxclose()", "tagged index closed.", rc);
    rc = dbfclose(dbf);
    errorproc("dbfclose()", "table closed.", rc);

    exit(0);
}

static void errorproc(func, str, rc)
char    *func;
char    *str;
int     rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## SEE ALSO

dbxckey(), dbxclose(), dbxgetrk(), dbxopen()

## dbxunlocki()

---

### NAME

**dbxunlocki()**

**unlock tagged index file**

### SYNOPSIS

```
#include "dbl.h"

int      dbxunlocki(dbx)

<input parameters>
char      *dbx;                /* Tagged index file descriptor */

<output parameters>
none
```

### RETURN VALUE

The dbxunlocki() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

The dbxunlocki() function unlocks the specified tagged index file, .DBX.

### EXAMPLE

The following example opens the table and tag index, locks the table and tag index, adds in records, flushes the buffers, then unlocks and closes the files.



```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file   */

static  DBFIELD fields[6] = {
    "VESSEL",  'C',   18,   0,   0,
    "LAT",     'N',   4,    1,   0,
    "LONG",    'N',   4,    1,   0,
    "AMOUNT",  'N',  10,    0,   0,
    "DATE",    'D',   8,    0,   0,
    "FLAG",    'C',   9,    0,   0
};

static void errorproc(
    char  *func,
    char  *str,
    int    rc);

static char *text_table[] = {
    "Santa Margarita  80.027.5  7000000095/01/01Spanish ",
    "Santa Rosa      81.824.8  3000000032/01/01Spanish ",
    "Unknown galleon  83.123.0    015/01/01Spanish ",
    "Jessie          97.127.4  10000075/01/01U.S.   ",
    "Lea             96.227.8  100000080/01/01U.S.   ",
    "S.J. Lee        96.926.9  20000075/01/01U.S.   ",
    "Genovase        78.418.4  185000030/01/01Spanish ",
    "Unknown Vessel  77.517.9    075/01/01U.S.    "
};

main()
{
    int    rc;          /* Return Code for error handling */
    char  *dbf;         /* File descriptor for table      */
    char  *dbx;         /* File descriptor for tagged index */
    int    i;           /* Loop control variable          */
    long   recno;       /* Table record variable          */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbicache(100);
    errorproc("dbicache()", "index cache specified.", rc);
    rc = dbfilemode(1,0);
    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened in shared mode.", rc);

```

```

rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
errorproc("dbxopen()", "tagged index opened.", rc);

for( i = 1; i <= 7; i++) {
    rc = dbappend(dbf, text_table[i], &recno);
    errorproc("dbappend()", "record appended.", rc);
    rc = dbxlocki(dbx);
    errorproc("dbxlocki()", "tagged index file locked.", rc);
    rc = dbxakey(dbx, recno);
    errorproc("dbxakey()", "key added.", rc);
    rc = dbxunlocki(dbx);
    errorproc("dbxunlocki()", "tagged index file unlocked.", rc);
}

rc = dbflush(dbf);
errorproc("dbflush()", "table cache flushed.", rc);
rc = dbxflsh(dbx);
errorproc("dbxflsh()", "Tagged index cache flushed.", rc);

rc = dbxclose(dbx);
errorproc("dbxclose()", "tagged index closed.", rc);
rc = dbclose(dbf);
errorproc("dbclose()", "table closed.", rc);

exit(0);
}

static void errorproc(func, str, rc)
char    *func;
char    *str;
int      rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## SEE ALSO

dbappend(), dbxakey(), dbxclose(), dbxflsh(), dbxlocki(), dbxopen()

### NAME

**dbxupd()**                    **update all tags in the tagged index file for the specified record**

### SYNOPSIS

```
#include "dbl.h"

int    dbxupd(dbx, recno)

<input parameters>
char    *dbx;          /* Tagged index file descriptor      */
long    recno;         /* Record number to update tag from */

<output parameters>
none
```

### RETURN VALUE

The dbxupd() function returns 0 for success, or < 0 if an error occurs. See the section on return code values for a detailed list of return codes.

### DESCRIPTION

The dbxupd() function will update all tags in the specified tagged index file, .DBX, for the given record number

### EXAMPLE

The following example creates the shipwreck table and tags on the VESSEL and FLAG fields. It appends a record, adds the key, then removes and updates the key.

```

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file    */
#include "dblproto.h"     /* Recital/Library prototype file    */

static  dBFIELD fields[6] = {
    "VESSEL",  'C',   18,   0,   0,
    "LAT",     'N',   4,    1,   0,
    "LONG",    'N',   4,    1,   0,
    "AMOUNT",  'N',  10,    0,   0,
    "DATE",    'D',   8,    0,   0,
    "FLAG",    'C',   9,    0,   0
};

static void errorproc(
    char  *func,
    char  *str,
    int   rc);

static char *text_table[] = {
    "Santa Margarita 80.027.5 700000095/01/01Spanish "
};

main()
{
    int      rc;          /* Return Code for error handling    */
    char     *dbf;        /* File descriptor for table         */
    char     *dbx;        /* File descriptor for tagged index   */
    FLDNUM   fieldcount;  /* Number of fields to create        */
    int      i;           /* Loop control variable             */
    long     recno;       /* Table record variable             */

    rc = dbdcache(100);
    errorproc("dbdcache()", "table cache specified.", rc);
    rc = dbicache(100);
    errorproc("dbicache()", "index cache specified.", rc);
    rc = dbfilemode(0,0);

    fieldcount = 6;

    rc = dbcreat("shipwreck.dbf", fieldcount, fields);
    errorproc("dbcreat()", "database created.", rc);

    rc = dbopen("shipwreck.dbf", &dbf);
    errorproc("dbopen()", "table opened.", rc);

```

