

A B-Tree Access Method
for QuickBASIC/PDS Programmers

QBTtree v5.50

31-July-91
(C) 1989-1991 Cornel Huth

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, print and then fill out the !ORDER.FRM on the distribution disk and send it along with full payment to:

Cornel Huth
ATTN: QBTree 5.50 REGISTRATION
6402 Ingram Rd.
San Antonio, TX 78238

All BASIC source code for the QBTree interface will be sent upon receipt of the completed registration form.

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 since much has changed from previous versions, especially: all network routines, OpenFile(), and FileExists().

--This file has been mechanically reproduced from the DTP source of the documentation with manual touchup. Page numbers, table of contents, and other facilitators have not been incorporated into this reproduction.

OVERVIEW

QBTree is a low-level file manager for Microsoft QuickBASIC and BASIC 7 Professional Development System compilers. It is based on the B-Tree sorting method and provides fast and efficient database functions for the QB/PDS programmer.

Some features of QBTree are:

- 1) Maintain up to 250 key and data files at one time.
- 2) Find any key and data record quickly - any key in a million in at most 5 disk accesses.
- 3) Forward and reverse sequential inorder access to the data file using just one key file (ISAM).
- 4) Automatic data file maintenance where deleted records are made available for reuse.
- 5) Automatic key file balancing.
- 6) DOS-compatible file and record locking network support.
- 7) Routines to support key file maintenance with any data file format the programmer specifies.

QBTree has evolved significantly over the past few years. Network support, Dbase compatibility (in the QBXDBF package, available with registration), and fast, compact B-Tree indexing all combined in one package and at one low price. Compare this to other packages at \$200-\$600+. But best of all, QBTree was designed specifically for QuickBASIC and PDS compilers. There is no separate TSR to be loaded nor is there a cryptic language interface. You'll find QBTree easy to use, just like QuickBASIC, with only a few of the routines needed to start creating database applications. From there, as you gain experience, you'll find that even sophisticated database applications will become easy to design and quick to code.

INSTALLATION --Shareware try-before-you-buy version.

The following files should be found in this shareware package:

!ORDER.FRM	Order form -- this form must be used when ordering this documentation suitable for printing
QBTREE55.PRN	QLB for QuickBASIC 4.5
QBTREE45.QLB	QLB for QuickBASIC 4.00b
QBTREE4B.QLB	QLB for BASIC PDS 7.1
QBTREE71.QLB	declares
QBTREE.BI	example QBTree source file
XBTREE1.BAS	example QBtree source file
XBTREE2.BAS	data file used by XBTREE1.BAS and XDBF1.EXE
XDATA1.DAT	information of XDBF1.EXE
XDBF1.TXT	example QBXDBF (dBASE) executable file
XDBF1.EXE	

To evaluate the QBTree package simply startup your environment with the appropriate QBTree QLB:

C>QB /!QBTREE4B or C>QB /!QBTREE45 or C>QB /!QBTREE71

All functionality of the registered versions is found in these QLBs. However, you cannot create executable programs without registering. The QuickBASIC versions of QBTree were compiled with their respective compiler using /o as the the compiler switch. The BASIC PDS version was compiled using /o/Fs as the compiler switches. If you are still using the buggy QuickBASIC 4.00 (original release) send \$5.00 and I'll ship a version 4.00 QLB to you.

Sample Go:

Start QB with the appropriate QLB. Load the XBTREE1.BAS source file. Press shift-F5. That's it. Be sure you have 'QBTREE.BI' accessible and XDATA1.DAT in the current directory.

Remember, registration is required for continued use. In addition, registering QBTree allows you to order QBXDBF for only \$5.00 more.

QBXDBF is similar to QBTree but uses the dBASE .DBF file format. There are many features built-in to QBXDBF that are not available in QBTree. For example, keys are entered once when the key file is created, e.g., kx\$=UPPER(SUBSTR(LASTNAME,1,1))+SUBSTR(SSN,6,4). Duplicate keys are handled automatically. Up to 1023 fields per record. A very fast reindex module (12,000 recs/min). Much more.

In short, whereas QBTree is a low-level file manager, QBXDBF is a mid-level file manager. It's not as versatile as QBTree (after all, QBXDBF was written in QBTree) but it is well suited to the task of database programming.

Please, use the !ORDER.FRM to place your order.

INSTALLATION -Registered version.

There are 3 libraries distributed with QBTree. QBTREE.LIB is for QuickBASIC compilers, QBTREEF.LIB and QBREEN.LIB are for BASIC 7 compilers. Your source code needs to REM \$INCLUDE: 'QBTREE.BI'.

To install for QuickBASIC: The BASIC interface portion of QBTREE.LIB was compiled with BC 4.50 (/O) and is intended for use with QuickBASIC 4.xx.

To use QBTree in the QB environment create a QLB:

```
C>link /qu QBTREE.LIB<+other.lib>,qb.qlb,nul,bqlb45
```

When you create an EXE file of your program, add QBTREE.LIB to the library prompt from LINK.

If you have a version of QB other than 4.5 you should recompile the BASIC source, update the library, and create a QLB:

```
C>ren BTREEZ.BAS BTREE.BAS
C>bc BTREE.BAS /O;
C>lib QBTREE -+BTREE;
C>link /qu QBTREE.LIB,qb.qlb,nul,bqlb4x
  (where x=0 for 4.00, x=1 for 4.00b)
```

To install for BASIC 7/PDS: The BASIC interface portion of QBTREEF.LIB was compiled with BC 7.10 (/O/Fs) and is intended for use with BASIC 7.xx and QBX using far-string support. QBREEN.LIB (/O/Ot) is intended for use with BASIC 7.xx using near-string support.

