A B-Tree Access Method
for QuickBASIC/PDS Programmers

QBTREE v5.01

April 7, 1991

The QBTREE package is a shareware product. You may try QBTREE to see if it fits your needs on a trial basis only. You may copy and distribute this shareware package freely. If you plan to use it after the trial, fill out the registration form at the end of this document and send it along with payment to:

Cornel Huth
ATTN: QBTREE 5.01 REGISTRATION
6402 Ingram Rd.
San Antonio, TX 78238

Source code for the QBTREE interface will be sent upon receipt of the completed registration form. This interface is all that is needed to modify QBTREE for most situations since all I/O and storage allocation is performed at this level. If you also need the low-level btree routine, I'll make it available. It's written in MASM assembly. (Contact me for more details.)

QBTREE requires QuickBASIC 4.00+ or BASIC 7.0+. For file sharing and record locking functions and for accessing more than 15 files at one time DOS 3.1+ is needed

Read this documentation before trying to use QBTREE, much has changed from previous versions.

LICENSE AGREEMENT

This is a legal agreement between you, the end user, and Cornel Huth. By using this software, you are agreeing to be bound by the terms of this agreement.

SOFTWARE LICENSE

1. GRANT OF LICENSE. Cornel Huth grants to you the right to use one copy of the SOFTWARE on a single terminal connected to a single computer (i.e., with a single CPU). You may not network the SOFTWARE or otherwise use it on more than one computer or computer terminal at a time.

2. COPYRIGHT. The SOFTWARE is owned by Cornel Huth and is protected by United States copyright laws and iternational treaty provisions. Therefore, you must treat the SOFTWARE like any other copyrighted material (e.g., a book or musical recording) except that you may either (a) make one copy of the SOFTWARE soley for backup or archival purposes, or (b) transfer the SOFTWARE to a single hard disk provided you keep the original solely for backup or archival purposes. You may make a single copy of this document for your own use only.

3. OTHER RESTRICTIONS. You may not rent or lease the SOFTWARE, but you may transfer the software and accompanying documentation on a permanent basis provided you retain no copies and the recipient agrees to the terms of this Agreement. You may not reverse engineer, decompile, or disassemble the software. If the SOFTWARE is an update, any transfer must include the most recent update and all previous versions.

NO WARRANTIES. Cornel Huth disclaims all warranties, either expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to the SOFTWARE and the documentation.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES. In no event shall Cornel Huth be liable for any damages whatsoever arising out of use of or inability to use this SOFTWARE.

U.S. GOVERNMENT RESTRICTED RIGHTS

The SOFTWARE and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at 52.227-7013. Contractor/manufacturer is Cornel Huth/6402 Ingram Rd/San Antonio, TX 78238.(512)684-8065.

This agreement is governed by the laws of the state of Texas.

WHAT IS IT?

QBTREE is a keyed-file system based on the b-tree sorting method that provides fast and efficient medium-level data base functions to the QB/PDS programmer. It maintains up to 250 key and data files at one time. Not only does QBTREE find any particular key and its data record very quickly - any key from a million-key file in at most 1/10 second on a 18ms hard disk - it also allows forward and reverse sequential inorder access to the data file. The data file is automatically maintained and the index file automatically balanced. Deletions in both the data and key files are made available for reuse, thus making repacking unnecessary. New with version 5 are additional routines to support key file-only maintenance where the data file format is left to you.

The following files should be found in this shareware package:

QBTREE5N.LIB	for use with QuickBASIC 4.xx
QBTREE5F.LIB	for use with PDS 7.x using far strings
QBTREE50.BI	declares
QBTREE50.PRN	this documentation
UPDATE.DOC	important changes
DEMOQBT.EXE	demo
DEMOQBT.BAS	source to the demo
XAPP1.BAS	example application

INSTALLATION

QBTREE5N.LIB was compiled with BC 4.00b (/O) and is intended for use with QuickBASIC 4.xx. QBTREE5F.LIB was compiled with BC 7.1 (/O/Ot/Fs) and is intended for use with BASIC 7.x when using far strings and for the QBX environment.

Add QBTREE5x.LIB (where x is either N for QB4 or F for PDS7) to your standard library if you like, and then LINK /QU it into a QLB library for the environment.

```
C>lib yourstd.lib + QBTREE5x.LIB;

C>link /qu yourstd.lib,qb.qlb,nul,bqlb4x.lib
```

You may prefer to keep QBTREE5x.LIB as a separate library, instead.

```
C>link /qu yourstd.lib+QBTREE5x.LIB,qb.qlb,nul,bqlb4x.lib
```

For BASIC 7.x users substitute qbxqlb.lib for bqlb4x.lib. Don't forget to invoke QB with the /L libraryname parameter.

When linking a stand-alone program include the library in the link process by adding QBTREE5x.LIB to the libraries prompt from LINK.EXE. If you are using a compiler other than BC 4.00b (BC 7.1 for QBTREE5F.LIB), LINK will ask for the path to BCOM41.LIB. For the shareware evaluation you can enter the path to your run-time library. If you are using BC 4.00 (you should upgrade) the name is \path\BCOM40.LIB. If you are using BC 4.5 the name is

\path\BCOM45.LIB. BC 7.0 will most likely be \path\BCL70EFR.LIB. With registration you receive the source, ready to be compiled by your compiler version. Thereafter LINK won't prompt you for the run-time library path. See COMPILER INFORMATION and RESOURCE ALLOCATION below on how to regenerate QBTREE from the source for your compiler version.

INTERFACE SUMMARY

All QBTREE routines are FUNCTIONs and return an integer code which is detailed in ERROR CODES. These must be DECLAREd. Use REM $INCLUDE:"QBTREE50.BI" in your programs.