```

rc = dbxcreate( dbf,
                fields[0].fieldnm,          /* "VESSEL", */
                &dbx,
                fields[0].fieldnm,          /* "VESSEL", */
                NULL,
                0,
                0);
errorproc("dbxcreate()", "tag created", rc);

rc = dbxcreate( dbf,
                fields[5].fieldnm,          /* "FLAG", */
                &dbx,
                fields[5].fieldnm,          /* "FLAG", */
                NULL,
                0,
                0);
errorproc("dbxcreate()", "tag created", rc);
rc = dbxclose(dbx);
errorproc("dbxclose()", "tagged index closed.", rc);

rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
errorproc("dbxopen()", "tagged index opened.", rc);

rc = dbappend(dbf, text_table[0], &recno);
errorproc("dbappend()", "record appended.", rc);
rc = dbxakey(dbx, recno);
errorproc("dbxakey()", "key added.", rc);

rc = dbflush(dbf);
errorproc("dbflush()", "table cache flushed.", rc);
rc = dbxflsh(dbx);
errorproc("dbxflsh()", "tagged index cache flushed.", rc);

rc = dbxrmvkey(dbx, "Santa Margarita ", recno);
errorproc("dbxrmvkey()", "key physically removed.", rc);

rc = dbxakey(dbx, recno);
errorproc("dbxakey()", "key added back in.", rc);

rc = dbxupd(dbx, recno);
errorproc("dbxupd()", "update key.", rc);

rc = dbxclose(dbx);
errorproc("dbxclose()", "tagged index closed.", rc);
rc = dbclose(dbf);
errorproc("dbclose()", "table closed.", rc);

```

```

        exit(0);
    }

    static void errorproc(func, str, rc)
    char    *func;
    char    *str;
    int      rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## SEE ALSO

dbxakey(), dbxrmvkey()

# DBLSAMP PROGRAM

---

## DBLSAMP PROGRAM

The following program is written in 'C' as an example program using the RECITAL/Library functions. This file is stored in the SDK dbl sub-directory and is called "dblsamp.c".

```
/*#=====
*# Copyright (C) 1988-2004 Recital Corporation Inc.
*# As an unpublished licensed proprietary work.
*# All rights reserved worldwide.
*#
*# This program may be freely copied and used as a template for user
*# programs which make use of the Recital/Library in so long as the above
*# copyright notice is included.
*#
#=====
*
*MODULE          : dblsamp.c
*
*PURPOSE         : Recital Database Library sample program.
*
*AUTHOR          : RECITAL CORPORATION
*
*=====*/

#include <stdio.h>
#include "dbl.h"          /* Recital/Library include file      */
#include "dblproto.h"     /* Recital/Library prototype file    */

extern          int      g_disable_txcount;

/*-----*/
/* Table structure for db to be constructed. */
/* Structure dBFIELD defined in dbl.h */
/*-----*/
static dBFIELD fields[7] = {
    "VESSEL", 'C', 18, 0, 0,
    "LAT",    'N', 4, 1, 0,
    "LONG",   'N', 4, 1, 0,
    "AMOUNT", 'N', 10, 0, 0,
    "DATE",   'D', 8, 0, 0,
    "FLAG",   'C', 9, 0, 0,
    "MEMO",   'M', 4, 0, 0
};
```

```

/*-----*/
/* Static Error Handling procedure. */
/*-----*/
static int errorproc();

/*-----*/
/* Table records to add. */
/*-----*/
static char *text_table[] = {
    "Santa Margarita      80.027.5      7000000      95/01/01      Spanish ",
    "Santa Rosa           81.824.8      30000000     32/01/01     Spanish ",
    "Unknown galleon      83.123.0      0           15/01/01     Spanish",
    "Jessie               97.127.4      100000      75/01/01     U.S.  ",
    "Lea                  96.227.8      1000000     80/01/01     U.S.  ",
    "S.J. Lee             96.926.9      200000      75/01/01     U.S.  ",
    "Genovase             78.418.4      1850000     30/01/01     Spanish ",
    "Unknown Vessel       77.517.9      0           75/01/01     U.S.  "
};

/*-----*/
/* Main program entry point. */
/*-----*/
main()
{
    int rc; /* Return Code for error handling */
    char *dbf; /* File descriptor for table */
    char *dbx; /* File descriptor for tagged index */
    char *ndx; /* File descriptor for index */
    char *dbt; /* File descriptor for memo file */
    char record[58]; /* Array of char for records */
    char keytype; /* Variable for varying keytype */
    char key[21]; /* Storage location for keys */
    char tagname[21]; /* Storage location for tag name */
    char keyexpr[512]; /* Storage location for key expr */
    char forexpr[512]; /* Storage location for for expr */
    int kexprlen; /* The key expression length */
    int keylen; /* The key length */
    DBFIELD dbfields[128]; /* Storage location for field names */
    FLDNUM fieldcount; /* Number of fields to create */
    int fldpos[128]; /* Fields position buffer */
    char flddes[128][26]; /* Fields description buffer */
    char tmpbuf[81]; /* Temporary storage buffer */
    char fldbuf[128][256]; /* Field buffer (recin()/out()) */
    char nextkey[10][20]; /* Key buffers (dbnkey()) */
    char status; /* Deleted record flag */
    long recno; /* Table record variable */