To use QBTree in the QBX environment create a QLB:

```
C>link /qu QBTREEF.LIB<+other.lib>,qbx.qlb,nul,qbxqlb;
```

When you create an EXE file of your program, add QBTREEF.LIB (or QBREEN.LIB) to the library prompt from LINK.

If you have a version of BASIC other than 7.1 you should recompile the BASIC source, update the libraries, and create a QLB:

```
C>ren BTREEZ.BAS BTREEF.BAS
C>bc BTREEF.BAS /O/Fs;
C>lib QBTREEF -+BTREEF;

C>ren BTREEF.BAS BTREEN.BAS
C>bc BTREEN.BAS /O/Ot;
C>lib QBREEN -+BTREEN;

C>link /qu QBTREEF.LIB,qbx.qlb,nul,qbxqlb;
```

FUNCTION SUMMARY

All QBTree routines are FUNCTIONS and return an integer code which is detailed in APPENDIX D. ERROR CODES.

- 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) FreeDataFile()
- 13) FreeKeyFile()
- 14) GetDirect(dfileno, recno&, Qrec\$)
- 15) GetEqual(kfile, dfile, Qkey\$, Qrec\$)
- 16) GetFirst(kfile, dfile, Qkey\$, Qrec\$)
- 17) GetGT(kfile, dfile, Qkey\$, Qrec\$)
- 18) GetGTE(kfile, dfile, Qkey\$, Qrec\$)
- 19) GetLast(kfile, dfile, Qkey\$, Qrec\$)
- 20) GetLT(kfile, dfile, Qkey\$, Qrec\$)
- 21) GetLTE(kfile, dfile, Qkey\$, Qrec\$)
- 22) GetNext(kfile, dfile, Qkey\$, Qrec\$)
- 23) GetPosition(kfileno, recno&)
- 24) GetPrev(kfile, dfile, Qkey\$, Qrec\$)
- 25) InitQBTree(MKF, MDF)
- 26) LockDataHeader(dfileno)
- 27) LockKeyFile(kfileno)
- 28) LockRecord(dfileno, recno&)
- 29) OpenDataFile(filename\$, dfileno, asmode)
- 30) OpenKeyFile(filename\$, kfileno, asmode)
- 31) RetrieveEqual(kfile, Qkey\$, Qrecno&)
- 32) RetrieveFirst(kfile, Qkey\$, Qrecno&)
- 33) RetrieveGT(kfile, dfile, Qkey\$, Qrecno&)
- 34) RetrieveGTE(kfile, dfile, Qkey\$, Qrecno&)
- 35) RetrieveLast(kfile, Qkey\$, Qrecno&)
- 36) RetrieveLT(kfile, dfile, Qkey\$, Qrecno&)
- 37) RetrieveLTE(kfile, dfile, Qkey\$, Qrecno&)
- 38) RetrieveNext(kfile, Qkey\$, Qrecno&)
- 39) RetrievePrev(kfile, Qkey\$, Qrecno&)
- 40) QBTreeVer(ver)
- 41) StatDataFile(dfileno, recl, recs&, bfileno)
- 42) StatKeyFile(kfileno, keyl, keys&, bfileno)
- 43) StoreKey(kfile, Qkey\$, Qrecno&)
- 44) UnlockDataHeader(dfileno)
- 45) UnlockKeyFile(kfileno)
- 46) UnlockRecord(dfileno, recno&)
- 47) UpdateRecord(dfile, Qrec\$)

AddKey

TYPE FUNCTION

SYNTAX stat=AddKey(kfile,dfile,Qkey\$)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
dfile - INTEGER.Number used as fileno in OpenDataFile().
Qkey\$ - STRING.Key to add to kfile. This key will index
the current data record in dfile.

USE Add the key to kfile using the current data record as
this key's data record. This lets you have more than one
index file per data file.
For instance, you would like to index a data file on
both name and social security number: First, use
AddKeyRecord() to insert the name key in the name key
file and add the data record to the data file.
Immediately after, use AddKey() to insert the SSN key in
the SSN key file. Both the name key and SSN key point to
the same data record. Any individual data record can now
be found by using either the name key file or the SSN key
file.

RULES The key must be unique.
Qkey\$ must not begin with an ASCII 0 or 255.
If an error code is returned by AddKey() you must reset
the current data record before retrying the function.
This is because any error causes QBTree to invalidate the
current data record pointer.
For instance, if the key you are adding already exists
in the key file, error 201 is returned. After making the
key unique, you must reset the current data record by
REACCESSING it using GetEqual() with the appropriate key,
otherwise error 206 is returned.

NOTES QBTree is case-sensative. Convert all added keys (not
data records) to the same case unless there is a reason
not to.

RETURN Errors 201, 206, 219, 222, 223, and DOS.