 1) AddKey% (kfile%, dfile%, Qkey$)
 2) AddKeyRecord% (kfile%, dfile%, Qkey$, Qrec$)
 3) CloseDataFile% (dfileno%)
 4) CloseKeyFile% (kfileno%)
 5) CreateDataFile% (filename$, recl%)
 6) CreateKeyFile% (filename$, keyl%)
 7) DeleteKey% (kfile%, Qkey$)
 8) DeleteKeyRecord% (kfile%, dfile%, Qkey$)
 9) ExitQBTREE% ()
10) FlushDataFile% (dfileno%, dup%)
11) FlushKeyFile% (kfileno%, dup%)
12) GetDirect% (dfileno%, recno&, Qrec$)
13) GetEqual% (kfile%, dfile%, Qkey$, Qrec$)
14) GetFirst% (kfile%, dfile%, Qkey$, Qrec$)
15) GetLast% (kfile%, dfile%, Qkey$, Qrec$)
16) GetNext% (kfile%, dfile%, Qkey$, Qrec$)
17) GetPosition% (kfileno%, recno&)

18) GetPrev% (kfile%, dfile%, Qkey$, Qrec$)
19) InitQBTREE% (MKF%, MDF%)
20) LoadDataHeader% (dfileno%)
21) LoadKeyHeader% (kfileno%)
22) LockDataHeader% (dfileno%)
23) LockKeyFile% (kfileno%)
24) LockRecord% (dfileno%, recno&)
25) OpenDataFile% (filename$, dfileno%)
26) OpenKeyFile% (filename$, kfileno%)
27) RetrieveEqual% (kfile%, Qkey$, Qurecno&)
28) RetrieveFirst% (kfile%, Qkey$, Qurecno&)
29) RetrieveLast% (kfile%, Qkey$, Qurecno&)
30) RetrieveNext% (kfile%, Qkey$, Qurecno&)
31) RetrievePrev% (kfile%, Qkey$, Qurecno&)
32) QBTreeVer% (ver%)
33) StatDataFile% (dfileno%, recl%, recs&, bfileno%)
34) StatKeyFile% (kfileno%, keyl%, keys&, bfileno%)
35) StoreKey% (kfile%, Qkey$, Qurecno&)
36) UnlockDataHeader% (dfileno%)
37) UnlockKeyFile% (kfileno%)
38) UnlockRecord% (dfileno%, recno&)
39) UpdateRecord% (dfile%, Qrec$)

INTERFACE DETAIL

1) AddKey(kfile,dfile,Qkey$)

After having found a key with GetFirst(), GetLast(), GetPrev(), GetEqual(), GetNext(), or created with AddKeyRecord(), insert the key Qkey$ into the key file, kfile (which most likely is an index file other than the one where the key was found, though it can be the same). Associate with this key the data record that was indexed by the last found/created key. This allows you have multiple indexes per data file. If an error code is returned you must *REACCESS* the original key with one of the Get() functions above before attempting to retry AddKey(), or error 206 (invalid data pointer) is returned. If error 201 (key already exists) is returned, and you want to allow duplicate keys, you need to add an enumerator to the key. See NON-UNIQUE KEYS below.

kfile INTEGER. Number that was used as fileno in

OpenKeyFile().

dfile INTEGER. Number that was used as fileno in OpenDataFile().

Qkey$ STRING. Key for which you want to add to the key file, kfile, and have it also point to the data record in dfile whose key was last found.

```
stat = GetEqual(0,0,Qkey$,Qrec$)
if stat = 0 then
   enum = 0
   enum$ = mki$(enum)    'make it a string
   'need to reverse enum$ bytes to sort low to high
   'use a function in your code such as SwapByte$(),etc
   ZKey$ = NewKey$ + right$(enum$,1) + left$(enum$,1)
   stat = AddKey(1,0,ZKey$)
else
   stop
endif
if stat = 201 then       'AddKey() key exists error
   'reset the internal pointer to a valid record
   stat = GetEqual(0,0,Qkey$,Qrec$)
   'look for the absolute last one (won't be found=200)
   stat = GetEqual(1,0,left$(Zkey$,len(ZKey$-2)+_
                  chr$(255)+chr$(255),Zrec$)
   stat = GetPrev(1,0,Zkey$,Zrec$)  'so get last valid one
   enum$ = right$(ZKey$,2)
   enum$ = right$(enum$,1)+left$(enum$,1)
   enum = cvi(enum$)     'make it an integer
   enum = enum + 1       'inc the enumerator
   enum$ = mki$(enum)
   ZKey$ = NewKey$ + right$(enum$,1) + left$(enum$,1)
   stat = AddKey(1,0,ZKey$)
```


2) AddKeyRecord(kfile,dfile,Qkey$,Qrec$)

Add the key, Qkey$, to the key file, kfile, and the data record, Qrec$, to the data file, dfile. Qkey$ must not already exist. QBTREE is case-sensitive so it is recommended that keys be made upper-case (or lower) unless there is a reason not to do this.

kfile INTEGER. Number that was used as fileno in OpenKeyFile().

dfile INTEGER. Number that was used as fileno in OpenDataFile().

Qkey$ STRING. Key to add to key file.

Qrec$ STRING. Data record to add to the data file indexed by Qkey$.

```
DEFINT A-Z
kfile = 0
dfile = 0
Qkey$ = acctid$ + acctxn$
Qrec$ = xaction$
stat = AddKeyRecord(kfile,dfile,Qkey$,Qrec$)
```


3) CloseDataFile(dfile)

Close the data file, dfile, releasing the data buffer allocated to it. This function is essential for proper termination in QBTREE. Release any active locks first.

```
dfile           INTEGER.  Number that was used as fileno in
                OpenDataFile().

dfile = 0
stat = CloseDataFile(dfile)
```

4) CloseKeyFile(kfile)

Close the key file, kfile. This is essential for proper termination in QBTREE. Header information is written only when the file is closed, or flushed. Release any active locks first.

kfile INTEGER. Number that was used as fileno in OpenKeyFile().

```
kfile = 0
stat = CloseKeyFile(kfile)
```

5) CreateDataFile(filename$,reclen)

Create a new data file. filename$ must not already exist.

filename$	STRING. Pathname of data file to create. Any valid DOS drive/path/filename can be used.
reclen	INTEGER. Length of record for this data file. Valid range is 3 to 2048 bytes. The upper limit can be changed to 32767 with the BTREE50.BAS source.

```
filename$ = "C:\HIST\AR89.DAT"
reclen = 128
stat = CreateDataFile(filename$,reclen)
```