```



```

long      rec_no[10];          /* Record number array (dbnkey()) */
long      db_size;            /* Size of table variable          */
int       reclen;             /* Table record length variable    */
char      month;              /* Month variable                  */
char      day;                /* Day variable                    */
char      year;               /* Year variable                   */
FLDNUM    nofields;           /* Number of table fields variable */
int       i;                  /* Loop control variable           */
double    n;                  /* Double numeric variable         */
int       keyunique;          /* Boolean for unique tag key       */
int       keydescend;         /* Boolean for descending tag key   */
int       notags;             /* number of tags in a dbx file    */

/*-----*/
/* Perform operational Library environment functions. */
/*-----*/

/*-----*/
/* DBDCACHE() - this example allocates a 100 record caching area */
/*-----*/
rc = dbdcache(100);
errorproc("dbdcache()", "table cache specified.", rc);

/*-----*/
/* DBICACHE() - this example allocates 51200 bytes for index caching */
/*-----*/
rc = dbicache(100);
errorproc("dbicache()", "index cache specified.", rc);

/*-----*/
/* DBFILEMODE() - shared, read write access */
/*-----*/
rc = dbfilemode(1,0);
errorproc("dbfilemode()", "file mode specified.", rc);

/*-----*/
/* DBSET() - Set exact index match to true */
/*-----*/
rc = dbset("EXACT", 1);
errorproc("dbset()", "filemode specified.", rc);

```

```

/*-----*/
/* Perform basic table functions. */
/*-----*/
fieldcount = 7;      /* Number of Fields */

/*-----*/
/* DBCREAT() - this example creates a table according to the 'DBFIELD' */
/* array fields. */
/*-----*/
rc = dbcreat("shipwreck.dbf", fieldcount, fields);
errorproc("dbcreat()", "database created.", rc);

/*-----*/
/* DBCREATX() - Enables the addition of table field descriptions. */
/*-----*/
strcpy(flddes[0], "The vessel's name.");
strcpy(flddes[1], "The latitude.");
strcpy(flddes[2], "The longitude.");
strcpy(flddes[3], "The value.");
strcpy(flddes[4], "The date.");
strcpy(flddes[5], "The vessel's flag.");
strcpy(flddes[6], "Notes.");

rc = dbcreatx("shipwreck.dbf", fieldcount, fields, flddes);

/*-----*/
/* DBOPEN() - example opens table 'shipwreck.dbf' returning the table */
/* file descriptor (dbf) used in upcoming functions */
/*-----*/
rc = dbopen("shipwreck.dbf", &dbf);
errorproc("dbopen()", "table opened.", rc);

/*-----*/
/* DBGETF() - obtain table information on the table referenced by the */
/* 'dbf' file descriptor. */
/*-----*/
rc = dbgetf(dbf, &reclen, &month, &day,
            &year, &nofields, dbfields);
errorproc("dbgetf()", "information retrieved.", rc);
printf("\t\tRecord length in bytes: %d\n", reclen);
printf("\t\tCreation date: \t\t%d/%d/%d\n", month, day, year);
printf("\t\tNumber of fields: \t%d\n", nofields);

```

```

/*-----*/
/* DBGETFX() - obtain extended table information including the numeric */
/* field location array (fldpos) and the field descriptions (flddes). */
/*-----*/
rc = dbgetfx(dbf, &reclen, &month, &day, &year,
            &nofields, dbfields, fldpos, flddes);
errorproc("dbgetfx()", "extra information retrieved.", rc);
printf("\t\tLocation of fields in buffer: \t%d,%d,%d,%d,%d,%d\n",
        fldpos[1],fldpos[2],fldpos[3],fldpos[4],fldpos[5],fldpos[6]);

/*-----*/
/* DBLOCKF() - locks the table for the additions that follow. */
/* Synonymous with DBTLOCKF(), which tests the lock */
/*-----*/
rc = dblockf(dbf);
if ( rc == SUCCESS ) {
    printf("Function: \tdblockf() \tfile locked successfully.\n");
} else {
    printf("Lock Failed. \tError: %d\n", rc);
}

/*-----*/
/* DBCLOSE() - close the active table */
/*-----*/
rc = dbclose(dbf);
errorproc("dbclose()", "table closed.", rc);

/*-----*/
/* Exclusive read write access of the table is required to create or delete */
/* tags. */
/*-----*/
rc = dbfilemode(0,0);
rc = dbopen("shipwreck.dbf", &dbf);
errorproc("dbopen()", "table opened in exclusive mode.", rc);

/*-----*/
/* Perform basic tagged index functions. */
/*-----*/

```



```
printf("\t\t\t Key number: \t%d\n", i );
printf("\t\t\t Key name: \t%s\n", tagName );
printf("\t\t\t Key type: \t%c\n", keytype );
printf("\t\t\t Key expression: %s\n", keyexpr);
printf("\t\t\t Key length: %d Expression length: %d\n",
        kexprlen, keylen);
printf("\t\t\t For expression: %s\n", forexpr);
printf("\t\t\t Key unique: \t%s\n", keyunique ? "Yes" : "No" );
printf("\t\t\t Key descending: \t%s\n", keydescend?"Yes":"No" );
}

/*-----*/
/* DBXDROPTAGS() - Drop the specified tag */
/*-----*/
rc = dbxdroptag(dbx, fields[5].fieldnm);
errorproc("dbxdroptag()", "tagged index 'FLAG' dropped.", rc);

/*-----*/
/* DBXCLOSE() - close the active tagged index */
/*-----*/
rc = dbxclose(dbx);
errorproc("dbxclose()", "tagged index closed.", rc);

/*-----*/
/* DBCLOSE() - close the active table */
/*-----*/
rc = dbclose(dbf);
errorproc("dbclose()", "table closed.", rc);

/*-----*/
/* Reopen the table in shared, read-write access mode */
/*-----*/
rc = dbfilemode(1,0);
rc = dbopen("shipwreck.dbf", &dbf);
errorproc("dbopen()", "table opened in shared mode.", rc);

/*-----*/
/* DBXOPEN() - example opens index file 'shipwreck.dbx' returning the
/* tagged index file descriptor (dbx) used in upcoming functions
/*-----*/
rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
errorproc("dbxopen()", "tagged index opened.", rc);
```

```

/*-----*/
/* DBXDROPTAGS() - Drop the specified tag */
/*-----*/