EXAMPLE stat = AddKeyRecord(0,0,Qname\$,Qdatarec\$)
IF stat = 0 THEN
 stat = AddKey(1,0,QSSN\$)
 IF stat = 201 THEN
 'see Appendix C. Non-unique Keys for
 'method to create unique keys
 UQSSN\$ = DoMakeKeyUnique\$(1,0,QSSN\$)
 'reaccess data record to reset pointer
 stat2 = GetEqual(0,0,Qname\$,nul\$)
 'add the 'unique' key
 stat = AddKey(1,0,UQSSN\$)

AddKeyRecord

TYPE FUNCTION

SYNTAX stat=AddKeyRecord(kfile,dfile,Qkey\$,Qrec\$)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
dfile - INTEGER.Number used as fileno in OpenDataFile().
Qkey\$ - STRING.Key to add to kfile. This key will index
Qrec\$ in the data file.
Qrec\$ - STRING.Data record to add to dfile.

USE Add the key to kfile and add the data record to dfile.
Once added, using GetEqual() with this key returns this
data record.

RULES The key must be unique.
Qkey\$ must not begin with an ASCII 0 or 255.

NOTES QBTree is case-sensative. Convert all added keys (not
data records) to the same case unless there is a reason
not to.

RETURN Errors 201, 219, 222, 223, and DOS.

EXAMPLE Qdatarec\$ = "KEYPART DATARECORDPART"
Qname\$ = LEFT\$(Qdatarec\$,8)
stat = AddKeyRecord(0,0,Qname\$,Qdatarec\$)
IF stat = 201 THEN
 'see Appendix C. Non-unique Keys for
 'method to create unique keys
 UQname\$ = DoMakeKeyUnique\$(0,0,Qname\$)
 'add the 'unique' key
 stat = AddKeyRecord(0,0,UQname\$,Qdatarec\$)

CloseDataFile

TYPE FUNCTION

SYNTAX stat=CloseDataFile(dfile)

PARAMETERS dfile - INTEGER.Number used as fileno in OpenDataFile().

USE Write the data header from memory to disk and release the handle used by dfile. Tell DOS to close the file and update the directory entry.

RULES None.

NOTES See FlushDataFile.

RETURN Errors 219, 222, and DOS.

EXAMPLE 'done with data file 0, close it
 stat = CloseDataFile(0)

CloseKeyFile

TYPE FUNCTION

SYNTAX stat=CloseKeyFile(kfile)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().

USE Write the key header from memory to disk and release the handle used by kfile. Tell DOS to close the file and update the directory entry.

RULES None.

NOTES See FlushKeyFile.

RETURN Errors 219, 222, and DOS.

EXAMPLE 'done with key file 0, close it
stat = CloseKeyFile(0)

CreateDataFile

TYPE FUNCTION

SYNTAX stat = CreateDataFile(filename\$,reclen)

PARAMETERS filename\$ - STRING.Pathname of data file to create.
reclen - INTEGER.Length of data record.

USE Create a new data file with the given record length.
This does not leave the file open - you must
OpenDataFile() to use it.

RULES filename\$ must be <= 64 characters; reclen must be >= 3
and <= 2048.

NOTES Both filename\$ and reclen maximum lengths can be
changed, see the BASIC source file for information on
doing this.

RETURN Errors 220, 230, 233, and DOS.

EXAMPLE filename\$ = "F:\HIST\AR91.DAT"
reclen = 128
stat = CreateDataFile(filename\$,reclen)

CreateKeyFile

TYPE FUNCTION

SYNTAX stat = CreateKeyFile(filename\$,keylen)

PARAMETERS filename\$ - STRING.Pathname of key file to create.
 keylen - INTEGER.Length of key.

USE Create a new key file with the given key length. This does not leave the file open - you must OpenKeyFile() to use it.

RULES filename\$ must be <= 64 characters; keylen must be <= 64 bytes.

NOTES filename\$ maximum length can be changed, see the BASIC source file for information on doing this.

RETURN Errors 221, 230, 233, and DOS.

EXAMPLE filename\$ = "F:\HIST\AR91ANUM.KEY"
 keylen = 17
 stat = CreateKeyFile(filename\$,keylen)

DeleteKey

TYPE FUNCTION

SYNTAX stat = DeleteKey(kfile,Qkey\$)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
Qkey\$ - STRING.Key to remove from kfile.

USE Remove the key Qkey\$ from kfile.

RULES Qkey\$ must not begin with an ASCII 0 or 255.

NOTES Let's say you have a customer data file with two key files used to index it, one by name, the other by customer code, and you want to remove a customer from this database. You would use DeleteKey() on one of the key files, and DeleteKeyRecord() on the remaining key file (and data file). If you use DeleteKey() on both key files you would be unable to remove the data record.

RETURN Errors 200, 204, 219, 222, 223, and DOS.

EXAMPLE fileK0\$ = "C:\CUSTNAME.KEY"
 fileK1\$ = "C:\CUSTCODE.KEY"
 fileD0\$ = "C:\CUSTDATA.DAT"
 '[code to open files deleted]
 DelName\$ = "Sioux," + "Leslie"
 DelCode\$ = "100500"
 stat = DeleteKey(0,DelName\$)
 IF stat = 0 THEN
 stat = DeleteKeyRecord(1,0,DelCode\$)

DeleteKeyRecord

TYPE FUNCTION

SYNTAX stat = DeleteKeyRecord(kfile,dfile,Qkey\$)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
dfile - INTEGER.Number used as fileno in OpenDataFile().
Qkey\$ - STRING.Key to remove from kfile.

USE Remove the key Qkey\$ from kfile and also remove the data record that Qkey\$ indexes from dfile.

RULES Qkey\$ must not begin with an ASCII 0 or 255.