6) CreateKeyFile(filename$,keylen)

Create a new key file. filename$ must not already exist.

filename$	STRING. Pathname of key file to create. Any valid DOS drive/path/filename can be used.
keylen	INTEGER. Length of key for this key file. Valid range is 1 to 64 bytes.

```
filename$ = "C:\HIST\AR89.KEY"
keylen = 16
stat = CreateKeyFile(filename$,keylen)
```

7) DeleteKey(kfile,Qkey$)

Delete the key, Qkey$, from kfile.

kfile	INTEGER.	Number that was used as fileno in OpenKeyFile().
Qkey$	STRING.	Key to remove from kfile.

```
kfile = 0
Qkey$ = acctid$ + acctxn$
stat = DeleteKey(kfile,Qkey$)
```

8) DeleteKeyRecord(kfile,dfile,Qkey$)

Delete the key, Qkey$, from file kfile and also delete the associated record from dfile.

This function deletes Qkey$ in kfile and also deletes the data record that Qkey$ had pointed to in dfile. If you have other indexes pointing to this data record, use DeleteKey() on those indexes before doing this function.

kfile	INTEGER. Number that was used as fileno in OpenKeyFile().
dfile	INTEGER. Number that was used as fileno in OpenDataFile().
Qkey$	STRING. Key you want to delete from the key file, kfile, and also delete its associated data record in dfile.

```
kfile = 0
dfile = 0
Qkey$ = acctid$ + acctxn$
stat = DeleteKeyRecord(kfile,dfile,Qkey$)
```

9) ExitQBTREE()

Close all QBTREE files and release memory used by the buffers. Using this routine is optional. Once this routine is called you must call InitQBTREE() before using any of the QBTREE routines again.

10) FlushDataFile(dfile,dup)

Updates the data header information and, if dup is non-zero, causes DOS to update the directory entry for dfile. This is done by writing the data header out, having DOS duplicate dfile's handle, then closing the duplicate handle. This does not have the overhead of having to re-open the data file. This also flushes DOS's internal buffers.

In LAN applications you should perform this function before releasing a data record lock, but dup must be equal to 0. If you force DOS to update the directory entry, UnlockRecord() returns error 229 - Lock already in force. Not flushing the buffers is not a problem in LAN applications since the SHARE.EXE will not allow 'local buffer' problems to occur. DOS 3.3+ programmers may want to investigate using DOS function 68h, instead.

dfile INTEGER. Number that was used as fileno in OpenDataFile().

dup INTEGER. Non-zero to force directory update.

```
dfile = 0
dup = -1          'dup must be 0 for LAN applications
```

```
stat = FlushDataFile(dfile,dup)
```

Note: Flushing buffers should be done whenever processing can handle the delay. Once a file has been flushed (and no further changes have been made), a power-outage can bring your system down with no ill-effects to the QBTREE file - all will be intact.

11) FlushKeyFile(kfile,dup)

Updates the key header information and, if dup is non-zero, causes DOS to update the directory entry for kfile. This is done by writing the key header out, having DOS duplicate file's handle, then closing the duplicate handle. This does not have the overhead of having to re-open the key file. This also flushes DOS's internal buffers.

In LAN applications you should perform this function before releasing a key file lock, but dup must be equal to 0. If you force DOS to update the directory entry, UnlockKeyFile() returns error 229 - Lock already in force. Not flushing the buffers is not a problem in LAN applications since the SHARE.EXE will not allow 'local buffer' problems to occur. DOS 3.3+ programmers may want to investigate using DOS function 68h, instead.

kfile INTEGER. Number that was used as fileno in

OpenKeyFile().

dup INTEGER. Non-zero to force directory update.

```
kfile = 0
dup = -1            'dup must be 0 for LAN applications
stat = FlushKeyFile(kfile,dup)
```


12) GetDirect(dfile,recno&,Qrec$)

Return the data in dfile at the logical record number, recno&. This is especially useful for reindexing data files, provided that the key is imbedded in the data record. It is possible to retrieve deleted records from the data file; records are not physically deleted but are marked 'available' for a new data record. See the Data Record Format below on how to plan for this. This function does not affect the index file's internal pointers.

dfile INTEGER. Number that was used as fileno in OpenDataFile().

```
recno&          LONG.     Record number to seek (1-based).

Qrec$           STRING.   Returned record data.

dfile = 0
recno& = 456&
stat = GetDirect(dfile,recno&,Qrec$)
if stat = 0 then
   print "Record#";recno&;" =";Qrec$
```


13) GetEqual(kfile,dfile,Qkey$,Qrec$)

Search for key, Qkey$, in kfile, and if found, get the data record from dfile, and place it in Qrec$. If not found, the internal pointers indicate where it would have been. By using GetNext(), GetPrev(), the next or previous ordered key and data

can be obtained.

kfile	INTEGER. Number that was used as fileno in OpenKeyFile().
dfile	INTEGER. Number that was used as fileno in OpenDataFile().
Qkey$	STRING. Key to search for in key file.
Qrec$	STRING. Returned data record associated with Qkey$.

```
kfile = 0
dfile = 0
Qkey$ = acctid$ + acctxn$
stat = GetEqual(kfile,dfile,Qkey$,Qrec$)
       '{Qrec$ is read}
if stat = 200 then
   stat = GetNext(kfile,dfile,Qkey$,Qrec$)
          '{Qkey$ & Qrec$ are read}
```

14) GetFirst(kfile,dfile,Qkey$,Qrec$)

Get the first ordered key in kfile, placing it in Qkey$, and place its data record from dfile into Qrec$.

kfile	INTEGER. Number that was used as fileno in OpenKeyFile().
dfile	INTEGER. Number that was used as fileno in OpenDataFile().
Qkey$	STRING. Returned first ordered key.
Qrec$	STRING. Returned data record associated with returned key, Qkey$.

```
kfile = 0
dfile = 0
stat = GetFirst(kfile,dfile,Qkey$,Qrec$)
```

15) GetLast(kfile,dfile,Qkey$,Qrec$)