```

```
rc = dbxdroptag(dbx, fields[5].fieldnm);
errorproc("dbxdroptag()", "tagged index 'FLAG' dropped.", rc);
```

```

/*-----*/
/* DBXCLOSE() - close the active tagged index */
/*-----*/

```

```
rc = dbxclose(dbx);
errorproc("dbxclose()", "tagged index closed.", rc);
```

```

/*-----*/
/* DBCLOSE() - close the active table */
/*-----*/

```

```
rc = dbclose(dbf);
errorproc("dbclose()", "table closed.", rc);
```

```

/*-----*/
/* Reopen the table in shared, read-write access mode */
/*-----*/

```

```
rc = dbfilemode(1,0);
rc = dbopen("shipwreck.dbf", &dbf);
errorproc("dbopen()", "table opened in shared mode.", rc);
```

```

/*-----*/
/* DBXOPEN() - example opens index file 'shipwreck.dbx' returning the */
/* tagged index file descriptor (dbx) used in upcoming functions      */
/*-----*/

```

```
rc = dbxopen(dbf, "shipwreck.dbx", &dbx);
errorproc("dbxopen()", "tagged index opened.", rc);
```

```

/*-----*/
/* Perform basic index functions. */
/*-----*/

/*-----*/
/* DBICREAT() - in this example creates an index on field VESSEL */
/*-----*/
rc = dbicreat( "shipwreck.ndx", fields[0].fieldnm, 18, 'C');
errorproc("dbicreat()", "index created", rc);

/*-----*/
/* DBIOPEN() - opens index to active status */
/*-----*/
rc = dbiopen( "shipwreck.ndx", &ndx);
errorproc("dbiopen()", "index opened.", rc);

/*-----*/
/* DBKEXPR() - obtains index information */
/*-----*/
rc = dbkexpr(ndx, &keytype, keyexpr, &kexprlen, &keylen);
errorproc("dbkexpr()", "key information retrieved.", rc);
printf("\t\t\t Key type: \t%c\n", keytype );
printf("\t\t\t Key expression: %s\n", keyexpr);
printf("\t\t\t Key length: %d\t Expression length: %d\n",
\t\t\t kexprlen, keylen);

/*-----*/
/* Lock data file */
/*-----*/
rc = dblockf(dbf);
if ( rc == SUCCESS ) {
\t\t\t printf("Function: \tdblockf() \tfile locked successfully.\n");
} else {
\t\t\t printf("Lock Failed. \tError: %d\n", rc);
}

/*-----*/
/* DBLOCKI() - locks the index for following additions */
/* Synonymous with DBTLOCKI() which tests the lock */
/*-----*/
rc = dblocki(ndx);
if ( rc == SUCCESS ) {
\t\t\t printf("Function: \tdblocki() \tindex locked successfully.\n");
} else {
\t\t\t printf("Lock Failed. \tError: %d\n", rc);
}

```

```

/*-----*/
/* Perform basic memo file functions. */
/*-----*/

/*-----*/
/* DBMCREAT() - creates a table memo file (if one needs creating) */
/*-----*/
rc = dbmcreat("shipwreck.dbt");
errorproc("dbmcreat()", "memo file create", rc);

/*-----*/
/* DBMOPEN() - opens an associated memo file */
/*-----*/
rc = dbmopen("shipwreck.dbt", &dbt);
errorproc("dbmopen()", "memo file open.", rc);

/*-----*/
/* Perform table input functions. */
/*-----*/

/*-----*/
/* DBAPPEND() - appends a record to the end of the active table */
/*-----*/
rc = dbappend(dbf, text_table[0], &recno);
errorproc("dbappend()", "record appended.", rc);

/*-----*/
/* DBAKEY() - adds a key to the active index after dbappend() */
/*-----*/
rc = dbakey(ndx, "Santa Margarita ", recno);
errorproc("dbakey()", "key added.", rc);

/*-----*/
/* DBXAKEY() - adds a key to the tagged index after dbappend() */
/* The tagged index file must be lock first */
/*-----*/
rc = dbxlocki(dbx);
errorproc("dbxlocki()", "tagged index file locked.", rc);
rc = dbxakey(dbx, recno);
errorproc("dbxakey()", "key added.", rc);
rc = dbxunlocki(dbx);
errorproc("dbxunlocki()", "tagged index file unlocked.", rc);