NOTES Let's say you have a customer data file with two key files used to index it, one by name, the other by customer code, and would like to remove a customer from this database. You would use DeleteKey() on one of the key files, and DeleteKeyRecord() on the remaining key file (and data file).

RETURN Errors 200, 204, 219, 222, 223, and DOS.

EXAMPLE fileK0\$ = "C:\CUSTNAME.KEY"
fileK1\$ = "C:\CUSTCODE.KEY"
fileD0\$ = "C:\CUSTDATA.DAT"
'[code to open files deleted for space]
DelName\$ = "Sioux," + "Leslie"
DelCode\$ = "100500"
stat = DeleteKey(0,DelName\$)
IF stat = 0 THEN
 stat = DeleteKeyRecord(1,0,DelCode\$)

ExitQBTre

TYPE FUNCTION

SYNTAX stat = ExitQBTre

PARAMETERS None.

USE Close all QBTre files and release the RAM used by the buffers. Use is optional.

RULES None.

NOTES Once ExitQBTre() is called you must InitQBTre() before using any other QBTre function. This is the only way you can release the RAM used by the data buffers other than exiting your program.

RETURN Error 233.

EXAMPLE PRINT "done."
 nul = ExitQBTre
 END

FlushDataFile

TYPE FUNCTION

SYNTAX stat = FlushDataFile(dfile,dup)

PARAMETERS dfile - INTEGER.Number used as fileno in OpenDataFile().
 dup - INTEGER.Non-zero to force a directory update.

USE Flush the data file by writing the data file header in RAM to the data file header on disk. If dup is non-zero then also update the directory entry.

RULES If any part of dfile has been locked with either LockDataHeader() or LockRecord() then dup must be zero. If not, a locking violation error will occur when you unlock.

NOTES If DOS 3.3+ is detected, the header is written as above but the directory information is updated using the DOS commit function (func 68h), regardless the value of dup. When programming for network environments, always use dup=0.
 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. Closing and then reopening the file will have the same effect as this routine but will incur a much higher overhead.

RETURN Errors 219, 222, 225, and DOS.

EXAMPLE IF IsOnNetwork THEN dup = 0 ELSE dup = -1
 stat = FlushDataFile(dfile,dup)

FlushKeyFile

TYPE FUNCTION

SYNTAX stat = FlushKeyFile(kfile,dup)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
dup - INTEGER.Non-zero to force a directory update.

USE Flush the key file by writing the key file header in RAM to the key file header on disk. If dup is non-zero then also update the directory entry.

RULES If kfile has been locked with LockKeyFile() then dup should be zero.

NOTES If DOS 3.3+ is detected, the header is written as above but the directory information is updated using the DOS commit function (func 68h), regardless the value of dup. When programming for network environments, always use dup=0.
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. Closing and then reopening the file will have the same effect as this routine but will incur a much higher overhead.

RETURN Errors 219, 222, 225, and DOS.

EXAMPLE IF IsOnNetwork THEN dup = 0 ELSE dup = -1
stat = FlushKeyFile(kfile,dup)

FreeDataFile

TYPE FUNCTION

SYNTAX fileno = FreeDataFile

PARAMETERS None.

USE Obtain a free QBTree data file handle.

RULES None.

NOTES The handle returned will be a number between 0 and MDF
 where MDF is specified in InitQBTree().
 This function is analagous to QB's FREEFILE.

RETURN Available free data file handle or -1 if none are
 available.

EXAMPLE dfileno = FreeDataFile
 IF dfileno < 0 THEN DoNoDataHandlesAvailable
 stat = OpenDataFile(dfname\$,dfileno,asmode)

FreeKeyFile

TYPE FUNCTION

SYNTAX fileno = FreeKeyFile

PARAMETERS None.

USE Obtain a free QBTREE key file handle.

RULES None.

NOTES The handle returned will be a number between 0 and MKF
 where MKF is specified in InitQBTREE().
 This function is analagous to QB's FREEFILE.

RETURN Available free key file handle or -1 if none are
 available.

EXAMPLE kfileno = FreeKeyFile
 IF kfileno < 0 THEN DoNoKeyHandlesAvailable
 stat = OpenKeyFile(kfname\$,kfileno,asmode)

GetDirect

TYPE FUNCTION

SYNTAX stat = GetDirect(dfile,recno&,Qrec\$)

PARAMETERS dfile - INTEGER.Number used as fileno in OpenDataFile().
 recno& - LONG.Number of record to get.
 Qrec\$ - STRING.The data record at recno& is placed in
 this variable.

USE Get a data record from the data file without using any
 index file. recno& = 1 gets the very first data record in
 dfile, 2 gets then next, and so on.

RULES recno& must be > 0 and <= the last possible record.

NOTES This function can be used to get records from the data
 file to reindex a key file. Error 224 is returned if you
 try to get a record past the absolute end of file.

RETURN Errors 219, 222, 224, and DOS.

EXAMPLE INPUT "Retrieve which record number ";recno&
 stat = GetDirect(0,recno&,Qrec\$)
 IF stat = 0 THEN
 PRINT "Record=";Qrec\$

GetEqual

TYPE FUNCTION

SYNTAX stat = GetEqual(kfile,dfile,Qkey\$,Qrec\$)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
 dfile - INTEGER.Number used as fileno in OpenDataFile().
 Qkey\$ - STRING.The key (or partial key) to search for.
 Qrec\$ - STRING.The data record for the found key is
 placed in this variable.