Get the last ordered key in kfile, placing it in Qkey$, and place its data record from dfile into Qrec$.

kfile	INTEGER. Number that was used as fileno in OpenKeyFile().
dfile	INTEGER. Number that was used as fileno in OpenDataFile().
Qkey$	STRING. Returned last ordered key.
Qrec$	STRING. Returned data record associated with returned key, Qkey$.

```
kfile = 0
dfile = 0
stat = GetLast(kfile,dfile,Qkey$,Qrec$)
```

16) GetNext(kfile,dfile,Qkey$,Qrec$)

Get the next ordered key in kfile, placing it in Qkey$, and place its data record from dfile into Qrec$. This function allows for sequential inorder processing of the data file.

kfile	INTEGER. Number that was used as fileno in OpenKeyFile().
dfile	INTEGER. Number that was used as fileno in OpenDataFile().
Qkey$	STRING. Returned key which immediately follows the key found (or not found) in GetEqual() or GetFirst().
Qrec$	STRING. Returned data record associated with returned key, Qkey$.

```
kfile = 0
dfile = 0
Qkey$ = acctid$ + acctxn$
stat = GetNext(kfile,dfile,Qkey$,Qrec$)
```

17) GetPosition(kfile,recno&)

Return the logical record number of the current record pointed to by the last key FOUND in kfile. This function can be used to match keys from different key files that relate to a data file by recno& only. (Useful in deleting non-relational keys.)

kfile INTEGER. Number that was used as fileno in OpenKeyFile().

recno& LONG. Returned record number of the last read or written data record that was accessed using key file, kfile.

```
kfile = 0
dfile = 0
stat = AddKeyRecord(kfile,dfile,Qkey$,Qrec$)
if stat = 0 then
   stat = GetPosition(kfile,recno&)
   if stat = 0 then
      print "The data record number just used was";recno&
```

18) GetPrev(kfile,dfile,Qkey$,Qrec$)

Get the previous ordered key in kfile, placing it in Qkey$, and place its data record from dfile into Qrec$. This function allows for reverse sequential inorder processing of the data file.

kfile	INTEGER. Number that was used as fileno in OpenKeyFile().
dfile	INTEGER. Number that was used as fileno in OpenDataFile().
Qkey$	STRING. Returned key which immediately precedes the key found (or not found) in GetEqual() or GetLast().
Qrec$	STRING. Returned data record associated with returned key, Qkey$.

```
kfile = 0
dfile = 0
Qkey$ = acctid$ + acctxn$
stat = GetPrev(kfile,dfile,Qkey$,Qrec$)
```

19) InitQBTREE(MKF,MDF)

```
************************************************************
* THIS MUST BE CALLED BEFORE USING ANY OTHER QBTREE ROUTINE! *
************************************************************
```

Initialize the QBTREE file system and allocate the required buffers. This routine must be called before using any other QBTREE functions.

MKF	INTEGER. Maximum number of key files that you plan on using this session (0-based).
MDF	INTEGER. Maximum number of data files that you plan on using this session (0-based).

```
'{need 30 key and 10 data files this session}
stat = InitQBTREE(29,9)
'{you can now OpenKeyFile(file$,numkey) where numkey}
'{can range from 0 to 29 and OpenDataFile(file$,numdat)}
'{where numdat can range from 0 to 9.}

'{QBTREE 5.0 lets you open and use up to 250 files at one time}
'{using DOS 3+. However, the FILES= statement in CONFIG.SYS will}
'{be the limiting factor in how many you can actually use. The}
'{internal routine SFTFiles() can be used to determine what the}
'{FILES= is. Other useful internal routines are GetDOSVersion(),}
'{GetDiskInfo(), GetDefaultDrive(), and FileExists().}
```

20) LoadDataHeader(dfile)

Load the header information for the data file, dfile, from disk. This should be used in networking applications whenever a file has been released to other processes.

dfile INTEGER. Number that was used as fileno in OpenDataFile().

{See LoadKeyHeader() for an example.}

21) LoadKeyHeader(kfile)

Load the header information for the key file, kfile, from disk. This should be used in networking applications whenever a file has been released to other processes. Note that internal pointers will probably have changed. A GetEqual() to the last processed record will need to be used followed by a GetNext() for sequential inorder processing.

kfile INTEGER. Number that was used as fileno in OpenKeyFile().

```
kfile = 0
dfile = 0
stat = LockKeyFile(kfile)
if stat = 0 then
   stat = LockDataHeader(dfile)
   if stat = 0 then
```

```
      stat = LockRecord(dfile,0&)
      if stat = 0 then
         stat = LoadKeyHeader(kfile)
         stat = LoadDataHeader(dfile)
         '{have full access rights}
      else
         '{could not go all the way so back out}
         statU = UnlockDataHeader(dfile)
         statU = UnlockKeyFile(kfile)
      endif
   else
      statU = UnlockKeyFile(kfile)
   endif
endif
if stat = 0 then DoWork else DoRetry
```


22) LockDataHeader(dfile)

Lock the data header in dfile to the current process. Any other process requesting access to the header will be denied permission. This function should be used before adding or deleting a data record, though not needed for updating an existing record. To perform this DOS LAN function, the DOS program SHARE.EXE must be run.

dfile	INTEGER. Number that was used as fileno in OpenDataFile().

```
dfile = 0
stat = LockDataHeader(dfile)
if stat = 229 then
   print "Lock already in force"
elseif stat = 232 then
   print "SHARE.EXE needs to be run"
```


23) LockKeyFile(kfile)

Lock the key file, kfile, to the current process. Any other process requesting access to kfile will be denied permission. It is recommended that this be used in LAN applications prior to AddKeyRecord(), AddKey(), DeleteKeyRecord(), DeleteKey(). To

perform this DOS LAN function, the DOS program SHARE.EXE needs to be run.

kfile INTEGER. Number that was used as fileno in OpenKeyFile().

```
kfile = 0
stat = LockKeyFile(kfile)
if stat = 229 then
   print "Lock already in force"
elseif stat = 232 then
   print "SHARE.EXE has not been run"
```

24) LockRecord(dfile,recno&)

Lock a specific record, recno&, in dfile, to the current process. Any other process requesting access to recno& will be denied permission. If recno&=0& then all records in dfile will be locked (records 1 to EOF plus). To perform this DOS LAN function, the DOS program SHARE.EXE must be run. Note that you must use LockDataHeader() to lock the header record.

dfile	INTEGER.	Number that was used as fileno in OpenDataFile().
recno&	LONG.	Data record number in data file to lock.

```
dfile = 0
recno& = 1&         '{lock the first record in dfile}
stat = LockRecord(dfile,recno&)
if stat = 229 then
   print "Lock already in force"
elseif stat = 232 then
   print "SHARE.EXE needs to be run"
```

25) OpenDataFile(filename$,fileno)

Open an EXISTING data file and associate it with fileno.