```

```

/*-----*/
/* DBFLUSH() - flushes the record cache to disk */
/*-----*/
rc = dbflush(dbf);
errorproc("dbflush()", "table cache flushed.", rc);

/*-----*/
/* DBIFLSH() - flushes the index cache to disk */
/*-----*/
rc = dbiflsh(ndx);
errorproc("dbiflsh()", "index cache flushed.", rc);

/*-----*/
/* DBXFLSH() - flushes the tagged index cache to disk */
/*-----*/
rc = dbxflsh(dbx);
errorproc("dbxflsh()", "Tagged index cache flushed.", rc);

/*-----*/
/* DBGETR() - retrieves a record */
/*-----*/
rc = dbgetr(dbf, (long)1, record, &status);
errorproc("dbgetr()", "record retrieved.", rc);
printf("\t\t%s\n", record);

/*-----*/
/* DBPUTM() - put the memo information */
/*-----*/
rc = dbputm(dbt, "128' 30% coral cover.", &record[53]);
errorproc("dbputm", "memo placed in buffer", rc);

/*-----*/
/* DBUPDR() - updates the record in buffer 'record' */
/* As of v8.0 this function automatically updates any attached indexes. */
/*-----*/
rc = dbupdr(dbf, (long)1, record);
errorproc("dbupdr()", "record updated.", rc);

/*-----*/
/* DBGETM() - this example extracts the memo from the record buffer */
/*-----*/
rc = dbgetm(dbt, &record[53], tmpbuf, 0);
errorproc("dbgetm()", "retrieved memo.", rc);

```



```

/*-----*/
/* DBUPDM() - Updates the memo field for this record */
/*-----*/
strcat(tmpbuf, " Appending this string to the end of memo...");
rc = dbupdm(dbt, tmpbuf, &record[53], &record[53]);
errorproc("dbupdm()", "updated memo.", rc);

/*-----*/
/* DBPUTRK() - the following example adds a record and index */
/* to the respective files. */
/* DBRECOUT() - scatter the record stored in buffer 'record' to a two */
/* dimensional character array 'fldbuf' */
/*-----*/

/*-----*/
/* Read in 6 more records */
/*-----*/

for( i = 1; i <= 6; i++) {
dbrecout( dbf, dbfields, fldpos, text_table[i], fldbuf);
rc = dbputrk(dbf, ndx, fldbuf[0], text_table[i]);
errorproc("dbputrk()", "record placed by key.", rc);
}
errorproc("dbputrk()", "6 records placed by key.", rc);

/*-----*/
/* DBFLUSH() - flushes the record cache to disk */
/*-----*/
rc = dbflush(dbf);
errorproc("dbflush()", "table cache flushed.", rc);

/*-----*/
/* DBIFLSH() - flushes the index cache to disk */
/*-----*/
rc = dbiflsh(ndx);
errorproc("dbiflsh()", "index cache flushed.", rc);

/*-----*/
/* DBXFLSH() - flushes the tagged index cache to disk */
/*-----*/
rc = dbxflsh(dbx);
errorproc("dbxflsh()", "Tagged index cache flushed.", rc);

```

```

/*-----*/
/* DBFIELD() - this example extracts the field 'VESSEL' from the      */
/* record (#5) stored in buffer 'record'                               */
/*-----*/
rc = dbgetr(dbf, (long)5, record, &status);
/* rc = dbfield( dbf, record, "FLAG", tmpbuf); */
rc = dbfield( dbf, record, fields[5].fieldnm, tmpbuf);
errorproc("dbfield()", "field extracted.", rc);
printf("\t\tFlag: \t%s\n", tmpbuf);

/*-----*/
/* Modify a single field in record number 5                           */
/*-----*/

/*-----*/
/* DBSCATTER() - scatter the record into specified field buffers      */
/*-----*/
rc = dbscatter(dbf, record, fldbuf);
errorproc("dbscatter()", "record scattered.", rc);

for (i=0; i < nofields; i++) {
    fprintf(stdout, "\t\tfldbuf[%d]='%s'\n", i, fldbuf[i]);
}
strcpy(fldbuf[5], "Spanish");    /* Modify Field */

/*-----*/
/* DBGATHER() - gather the field buffers and place the contents      */
/*-----*/
rc = dbgather(dbf, record, fldbuf);
errorproc("dbgather()", "record gathered.", rc);
printf("\t\t%s\n", record);

rc = dbupdr(dbf, (long)5, record);    /* Finish Example */
errorproc("dbupdr()", "record updated.", rc);

/*-----*/
/* Perform index / index conversion functions.                       */
/*-----*/

/*-----*/
/* DBREWIND() - position the record pointer to the top of the index  */
/*-----*/
rc = dbrewind(ndx);
errorproc("dbrewind()", "index rewound.", rc);

```

```

/*-----*/
/* DBGETNR() - after 'rewinding' to the top of the table, the first record */
/* can be retrieved with dbgetnr() using the active key */
/*-----*/
for( i = 1; i <= 6; i++) {
    rc = dbgetnr(dbf, ndx, record, &status);
    printf("\t\t%d: %s\n", i, record);
}
errorproc("dbgetnr()", "6 records retrieved by key.", rc);

/*-----*/
/* DBNKEY() - get next key */
/*-----*/
rc = dbnkey(ndx, nextkey[0], rec_no);
errorproc("dbnkey()", "next key retrieved.", rc);
printf("\t\tnext key: \t%s\n", nextkey[0]);

/*-----*/
/* DBFWD() - position the pointer to the end of the table */
/*-----*/
rc = dbfwd(ndx);
errorproc("dbfwd()", "index pointer placed at bottom.", rc);

/*-----*/
/* DBPKEY() - get previous key */
/*-----*/
rc = dbpkey(ndx, nextkey[0], rec_no);
errorproc("dbpkey()", "previous key found.", rc);
printf("\t\tprev key: \t%s\n", nextkey[0]);

/*-----*/
/* DBGETPR() - obtain the previous record by key */
/*-----*/
rc = dbgetpr(dbf, ndx, record, &status);
errorproc("dbgetpr()", "previous record (by key) retrieved.", rc);
printf("Previous: \t%s\n", record);

/*-----*/
/* Rewind the index */
/*-----*/
dbrewind(ndx);