USE Search for the key Qkey\$ in kfile. If it's found, return
 the data record that it's pointing to in Qrec\$. If it's
 not found the QBTree tracking pointers are positioned so
 that by using GetNext() or GetPrev() the key records
 after or before could be found.

RULES Qkey\$ must not begin with an ASCII 0 or 255.

NOTES For partial searches use a key that you want to start
 the search on.
 For instance, to find all surnames starting with a key
 of KING in your database, use a Qkey\$ = "KING". Even if
 there is no exact match with GetEqual() you can still get
 all other records starting with KING, e.g., KINGMAN,
 KINGSTON, etc., by using the GetNext() routine while
 LEFT\$(Qkey\$,4) = "KING", or until the end of file is
 reached.

RETURN Errors 200, 202, 204, 219, 222, 223, and DOS.

EXAMPLE INPUT "Get which key ";Qkey\$
 stat = GetEqual(0,0,Qkey\$,Qrec\$)
 IF stat = 0 THEN
 PRINT "Record=";Qrec\$

GetFirst

TYPE FUNCTION

SYNTAX stat = GetFirst(kfile,dfile,Qkey\$,Qrec\$)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
 dfile - INTEGER.Number used as fileno in OpenDataFile().
 Qkey\$ - STRING.The first key in kfile is placed in this
 variable.
 Qrec\$ - STRING.The data record for the first key is
 placed in this variable.

USE Get the first key in kfile and return it in Qkey\$. Also
 get that key's data record and return it in Qrec\$. Since
 QBTree sorts on ASCII values, the first key will be the
 key with the lowest ASCII value.

RULES None.

NOTES For sequential in-order processing of the entire data
 file in kfile order, you would want to start at the very
 first key; this function does just that. After processing
 this first key and record, use GetNext() and continue
 processing until end of file (error 202).

RETURN Errors 204, 219, 222, and DOS.

EXAMPLE stat = GetFirst(0,0,Qkey\$,Qrec\$)
 IF stat = 0 THEN
 PRINT "First key="; Qkey\$

GetGT

TYPE FUNCTION

SYNTAX stat = GetGT(kfile,dfile,Qkey\$,Qrec\$)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
dfile - INTEGER.Number used as fileno in OpenDataFile().
Qkey\$ - STRING.Get the next key greater than Qkey\$ in
kfile and also place it in this variable.
Qrec\$ - STRING.The data record for this GT key is placed
in this variable.

USE Get the next key in kfile that is greater than Qkey\$ and
return it in Qkey\$. Also get that key's data record and
return it in Qrec\$.

RULES None.

NOTES The original value in Qkey\$ is replaced with the next GT
key.

RETURN Errors 202, 204, 219, 222, and DOS.

EXAMPLE INPUT "Get first key greater than ";lastkey\$
stat = GetGT(0,0,lastkey\$,Qrec\$)
IF stat = 0 THEN
 PRINT "First greater key=";lastkey\$

GetGTE

TYPE FUNCTION

SYNTAX stat = GetGTE(kfile,dfile,Qkey\$,Qrec\$)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
dfile - INTEGER.Number used as fileno in OpenDataFile().
Qkey\$ - STRING.Get the next key greater or equal to
Qkey\$ in kfile and also place it in this variable.
Qrec\$ - STRING.The data record for this GTE key is
placed in this variable.

USE Get the next key in kfile that is greater or equal to
Qkey\$ and return it in Qkey\$. Also get that key's data
record and return it in Qrec\$.

RULES None.

NOTES The original value in Qkey\$ is replaced with the next
GTE key.

RETURN Errors 202, 204, 219, 222, and DOS.

EXAMPLE INPUT "Get first key GTE to ";lastkey\$
stat = GetGTE(0,0,lastkey\$,Qrec\$)
IF stat = 0 THEN
 PRINT "First key GTE=";lastkey\$

GetLast

TYPE FUNCTION

SYNTAX stat = GetLast(kfile,dfile,Qkey\$,Qrec\$)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
dfile - INTEGER.Number used as fileno in OpenDataFile().
Qkey\$ - STRING.The last key in kfile is placed in this variable.
Qrec\$ - STRING.The data record for the last key is placed in this variable.

USE Get the last key in kfile and return it in Qkey\$. Also get that key's data record and return it in Qrec\$. Since QBTtree sorts on ASCII values, the last key will be the key with the highest ASCII value.

RULES None.

NOTES For sequential reverse-order processing of the entire data file in kfile order, you would want to start at the very last key; this function does just that. After processing this last key and record, use GetPrev() and continue processing until top of file (error 203).

RETURN Errors 204, 219, 222, and DOS.

EXAMPLE stat = GetLast(0,0,Qkey\$,Qrec\$)
 IF stat = 0 THEN
 PRINT "Last key="; Qkey\$

GetLT

TYPE FUNCTION

SYNTAX stat = GetLT(kfile,dfile,Qkey\$,Qrec\$)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
dfile - INTEGER.Number used as fileno in OpenDataFile().
Qkey\$ - STRING.Get the previous key less than Qkey\$ in
kfile and also place it in this variable.
Qrec\$ - STRING.The data record for this LT key is placed
in this variable.

USE Get the previous key in kfile that is less than Qkey\$
and return it in Qkey\$. Also get that key's data record
and return it in Qrec\$.

RULES None.