The fileno is a number you choose between 0 to MDF where MDF is the number of data files used with InitQBTREE. This number is used to reference the data file in any later operation.

If version 3.0 or greater of DOS is detected by QBTREE then the open is performed with a READ/WRITE access and DENY NONE sharing. Older versions of DOS will have a normal open call (COMPATIBLE). What this means is that other processes may read from and write to the data file, filename$. See the LockRecord() and UnlockRecord() functions for details on locking the data file.

filename$ STRING. Pathname of existing data file.

fileno INTEGER. Number to associate the data file with for future operations. Valid range is 0 to MDF.

```
filename$ = "C:\HIST\AR89."
ARKEY = 0
ARDAT = 0
stat = OpenKeyFile(filename$+"KEY",ARKEY)
if stat = 0 then stat =  OpenDataFile(filename$+"DAT",ARDAT)
if stat <> 0 then DoErrorProc stat
```

26) OpenKeyFile(filename$,fileno)

Open an EXISTING key file and associate it with fileno.

The fileno is a number you choose between 0 to MKF, where MKF is the number of key files used with InitQBTREE. This number is used to reference the key file in any later operation.

If version 3.0 or greater of DOS is detected by QBTREE then the open is performed with a READ/WRITE access and DENY NONE sharing. Older versions of DOS will have a normal open call (COMPATIBLE). What this means is that other processes may read from and write to the key file, filename$. See the LockKeyFile() and UnlockKeyFile() functions for details on locking the key file.

filename$ STRING. Pathname of existing key file.

fileno INTEGER. Number to associate the key file with for future operations. Valid range is 0 to MKF.

```
filename$ = "C:\HIST\AR89."
ARKEY = 0
ARDAT = 0
stat = OpenKeyFile(filename$+"KEY",ARKEY)
if stat = 0 then stat =  OpenDataFile(filename$+"DAT",ARDAT)
if stat <> 0 then DoErrorProc stat
```

27) RetrieveEqual(kfile,Qkey$,Qurecno&)

Search for key, Qkey$, in kfile, and if found, retrieve the data record number and place it in Qurecno&. If not found, the internal pointers indicate where it would have been. By using RetrieveNext(), RetrievePrev(), the next or previous ordered key and data record number can be obtained.

kfile	INTEGER. Number that was used as fileno in OpenKeyFile().
Qkey$	STRING. Key for which to search in key file.
Qurecno&	LONG. Returned data record number associated with Qkey$.

```
kfile = 0
Qkey$ = acctid$ + acctxn$
stat = RetrieveEqual(kfile,Qkey$,Qurecno&)
       '{Qurecno& is retrieved}
if stat = 200 then
   stat = RetrieveNext(kfile,Qkey$,Qurecno&)
```

28) RetrieveFirst(kfile,Qkey$,Qurecno&)

Retrieve the first ordered key in kfile, placing it in Qkey$, and place its data record number into Qurecno&.

kfile	INTEGER. Number that was used as fileno in OpenKeyFile().
Qkey$	STRING. Returned first ordered key.
Qurecno&	LONG. Returned data record number associated with returned key, Qkey$.

```
kfile = 0
stat = RetrieveFirst(kfile,Qkey$,Qurecno&)
```

29) RetrieveLast(kfile,Qkey$,Qurecno&)

Retrieve the last ordered key in kfile, placing it in Qkey$, and place its data record number into Qurecno&.

kfile	INTEGER. Number that was used as fileno in OpenKeyFile().
Qkey$	STRING. Returned last ordered key.
Qurecno&	LONG. Returned data record number associated with returned key, Qkey$.

```
kfile = 0
stat = RetrieveLast(kfile,Qkey$,Qurecno&)
```

30) RetrieveNext(kfile,Qkey$,Qurecno&)

Retrieve the next ordered key in kfile, placing it in Qkey$, and place its data record number into Qurecno&. This function allows for sequential inorder processing of the data file.

kfile	INTEGER. Number that was used as fileno in OpenKeyFile().
Qkey$	STRING. Returned key which immediately follows the key found (or not found) in GetEqual() or GetFirst().
Qurecno&	LONG. Returned data record number associated with returned key, Qkey$.

```
kfile = 0
Qkey$ = acctid$ + acctxn$
stat = RetrieveNext(kfile,Qkey$,Qurecno&)
```

31) RetrievePrev(kfile,Qkey$,Qurecno&)

Retrieve the previous ordered key in kfile, placing it in Qkey$, and place its data record number into Qurecno&. This function allows for reverse sequential inorder processing of the data file.

kfile	INTEGER. Number that was used as fileno in OpenKeyFile().
Qkey$	STRING. Returned key which immediately precedes the key found (or not found) in RetrieveEqual() or RetrieveLast().
Qurecno&	LONG. Returned data record number associated with returned key, Qkey$.

kfile = 0

```
Qkey$ = acctid$ + acctxn$
stat = RetrievePrev(kfile,Qkey$,Qurecno&)
```


32) QBTreeVer(ver)

Return the version of the QBTREE access method (X100).

ver INTEGER. Returned version number * 100.

```
stat = QBTreeVer(ver)
print "QBTREE version";ver\100
```

33) StatDataFile(dfile,reclen,recs&,bfileno)

Return information about the data file, dfile.