```

```
for( i = 1; i <= 6; i++) {
    rc = dbgetnr(dbf, ndx, record, &status);
    printf("\t\t%d: %s\n", i, record);
}
errorproc("dbgetnr()", "6 records retrieved by key.", rc);
```

```
/*-----*/
/* DBNKEY() - get next key */
/*-----*/
```

```
rc = dbnkey(ndx, nextkey[0], rec_no);
errorproc("dbnkey()", "next key retrieved.", rc);
printf("\t\tnext key: %t%s\n", nextkey[0]);
```

```

/*-----*/
/* DBFWD() - position the pointer to the end of the table */
/*-----*/

```

```
rc = dbfwd(ndx);
errorproc("dbfwd()", "index pointer placed at bottom.", rc);
```

```
/*-----*/
/* DBPKEY() - get previous key */
/*-----*/
```

```
rc = dbpkey(ndx, nextkey[0], rec_no);
errorproc("dbpkey()", "previous key found.", rc);
printf("\t\tprev key: \t%s\n", nextkey[0]);
```

```

/*-----*/
/* DBGETPR() - obtain the previous record by key */
/*-----*/

```

```
rc = dbgetpr(dbf, ndx, record, &status);
errorproc("dbgetpr()", "previous record (by key) retrieved.", rc);
printf("Previous: \t%s\n", record);
```

```
/*-----*/
/* Rewind the index                                */
/*-----*/
```

```
dbrewind(ndx);
```

```

/*-----*/
/* Perform table retrieval functions using indexes. */
/*-----*/

/*-----*/
/* DBTKEY() - translates supplied key to record number */
/*-----*/
rc = dbtkey(ndx, "Santa Rosa", &recno);
errorproc("dbtkey()", "record number found by key.", rc);
printf("\t\trecord number: \t%d\n", recno);

/*-----*/
/* DBGETRK() - obtains the record from the specified key */
/*-----*/
rc = dbgetrk(dbf, ndx, "Santa Rosa", record, &status);
errorproc("dbgetrk()", "record retrieved by key.", rc);
printf("\t\t\t%s\n", record);

/*-----*/
/* DBCKEY() - read the current key */
/*-----*/
rc = dbckey(ndx, key, &recno);
errorproc("dbckey()", "current key read.", rc);
printf("\t\t\tcurrent key: \t%s\n", key);

/*-----*/
/* Perform tagged index functions. */
/*-----*/

/*-----*/
/* DBXREWIND() - position the pointer to the top of the index file */
/*-----*/
rc = dbxrewind(dbx);
errorproc("dbxrewind()", "Tagged index rewound.", rc);

/*-----*/
/* DBXGETNR() - after 'rewinding' to the top of the table, the first record */
/* can be retrieved with dbgetnr() using the active key */
/*-----*/
for( i = 1; i <= 6; i++) {
    rc = dbxgetnr(dbf, dbx, record, &status);
    printf("\t\t\t%d: %s\n", i, record);
}
errorproc("dbxgetnr()", "6 records retrieved by tag key.", rc);

```

```

/*-----*/
/* DBXNKEY() - get next key */
/*-----*/
rc = dbxnkey(dbx, nextkey[0], rec_no);
errorproc("dbxnkey()", "next key retrieved in tagged index.", rc);
printf("\t\tnext key: \t%s\n", nextkey[0]);

/*-----*/
/* DBXFWD() - position the pointer to the end of the table */
/*-----*/
rc = dbxfwd(dbx);
errorproc("dbxfwd()", "index pointer placed at bottom.", rc);

/*-----*/
/* DBXPKEY() - get previous key */
/*-----*/
rc = dbxpkey(dbx, nextkey[0], rec_no);
errorproc("dbxpkey()", "previous key found in tagged index.", rc);
printf("\t\tprev key: \t%s\n", nextkey[0]);

/*-----*/
/* DBXGETPR() - obtain the previous record by key */
/*-----*/
rc = dbxgetpr(dbf, dbx, record, &status);
errorproc("dbxgetpr()", "previous record (by key) retrieved.", rc);
printf("Previous: \t%s\n", record);

/*-----*/
/* Rewind the index */
/*-----*/
dbxrewind(dbx);

/*-----*/
/* Perform table retrieval functions using tagged indexes. */
/*-----*/

/*-----*/
/* DBTKEY() - translates supplied key to record number */
/*-----*/
rc = dbtkey(ndx, "Santa Rosa", &recno);
errorproc("dbtkey()", "record number found by key.", rc);
printf("\t\trecord number: \t%d\n", recno);

```

```
rc = dbxnkey(dbx, nextkey[0], rec_no);
errorproc("dbxnkey()", "next key retrieved in tagged index.", rc);
printf("\t\tnext key: \t%s\n", nextkey[0]);
```

```
rc = dbxfwd(dbx);
errorproc("dbxfwd()", "index pointer placed at bottom.", rc);
```

```
rc = dbxpkey(dbx, nextkey[0], rec_no);
errorproc("dbxpkey()", "previous key found in tagged index.", rc);
printf("\t\tprev key: %s\n", nextkey[0]);
```

```
rc = dbxgetpr(dbf, dbx, record, &status);
errorproc("dbxgetpr()", "previous record (by key) retrieved.", rc);
printf("Previous: \t%s\n", record);
```

```
dbxrewind(dbx);
```

```

/*-----*/
/* DBTKEY() - translates supplied key to record number */
/*-----*/