NOTES The original value in Qkey\$ is replaced with the
previous LT key.

RETURN Errors 203, 204, 219, 222, and DOS.

EXAMPLE INPUT "Get key less than ";lastkey\$
stat = GetLT(0,0,lastkey\$,Qrec\$)
IF stat = 0 THEN
 PRINT "Last less than key=";lastkey\$

GetLTE

TYPE FUNCTION

SYNTAX stat = GetLTE(kfile,dfile,Qkey\$,Qrec\$)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
dfile - INTEGER.Number used as fileno in OpenDataFile().
Qkey\$ - STRING.Get the previous key less or equal to
Qkey\$ in kfile and also place it in this variable.
Qrec\$ - STRING.The data record for this LTE key is
placed in this variable.

USE Get the previous key in kfile that is less or equal to
Qkey\$ and return it in Qkey\$. Also get that key's data
record and return it in Qrec\$.

RULES None.

NOTES The original value in Qkey\$ is replaced with the
previous LTE key.

RETURN Errors 203, 204, 219, 222, and DOS.

EXAMPLE INPUT "Get key LTE to ";lastkey\$
 stat = GetLTE(0,0,lastkey\$,Qrec\$)
 IF stat = 0 THEN
 PRINT "Last LTE=";lastkey\$

GetNext

TYPE FUNCTION

SYNTAX stat = GetNext(kfile,dfile,Qkey\$,Qrec\$)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
dfile - INTEGER.Number used as fileno in OpenDataFile().
Qkey\$ - STRING.The next key in kfile is placed in this variable.
Qrec\$ - STRING.The data record for this next key is placed in this variable.

USE Get the next key in kfile and return it in Qkey\$. Also get that key's data record and return it in Qrec\$. Use this function to process the key file sequentially in-order.

RULES None.

NOTES This function is usually used after a GetFirst() or GetEqual(). If you are searching for a key based on a partial key, use GetEqual() with the partial key and continue searching with GetNext() until you either find the key you want, determine that it's not in the key file, or reach the end of file.

RETURN Errors 202, 204, 219, 222, and DOS.

EXAMPLE partkey\$ = "73ST"
stat = GetEqual(0,0,partkey\$,Qrec\$)
IF stat = 200 THEN
 DO
 stat = GetNext(0,0,Qkey\$,Qrec\$)
 IF stat = 0 THEN
 IsKey = AskIfWantedKey(Qkey\$)
 IF IsKey THEN EXIT DO
 ENDIF
 LOOP WHILE stat = 0

GetPosition

TYPE FUNCTION

SYNTAX stat = GetPosition(kfile,recno&)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
recno& - LONG.The current record number last used by
kfile is placed in this variable.

USE Get the last data record number accessed by kfile. This
record number is updated everytime you use any of the Add
or Get functions (excluding GetDirect), or UpdateRecord().

RULES None.

NOTES This function is a special-use routine. If you construct
relational-based file structures only then this function
would not be needed. However, there are times, when for
reasons of speed or simplicity, that you need to sidestep
relational ideals.

Let's say we have 2 key files, k1 and k2, both pointing
to the same data record in the single data file, d1. We
want to delete a key in k1 and its data record in d1, and
also delete the key in k2 that is pointing to that same
record in d1. We know there is a key in k2 that points to
that record in d1. The problem is that, since we don't
have the key of k2 imbedded in the record, we don't have
enough information in k2 to allow us to find the key to
delete by using any of the Get() routines. What we need
to do is GetEqual() the key to delete in k1, and then do
a GetPosition() on k1 and save the record number
returned. Then we do a GetFirst() of k2, do a
GetPosition(), check to see if it's the same record
number as the k1 key, and if so, we found the k2 key to
delete. If it's not the same, we use
GetNext()/GetPosition() until we find it. When we find a
match, we use DeleteKey() on the k2 key returned by
GetNext(), and then DeleteKeyRecord() the k1 key we know
about. Both keys and the data record are now deleted.
See LockRecord() for another use of this function.

RETURN Error 219, 222.

EXAMPLE stat = GetFirst(k1,d1,Qkey\$,Qrec\$)
PRINT "First key in k1 is using record#";
stat = GetPosition(k1,recno&)
PRINT recno&

GetPrev

TYPE FUNCTION

SYNTAX stat = GetPrev(kfile,dfile,Qkey\$,Qrec\$)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
dfile - INTEGER.Number used as fileno in OpenDataFile().
Qkey\$ - STRING.The previous key in kfile is placed in this variable.
Qrec\$ - STRING.The data record for this previous key is placed in this variable.

USE Get the previous key in kfile and return it in Qkey\$.
Also get that key's data record and return it in Qrec\$.
Use this function to process the key file sequentially in reverse-order.

RULES None.

NOTES This function is usually used after a GetLast() or GetEqual(). If you are searching for a key based on a partial key, use GetEqual() with the partial key and continue searching with GetPrev() until you either find the key you want, determine that it's not in the key file, or reach the top of file.

RETURN Errors 203, 204, 219, 222, and DOS.

EXAMPLE partkey\$ = "73STZZZZ"
stat = GetEqual(0,0,partkey\$,Qrec\$)
IF stat = 200 THEN
 DO
 stat = GetPrev(0,0,Qkey\$,Qrec\$)
 IF stat = 0 THEN
 IsKey = AskIfWantedKey(Qkey\$)
 IF IsKey THEN EXIT DO
 ENDIF
 LOOP WHILE stat = 0

InitQBTree

TYPE FUNCTION

SYNTAX stat = InitQBTree(MKF,MDF)

PARAMETERS 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).