dfile INTEGER. Number that was used as fileno in OpenDataFile().

```
reclen          INTEGER.  Returned length of records for dfile.

recs&           LONG.     Returned number of records in dfile.

bfileno         INTEGER.  Returned BASIC's file number for dfile.

dfile = 0
stat = StatDataFile(dfile,reclen,recs&,bfileno)
```

34) StatKeyFile(kfile,keylen,keys&,bfileno)

Return information about the key file, kfile.

kfile	INTEGER.	Number that was used as fileno in OpenKeyFile().
keylen	INTEGER.	Returned length of keys for kfile.
keys&	LONG.	Returned number of keys in kfile.
bfileno	INTEGER.	Returned BASIC's file number for kfile.

```
kfile = 0
stat = StatKeyFile(kfile,keylen,keys&,bfileno)
```

35) StoreKey(kfile,Qkey$,Qurecno&)

Store the key, Qkey$, to the key file, kfile, and associate with the key the data record number, Qurecno&. Qkey$ must not already exist. QBTREE is case-sensitive so it is recommended that keys be made upper-case (or lower) unless there is a reason not to do this.

kfile	INTEGER. Number that was used as fileno in OpenKeyFile().
Qkey$	STRING. Key to store to key file.
Qurecno&	LONG. Data record number to store that is associated with Qkey$.

```
kfile = 0
Qkey$ = acctid$ + acctxn$
Qurecno& = NextFreeRecordNumber&(Arg1)
stat = StoreKey(kfile,Qkey$,Qurecno&)
```

36) UnlockDataHeader(dfile)

Unlock the data header in dfile. Commit the file header with a FlushDataFile() first. To perform this DOS LAN function, the DOS program SHARE.EXE must be installed.

dfile	INTEGER. Number that was used as fileno in OpenDataFile().

```
dfile = 0
stat = UnlockDataHeader(dfile)
if stat = 229 then
   print "Lock already in force/cannot DUP LAN..."
 elseif stat = 232 then
   print "SHARE.EXE needs to be run"
```

37) UnlockKeyFile(kfile)

Unlock the key file, kfile. You should do this as soon as you've committed the file with FlushKeyFile() after a AddKeyRecord(), AddKey(), DeleteKeyRecord(), DeleteKey(). To perform this DOS LAN function, the DOS program SHARE.EXE needs to be run.

kfile	INTEGER. Number that was used as fileno in OpenKeyFile().

```
kfile = 0
stat = UnlockKeyFile(kfile)
if stat = 232 then
   print "SHARE.EXE has not been run"
```

38) UnlockRecord(dfile,recno&)

Unlock a specific record, recno&, in dfile. Once the record that was locked is no longer needed, you should unlock it. If a process terminates without releasing active locks on a file, the result is undefined. If recno&=0& then all records in dfile will be unlocked (records 1 to EOF plus). To perform this DOS LAN function, the DOS program SHARE.EXE must be installed.

dfile	INTEGER. Number that was used as fileno in OpenDataFile().
recno&	LONG. Data record number in data file to unlock.

```
dfile = 0
recno& = 1&         '{unlock the first record in dfile}
stat = UnlockRecord(dfile,recno&)
if stat = 232 then
   print "SHARE.EXE has not been run"
```

39) UpdateRecord(dfile,Qrec$)

Overwrite the current data record in dfile with Qrec$. This allows you to update the contents of a record in a data file. Do not use this if you change an imbedded key in this record. If you want to change a key, first AddKeyRecord() the new data and if a-ok, delete the old one using DeleteKeyRecord().

dfile	INTEGER. Number that was used as fileno in OpenDataFile().
Qrec$	STRING. Data to use in replacing the previous data in dfile.

```
kfile = 0
dfile = 0
Qkey$ = acctid$
stat = GetEqual(kfile,dfile,Qkey$,Qrec$)
if stat = 0 then
   Qrec$ = newQrec$
   stat = UpdateRecord(dfile,Qrec$)
```

ERROR CODES

QSAM generated error codes (200-218):

200 Key not found
- the key is not in the index file. This will occur during a GetEqual(), DeleteKey(), DeleteKeyRecord(), RetrieveKey().

201 Key already exists
- duplicate keys are not allowed in QBTREE. This will occur during AddKeyRecord(), AddKey(), StoreKey(). If this error occurs with AddKey(), you must re-establish a valid data pointer before trying again.

202 End of file
- GetNext(), RetrieveNext() has reached the end of file.

203 Top of file
- GetPrev(), RetrievePrev() has reached the top of file.

204 Empty file
- there are no keys in the index file. This will occur when you attempt to perform functions that would require a non-empty file.

205 Disk full
- if the key file drive and the data file drive are the same a minimum of 2 clusters are needed for the key file plus the clusters required to completely hold the data record before you can AddKeyRecord(), AddKey(), or StoreKey(). If they're separate drives, the key file needs at least 2 clusters free on its disk and the data file disk as many clusters as needed to hold the data record. This will also occur during CreateKeyFile(), CreateDataFile().

206 Data pointer invalid
- a valid QBTREE key access, either GetEqual(), GetNext(), GetPrev(), GetFirst(), GetLast() must occur right before a data record can be UpdateRecord() or a key can be AddKey().

207-209 reserved

210 Stack overflow (10 levels)
- the internal tracking stack exceeded capacity. This should never occur. If it does it means that the index file is corrupt.

211 Function not implemented
- this won't happen unless you call the low-level driver (QSAM).

212-218 reserved

BTREE50.BAS generated error codes:

219 File number invalid
- the file number for the keyfile or datafile is greater than the highest allocated in InitQBTREE().

220 Data record length invalid
- length must be from 3-2048 bytes when creating a data file.

221 Key length invalid
- length must be from 1-64 bytes when creating a key file.

222 File not open
- the fileno for a key or data file operation is not opened.

223 Invalid null key assignment

-a null key cannot be used. Null is defined in QBTREE as either ASCII 0 or ASCII 255. This will occur in AddKeyRecord(), AddKey(), DeleteKeyRecord(), DeleteKey(), GetEqual(), RetrieveEqual(). This pertains only to the first character of the key. This will also occur if you reach top of file and attempt to delete.