```

```

/*-----*/
/* DBXGETRK() - obtains the record from the specified key */
/*-----*/
rc = dbxgetrk(dbf, dbx, "Santa Rosa", record, &status);
errorproc("dbxgetrk()", "record retrieved by key.", rc);
printf("\t\t%s\n", record);

/*-----*/
/* DBXCKEY() - read the current key */
/*-----*/
rc = dbxckey(dbx, key, &recno);
errorproc("dbxckey()", "current key read.", rc);
printf("\t\tcurrent key: %s\n", key);

/*-----*/
/* Perform field level functions including index/ASCII functions. */
/*-----*/

/*-----*/
/* DBATOFD() - converts ASCII string to Recital field */
/*-----*/
rc = dbatofld("23.3", 4, 1, tmpbuf);
errorproc("dbatofld()", "string converted.", rc);

/*-----*/
/* DBATOKEY() - converts a ASCII string to Recital date key */
/*-----*/
key[0] = 'D';
rc = dbatoken("16430411", key);
errorproc("dbatoken()", "string converted to key.", rc);

/*-----*/
/* DBFLDTOA() - converts extracted field to ASCII */
/*-----*/
rc = dbfldtoa(record + 18, 4, tmpbuf);
errorproc("dbfldtoa()", "field converted.", rc);

/*-----*/
/* DBSTRCPY() - formats character strings to Recital types */
/*-----*/
dbstrcpy(key, 'L', 10, " 320000");
printf("Function: \tdbstrcpy() \toutput: %s\n", key);

```

```

/*-----*/
/* DBSTRING() - formats non-'C' strings to C strings (null term) */
/*-----*/
dbstring("Santa Pedro", 13, tmpbuf);
printf("Function: \tdbstring() \toutput: %s\n", tmpbuf);

/*-----*/
/* DBKEYTOA() - converts an extracted key to ASCII */
/*-----*/
tmpbuf[0] = 0;      /* Number of decimal places */
tmpbuf[1] = 'N';    /* Key Type */
rc = dbkeytoa(key, tmpbuf);
errorproc("dbkeytoa()", "key converted.", rc);

/*-----*/
/* Perform record/index deletion and file access */
/*-----*/

/*-----*/
/* DBSIZE() - retrieve the size of the table */
/*-----*/
rc = dbsize(dbf, &db_size);
errorproc("dbsize()", "size retrieved.", rc);
printf("\t\t No. of Records: \t%d\n", db_size);

/*-----*/
/* DBPUTR() - adds a record to the end of the table */
/* You can also 'insert' a record using this function */
/*-----*/
rc = dbputr(dbf, db_size + 1L, text_table[7]);
errorproc("dbputr()", "record placed.", rc);
rc = dbakey(ndx, "Unknown Vessel ", db_size + 1L);
errorproc("dbakey()", "key added.", rc);
/*-----*/
/* Add record to the Production Index as well */
/*-----*/
rc = dbxakey(dbx, db_size + 1L);
errorproc("dbxakey()", "key added.", rc);

dbflush(dbf);
dbiflsh(ndx);
dbxflsh(dbx);

```

```

/*-----*/
/* DBDELETE() - mark a record (number 8) for deletion */
/*-----*/
rc = dbdelete(dbf, 8L);
errorproc("dbdelete()", "record 8 marked for deletion.", rc);

/*-----*/
/* DBRECALL() - recall a record marked for deletion (number 8) */
/*-----*/
rc = dbrecall(dbf, 8L);
errorproc("dbrecall()", "record 8 recalled.", rc);

/*-----*/
/* DBRMVKEY() - physically remove a key. As of v8.0, this function */
/* is not permitted on tables opened shareable. */
/*-----*/
/* rc = dbrmvkey(ndx, "Unknown Vessel ", db_size+1L); */
/* errorproc("dbrmvkey()", "key physically removed.", rc); */

/*-----*/
/* DBRMVR() - physically remove a data record. As of v8.0 this */
/* function is not permitted on tables opened shareable. */
/*-----*/
/* rc = dbrmvr(dbf, db_size+1L); */
/* errorproc("dbrmvr()", "record 7 physically removed.", rc); */

/*-----*/
/* DBUNLOCKF() - unlocks a table file */
/*-----*/
rc = dbunlockf(dbf);
if ( rc == SUCCESS ) {
    printf("Function: \tdbunlockf() \tFile unlocked successfully\n");
} else {
    printf("Unlock Failed. \tError: %d\n", rc);
}

/*-----*/
/* DBUNLOCKI() - unlocks an index file */
/*-----*/
rc = dbunlocki(ndx);
if ( rc == SUCCESS ) {
    printf("Function: \tdbunlocki() \tFile unlocked successfully\n");
} else {
    printf("Unlock Failed. \tError: %d\n", rc);
}

```



```

/*-----*/
/* DBLOCKR() - tests available record locking in a multi user      */
/* environment. Same as DBTLOCKR() in multi access                  */
/*-----*/
rc = dblockr(dbf, (long)7);
if ( rc == SUCCESS ) {
    printf("Function: \tdblockr() \trecored locked successfully.\n");
} else {
    printf("Lock Failed. \tError: %d\n", rc);
}

/*-----*/
/* DBUNLOCKR() - unlocks a table Record                            */
/*-----*/
rc = dbunlockr(dbf, (long)7);
if ( rc == SUCCESS ) {
    printf("Function \tdbunlockr(): \trecored unlocked successfully\n");
} else {
    printf("Unlock Failed. \tError: %d\n", rc);
}

/*-----*/
/* Perform record/index close functions                             */
/*-----*/

/*-----*/
/* DBICLOSE() - close the active index                             */
/*-----*/
rc = dbiclose(ndx);
errorproc("dbiclose()", "index closed.", rc);

/*-----*/
/* DBXCLOSE() - close the active tagged index                      */
/*-----*/
rc = dbxclose(dbx);
errorproc("dbxclose()", "tagged index closed.", rc);

/*-----*/
/* DBCLOSE() - close the active table                              */
/*-----*/
rc = dbclose(dbf);
errorproc("dbclose()", "table closed.", rc);