USE Initialize the QBTree file system and allocate the required buffers.

RULES This function must be called before any other QBTree function.

NOTES QBTree sets aside buffers for each file request you make with MKF and MDF. These buffers are created dynamically, e.g., REDIM keybuff(0 TO MKF) AS keynodeTYPE, where each element in the array is reserved for a separate file. Using QBTree functions without having first called this function may cause various BASIC run-time errors. QBTree 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.

RETURN Errors 225, 234.

EXAMPLE 'need 30 key and 10 data files this session
stat = InitQBTree(29,9)
'can OpenKeyFile(file\$,numkey,asmode) where numkey
'is 0-29 and OpenDataFile(file\$,numdat,asmode)
'where numdat is 0-9.

LockDataHeader

TYPE FUNCTION

SYNTAX stat = LockDataHeader(dfile)

PARAMETERS dfile - INTEGER.Number used as fileno in OpenDataFile().

USE Network. Lock the data header of dfile so that no other process may read or write it.

RULES SHARE.EXE must be installed.
You must unlock the data header when you no longer need exclusive access to it. If your program terminates without unlocking active locks, the result is undefined.

NOTES This function is not needed if dfile's asmode=&H12.
This function is needed only if the data file is being used on a network and you need to ensure that no other process will change things while you are working on the file. In addition to locking the data header, you may also need to lock a specific record in dfile or even all the records in dfile using LockRecord().
Anytime you use AddKeyRecord() or DeleteKeyRecord() you must first lock the data header and then lock all records.
This function will automatically call LoadDataHeader(), an internal routine, to refresh the data file header information from disk.

RETURN Errors 219, 222, and DOS.

EXAMPLE stat = OpenDataFile(datfile\$,0,asmode)
IF stat = 0 THEN
 FOR retries = 1 to MaxRetries
 stat = LockDataHeader(0)
 IF stat = 0 THEN
 EXIT FOR
 ELSE
 RetryDelayAbit
 ENDIF
 NEXT
 IF stat THEN DoAccessError stat

LockKeyFile

TYPE FUNCTION

SYNTAX stat = LockKeyFile(kfile)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().

USE Network. Lock the key file so that no other process may read or write it.

RULES SHARE.EXE must be installed.
You must unlock the key file when you no longer need exclusive access to it. If your program terminates without unlocking active locks, the result is undefined.

NOTES This function is not needed if kfile's asmode=&H12.
This function is needed only if the key file is being used on a network and there's a possibility that another process may access it.
This locks the file by locking all bytes within it. Anytime you use AddKey(), AddKeyRecord(), DeleteKey(), DeleteKeyRecord(), or StoreKey() you must first lock the key file.
This function will automatically call LoadKeyHeader(), an internal routine, to refresh the key file header information from disk.

RETURN Errors 219, 222, and DOS.

EXAMPLE stat = OpenKeyFile(keyfile\$,0,asmode)
IF stat = 0 THEN
 FOR retries = 1 to MaxRetries
 stat = LockKeyFile(0)
 IF stat = 0 THEN
 EXIT FOR
 ELSE
 RetryDelayAbit
 ENDIF
 NEXT
IF stat THEN DoAccessError stat

LockRecord

TYPE FUNCTION

SYNTAX stat = LockRecord(dfile,recno&)

PARAMETERS dfile - INTEGER.Number used as fileno in OpenDataFile().
 recno& - LONG.Record number in dfile to lock, or if 0,
 lock all records in dfile.

USE Network. Lock the record(s) in dfile so that no other
 process may read or write it.

RULES SHARE.EXE must be installed.
 You must unlock the record when you no longer need
 exclusive access to it. If you lock multiple records by
 calling LockRecord() multiple times then each record
 needs to be unlocked using UnlockRecord() an equal number
 of times. If your program terminates without unlocking
 active locks, the result is undefined.

NOTES This function is not needed if dfile's asmode=&H12.
 This function is needed only if the data file is being
 used on a network and you need to ensure that no other
 process will change the current record you're working on.
 In addition to locking a specific record you may need to
 lock the data header.
 Anytime you use AddKeyRecord() or DeleteKeyRecord() you
 must lock the data header and lock all records (recno&=0)
 in addition to using LockKeyFile().
 If you are using one of the Get() functions to access
 the data file and plan on using UpdateRecord(), use
 GetPosition() after the Get() function to determine which
 record you need to lock before performing the update.

RETURN Errors 219, 222, and DOS.

EXAMPLE stat = OpenDataFile(datfile\$,0,asmode)
 IF stat = 0 THEN
 FOR retries = 1 to MaxRetries
 stat = LockDataHeader(0)
 IF stat = 0 THEN
 EXIT FOR
 ELSE
 RetryDelayAbit
 ENDIF
 NEXT
 IF stat = 0 THEN stat = LockRecord(0,0)

OpenDataFile

TYPE FUNCTION

SYNTAX stat = OpenDataFile(filename\$,fileno,asmode)

PARAMETERS filename\$ - STRING.Pathname of data file to open.
fileno - INTEGER.Number to associate the data file with for future operations. Valid range is 0 to MDF.
asmode - INTEGER.Network. When asmode (Access/Sharing mode) is:
 &H42 open with shared R/W access (others R/W access),
 2 open in DOS 2.x compatible R/W mode (not for networks),
 &H12 open with exclusive R/W access (others have no access),
 &H22 open with exclusive W access (others have read access),
 &H32 open with exclusive R access (others have write access),
 any valid DOS value permitted.