224 Invalid record number
- a record number less than 1 or a record number greater than the maximum records possible was used in GetDirect().

225 No handles available
- no DOS handles are available to flush files. Use a greater value in config.sys FILES= (FILES=20). Also occurs in InitQBTREE() if you try to open more than 250 files with DOS 3+ or more than 15 files with DOS 2.

226 Invalid drive specifier
- the drive specified in CreateKeyFile(), CreateDataFile() is not a valid DOS drive.

227 reserved

228 File not QBTREE
- the filename in OpenKeyFile() or OpenDataFile() is not recognized as a QBTREE file.

229 Lock already in force
- the lock requested cannot be made because an existing lock of the header, record or file is in force. See FlushFile().

230 File already exists
- the CreateKeyFile() or CreateDataFile() filename already exists. Delete the file if you really want to use the name.

231 File not found
- the filename$ specified in OpenKeyFile() or OpenDataFile() does not exist or is not valid.

232 General lock failure
- usually means that SHARE.EXE was not run before using the network functions.

233 Init not active
- using QBTREE routines without having called InitQBTREE().

234 Init already active
     - calling InitQBTREE() more than once.

235-255 reserved

TECHNICAL SPECIFICATIONS

Key length   : 1-64 bytes (ASCII sort), constant
Record length : 3-2048 bytes, constant (to 32767 with source)
Node size   : 512 bytes

```
Keys per node  : 7-84 keys, (512-3)\(key length+5)
Max keys/file  : 5.5 million keys (65535 nodes)
Max recs/file  : 16 million records
Max key files  : 250 (total combined files <= 250 DOS 3+)
Max data files : 250 (total combined files <= 15 DOS 2)
```

Key file header format (first sector (512 bytes) of key file):

```
filetype        char    ;  0 file type, "*", ASCII 42
rootnode        int     ;  1 sector in file of root node
nokeyslo        int     ;  3 number of keys (low word)
nokeyshi        byte    ;  5 number of keys (high byte)
keyavsec        int     ;  6 key node available list
nxkeysec        int     ;  8 next free node/sector
keylen          byte    ; 10 length of key
maxkeys         byte    ; 11 maximum keys per node
internal        any     ; 12 - 58 internal use
reserved        any     ; 59 - 511
```

Data file header format (first 32 bytes of data file):

```
filetype        char    ;  0 file type, "S"
reclen          int     ;  1 length of record
norecslo        int     ;  3 number of records (low word)
norecshi        byte    ;  5 number of records (high byte)
datavlo         int     ;  6 data available list (low word)
datavhi         byte    ;  8 data available list (high byte)
nxdalo          int     ;  9 next data record avail (low word)
nxdahi          byte    ; 11 next data record avail (high byte)
internal        any     ; 12 - 31 reserved
```

Some structures above use a 24-bit value. These use 3 bytes (word+byte) to store information. For example, the value of the datavlo/hi structure in the data file header is:

```
DataAvailRec& = 1& * datavlo + (datavhi * 65536)
```

Key record format:

There is an internal first key with a null value that logically marks the top of the key file.

Beginning each node (sector) is a count key byte. This is the count of valid keys on that sector. Then for each key is a 16-bit previous node pointer, the key itself, the 24-bit data record pointer for that key, and a 16-bit next node pointer (node pointers are zero at the leaf nodes).

```
02  00 00  000000  00 00 00  00 00  KEY001  01 00 00  00 00 ...
 1.   2.     3.       4.       5.     6.       7.       8.   9.
```

1. Key count for that node
2. Node back pointer (for non-leaf nodes)
3. The internal null key (in least-valued node)
4. The 24-bit data record pointer (LB/HB/XB)
5. Forward ptr/back ptr (for non-leaf nodes)
6. First logical key
7. Its data pointer (record number in data file)
8. Its forward pointer (for non-leaf nodes)
9. repeat 6 to 8 for each key on node

QBTREE makes some allowances on the strict B-TREE and ISAM definitions. For all practical purposes, consider QBTREE superior to the text-book implementations of these.

Data record format:

Straight data after the header. Logical record #1 follows the header. Records that are deleted have the first 3 bytes used as a link in the records available list. datavlo/hi in the header point to the last deleted record (or 0 0 0 if none). The first 3 bytes of THAT deleted record point to the previously deleted record, and so on. When 3 zero bytes form the pointer, THAT record is the last in the list of deletes that can be reused. Any additional records added will then will expand the file.

By reserving the first 3 bytes of the data record it would be easy to reindex a datafile. Set the bytes to all ASCII 255. You then use GetDirect(), check to see if the 3 bytes are still 255s, and if so, AddKeyRecord() to a new key and data file. If those 3 bytes are not 255s, you know that that record is deleted.

A more difficult method is to track the datavlo/hi singly-linked list to scan for deleted records. It's more complex but doesn't require you to reserve 3 bytes. Briefly, start by reading the FLUSHED data header to get the datavlo/hi structure. This structure points to the last deleted record, or, if 0 there are no deleted records available in the file. Assuming it is not 0, (let's say it's record 5), flag that this record (5) is deleted. Use GetDirect() to read record 5. If the first 3 bytes in record 5 are not 0, continue the process, flagging each record that is deleted. When you reach the last deleted record, the first 3 bytes will be 0. You now know what records to skip when you scan through the data file using GetDirect() and AddKeyRecord().

By using GetDirect() with StoreKey() it is possible to reindex a data file without having to reconstruct the data file. The benefit of removing deleted data records is lost, however.

COMPILER INFORMATION and RESOURCE ALLOCATION

QBTREE5N.LIB's I/O and interface portion was written and compiled with QuickBASIC 4.00b with the /O option only. If you are using QuickBASIC 4.00b+, which has a better error handling algorithm than 4.0, you can trap I/O errors with QB even though the BTREE50.BAS module was compiled without the /E or /X options. However, version 4.00 will dump the program to DOS on errors. So if your are using version 4.00 you need to perform common I/O checks for fatal errors such as checking to see that the drive is ready etc., before using QBTREE functions with QB/BC 4.0.