```

```

/*-----*/
/* DBMCLOSE() - close the active memo table */
/*-----*/
rc = dbmclose(dbt);
errorproc("dbmclose()", "memo table closed.", rc);

/*-----*/
/* Perform miscellaneous Recital/Library functions */
/*-----*/

/*-----*/
/* DBONERROR() - enables/disables (1/0) recovery from errors */
/*-----*/
dbonerror(0);      /* Recovery disabled */
printf("Function: \tdbonerror() \terror checking disabled\n");

printf("\nEnd of Recital C Library test program....\n");

exit(0);

}

/*-----*/
/* This procedure handles the return codes from called RECITAL/library */
/* functions. Intended to eliminate cryptic 'if...else...' statements */
/* following library calls. */
/*-----*/

static int errorproc(func, str, rc)
char *func;
char *str;
int rc;
{
    if ( rc != SUCCESS ) {
        printf("\n Error performing function %s -> %d\n", func, rc);
        exit(1);
    }
    printf("Function: \t%s, \t%s - Ok\n", func, str);
    return;
}

```

## ODBC COMPATIBLE FUNCTIONS

On Recital supported UNIX and Linux platforms, the following ODBC compatible functions are also available in addition to the standard Recital/Library functions.

<b>FUNCTION</b>	<b>DESCRIPTION</b>
SQLAllocConnect()	Allocate a connection handle.
SQLAllocEnv()	Allocate an environment handle.
SQLAllocStmt()	Allocate a new statement and associate it with a connection handle.
SQLBindCol()	Bind application storage space to columns in a result set.
SQLBindParameter()	Bind an application variable to a parameter marker.
SQLColAttributes()	Return descriptor information for a column.
SQLColumnPrivileges()	Return a list of columns and associated privileges for the specified table.
SQLColumns()	Return a list of column names for the specified tables.
SQLConnect()	Establish a connection to a database.
SQLDataSources()	Return information about a data source. This function is implemented solely by the Driver Manager.
SQLDescribeCol()	Return column information.
SQLDescribeParam()	Return the description of a parameter marker.
SQLDisconnect()	Disconnect a connection from a database.
SQLError()	Return information about the most recent error.
SQLExecDirect()	Prepare and execute an SQL statement.
SQLExecImmediate()	Execute an SQL statement.
SQLExecute()	Execute a previously prepared dynamic SQL statement.
SQLFetch()	Advance cursor to next row.
SQLForeignKeys()	Return a list of foreign keys in the specified table.
SQLFreeConnect()	Free connection from a database.
SQLFreeEnv()	Free environment handle.
SQLFreeStmt()	Close a cursor.
SQLGetConnectOption()	Return the current setting of a connection option.
SQLGetCursorName()	Associate a cursor name with a statement handle.
SQLGetData()	Retrieve data for a single column in the result set. It can be called multiple times to retrieve variable length data in parts.
SQLGetInfo()	Return general information about the currently connected DBMS.
SQLGetStmtOption()	Set options related to the HSTMT.
SQLGetTypeInfo()	Return information about data types supported by the data source.
SQLLockTable()	Apply the specified lock to the table.
SQLMoreResults()	Determine whether there are more results available on a statement containing SELECT, UPDATE, INSERT, or DELETE statements and, if so, initialize processing for those results.

SQLNativeSql()	Return the SQL string as translated by the driver.
SQLNumParams()	Return the number of parameter markers.
SQLNumResultCols()	Returns the number of columns in a result set.
SQLParamData()	Used in conjunction with SQLPutData to supply parameter data at statement execution time.
SQLPrepare()	Associate a cursor name with a statement handle.
SQLPrimaryKeys()	Return the column names that make up the primary key for a table.
SQLProcedureColumns()	Return a list of input and output parameters, as well as the columns that make up the result set for the specified procedures.
SQLProcedures()	Return the list of procedure names stored in a specific data source. Procedure is a generic term used to describe an executable object, or a named entity that can be invoked using input and output parameters.
SQLPutData()	Allow an application to send data for a parameter or column to the driver at statement execution time.
SQLRowCount()	Build a scroll cursor.
SQLSavePoint()	Set or roll back a save point on the current transaction.
SQLSetConnectOption()	Set connection options.
SQLSetCursorName()	Associate a cursor name with a statement handle.
SQLSetLocking()	Change the transaction locking type.
SQLSetParam()	Bind application storage space to columns in a result set.
SQLSetPos()	Set the cursor position.
SQLSetStmtOption()	Set SQL statement options.
SQLSpecialColumns()	Set up fetch for special column names.
SQLStatistics()	Retrieve a list of statistics about a single table and the indexes associated with the table.
SQLTablePrivileges()	Return a list of tables and the privileges associated with each table.
SQLTables()	Return a list of table names in the specified data source.
SQLTransact()	Commit or roll back the current transaction.

NOTE: To use the ODBC compatible functions, the following shared object library must be loaded or linked into your application:

<path>/recital-X.X/shared/libodbc.recital.so