To regenerate QBTREE for your compiler version:
     - DO NOT use your only copy to make these changes -

1. Make any needed changes to the source file, BTREE50.BAS.
2. Recompile BTREE50.BAS, update the QBTREE5x.LIB file, and create the new QLB for the enviornment:

for QB4.xx:

```
C>bc BTREE50 /O;
C>lib QBTREE5N.LIB -+BTREE50.OBJ;
C>link /QU QBTREE5N.LIB,QB.QLB,nul,BQLB4x.LIB
```

for BASIC 7.xx (/Fs is required to make a QLB):

```
C>bc BTREE50 /O/Ot/Fs;
C>lib QBTREE5F.LIB -+BTREE50.OBJ
C>link /QU QBTREE5F.LIB,QBX.QLB,nul,QBXQLB.LIB;
```

Storage for the data buffers and headers are allocated at run-time in far memory. This means that very little string space is used by QBTREE. Data allocation is as follows (arrays are stored in far memory):

2 integer arrays (0 to MKF) + (0 to MDF) for file handles.
1 key hdr array (0 to MKF) for key file headers, ea 62 bytes.
1 data hdr array (0 to MDF) for data file headers, ea 32 bytes.
1 key node array (0 to MKF) for node buffers, ea 512 bytes.
1 data buffer array (0 to MDF) for data record I/O (size allocated is 2048 bytes each but can be changed with source).
1 long integer array (0 to MKF) to track current record numbers.
2 integer arrays (0 to MKF) + (0 to MDF) to track file drives.
1 type variable for interfacing with QBTREE low-level, 34 bytes.
... and other miscellaneous variables totalling about 2500 bytes.

USING TYPED VARIABLES

QBTREE uses a simple variable, namely a var-len string, to get and put data in the data file. To use QB TYPEd variables, you may want to use this method. Setup your TYPEd variable, then allocate enough space in a var-len string to copy to and from it.

```
TYPE dataTYPE
 TAG AS STRING * 4
 SSN AS STRING * 9
 AGE AS INTEGER
 XXX AS STRING * 1
END TYPE '16

DIM SSNage AS dataTYPE

SSNage.TAG = ""
SSNage.SSN = ssn$
SSNage.AGE = age

DIM SHARED work$
work$ = SPACE$(LEN(SSNage))

FromSeg = VARSEG(SSNage)
FromOff = VARPTR(SSNage)
ToSeg = VARSEG(work$)     '{we want the offset to the string's}
ToOff = SADD(work$)       '{address, not the string descriptor's}
count = LEN(work$)
MemCopy FromSeg,FromOff,ToSeg,ToOff,count

'{You now have the TYPEd variable data in a var-len string that}
'{can be used as the Qrec$. To put Qrec$ in the typed variable,}
'{reverse the From/To assignments:}

Qkey$ = mid$(work$,2,9)
Qrec$ = work$
stat = AddKeyRecord(0,0,Qkey$,Qrec$)

stat = GetEqual(0,0,Qkey$,Qrec$)
FromSeg = VARSEG(Qrec$)
FromOff = SADD(Qrec$)
```

```
ToSeg = VARSEG(SSNage)
ToOff = VARPTR(SSNage)
count = len(SSNage)
MemCopy FromSeg,FromOff,ToSeg,ToOff,count
print SSNage.SSN; SSNage.age

'see XAPP1.BAS for more on using TYPEd variables
'MemCopy is included in the QBTREE5x.LIB and QBTREE50.BI. The
'direction of copy is always forward.
```


NON-UNIQUE KEYS

QBTREE doesn't allow duplicate keys (this may be an inconvenience in some data base programming) but this can easily be worked around by adding to each key an additional two bytes. These bytes would act to differentiate up to 65536 'identical' keys.

For example, if the needed key length is 16 bytes, make it 18. Reserve bytes 17 and 18 so that you can enumerate identical keys. When adding a key that doesn't already exist, set bytes 17 and 18 to ASCII 0. If the key already exists (error 201), use the highest possible enumerator (ASCII 255,255) for the key. Do a GetEqual(). Since that key will not be found (error 200), do a GetPrev(). This retrieves the very last inorder record for the key. Add 1 to the found enumerator value and use that as the enumerator for the next key. Add the key.

Note: For proper sorting, you should have the most significant byte of the enumerator first and the least significant last. See the AddKey() code example for more.

DISK CACHING

QBTREE is compatible with disk caches and will benefit from their use. For example, DEMOQBT.EXE was run in the following environments:

Test: Store 32000 (integer loop) 9-byte random keys in file KEYONE.I using StoreKey(), skipping test II. This creates a key file 656,384 bytes in size.

```
C>DEMOQBT /K=32000 /2=0
```

Without caching:

```
     Add 32000 random 9-byte keys: 67 min (8/sec)
     Delete every other (16000):   39 min (7/sec)
```

Using Super PC-KWIK disk cache:

```
     Add 32000 random 9-byte keys: 17 min (31/sec)
     Delete every other (16000):   11 min (24/sec)
```

Not all disk caches will show the benefit obtained by PC-KWIK when adding keys.

Read times in both cases seemed to be more limited by how fast the print and scrolling could be done, in other words, retrieval is very fast!

********** QBTREE REGISTRATION FORM ********** 501 ***

Name of registrant: _____

Amount paid: $ _____ ($45 ea. US funds on US bank)
(Texas address add 8.25%, foreign orders add $5.00)

If you want me to send the interface source code, fill out the non-release form below.

Comments: _____

_____

Contact: _____ Phone:( )____-_____

Would you like to be added to our mailing list? _____

Mail to: _____

_____

_____

Send this page with payment to:

Cornel Huth
ATTN: QBTREE 5.01 REGISTRATION
6402 Ingram Rd.
San Antonio, TX 78238 USA

*** NON-RELEASE FORM for QBTREE 5.01 INTERFACE SOURCE CODE ****

I, _____, hereby agree not to distribute, nor claim ownership of, QBTREE source code or any modification to the QBTREE interface, and not remove the copyright notice of Cornel Huth from any part of QBTREE.

DATE:_____

_____

(If you would like the interface source code in QB 4.00+/PDS 7+ print your name in the first blank, date the second, and sign the third.)