USE Open an existing QBTree data file with the mode determined by asmode.

RULES fileno can be any unused fileno 0 to MDF. See InitQBTree. SHARE.EXE must be installed to use any of the exclusive access modes as well as any QBTree locking function (a LAN is not required).

NOTES For simplified network programming where the file can be opened for exclusive use set asmode to &H12. As long as the file remains open no other process may access any part of it. Using asmode = &H12 means that you do not need to use any of the other network support functions to successfully share the file on a network. If you need to use specific record locks set asmode to &H42. For non-network programs use asmode=2.
The fileno actually represent the data buffer index that this file will be using.

RETURN Errors 219, 227, 228, 233, and DOS.

EXAMPLE dfile = 0
 asmode = &H42
 stat = OpenDataFile(filename\$,dfile,asmode)

OpenKeyFile

TYPE FUNCTION

SYNTAX stat = OpenKeyFile(filename\$,fileno,asmode)

PARAMETERS filename\$ - STRING.Pathname of key file to open.
fileno - INTEGER.Number to associate the key file with
for future operations. Valid range is 0 to MKF.
asmode - INTEGER.Network. When asmode (Access/Sharing
mode) is:
&H42 open with shared R/W access (others R/W access),
2 open in DOS 2.x compatible R/W mode (not for networks),
&H12 open with exclusive R/W access (others have no access),
&H22 open with exclusive W access (others have read access),
&H32 open with exclusive R access (others have write access),
any valid DOS value permitted.

USE Open an existing QBTREE key file with the mode
determined by asmode.

RULES fileno can be any unused fileno 0 to MKF. See
InitQBTREE.
SHARE.EXE must be installed to use any of the exclusive
access modes as well as any QBTREE locking function (a
LAN is not required).

NOTES For simplified network programming where the file can be
opened for exclusive use set asmode to &H12. As long as
the file remains open no other process may access any
part of it. Using asmode = &H12 means that you do not
need to use any of the other network support functions to
successfully share the file on a network. If you need to
use specific record locks set asmode to &H42. For
non-network programs use asmode=2.
The fileno actually represent the key buffer index that
this file will be using.

RETURN Errors 219, 227, 228, 233, and DOS.

EXAMPLE kfile = 0
asmode = &H42
stat = OpenKeyFile(filename\$,kfile,asmode)

RetrieveEqual

TYPE FUNCTION

SYNTAX stat = RetrieveEqual(kfile,Qkey\$,Qrecno&)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
Qkey\$ - STRING.The key (or partial key) to search for.
Qrecno& - LONG.The record number that Qkey\$ indexes is placed in this variable.

USE Search for the key Qkey\$ in kfile. If it is found, return the record number that was set for this key by StoreKey(). If it's not found the QBTree tracking pointers are positioned so that by using RetrieveNext() or RetrievePrev() the keys after or before could be found.

RULES Qkey\$ must not begin with an ASCII 0 or 255.

NOTES For partial searches use a key that you want to start the search on.
For instance, to find all surnames starting with KING in your database, use a Qkey\$ = "KING". Even if there is no exact match with RetrieveEqual() you can still get all other keys starting with KING, e.g., KINGMAN, KINGSTON, etc., by using the RetrieveNext() routine while LEFT\$(Qkey\$,4) = "KING", or until the end of file is reached.
Unlike the Get() functions, the Retrieve() functions operate only on key files. Any data file handling is left to you using whatever data file format you need.

RETURN Errors 200, 202, 204, 219, 222, 223, and DOS.

EXAMPLE stat = RetrieveEqual(0,Qkey\$,Qrecno&)
IF stat = 200 THEN
 stat = RetrieveNext(0,Qkey\$,Qrecno&)

RetrieveFirst

TYPE FUNCTION

SYNTAX stat = RetrieveFirst(kfile,Qkey\$,Qrecno&)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
Qkey\$ - STRING.The first key in kfile is placed in this variable.
Qrecno& - LONG.The record number that the first key indexes is placed in this variable.

USE Get the first key in kfile and return it in Qkey\$. Also get that key's record number and return it in Qrecno&. Since QBTree sorts on ASCII values, the first key will be the key with the lowest ASCII value.

RULES None.

NOTES For sequential in-order processing of the entire key file, you would want to start at the very first key; this function does just that. After processing this first key, use RetrieveNext() and continue processing until end of file (error 202).

RETURN Errors 204, 219, 222, and DOS.

EXAMPLE stat = RetrieveFirst(0,Qkey\$,Qrecno&)
IF stat = 0 THEN
 PRINT "First key "; Qkey\$
 PRINT "indexes record number"; Qrecno&

RetrieveGT

TYPE FUNCTION

SYNTAX stat = RetrieveGT(kfile,Qkey\$,Qrecno&)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
Qkey\$ - STRING.Get the next key greater than Qkey\$ in
kfile and also place it in this variable.
Qrecno& - LONG.The record number that the next GT key
indexes is placed in this variable.

USE Get the next key in kfile that is greater than Qkey\$ and
return it in Qkey\$. Also get that key's data record
number and return it in Qrecno&.

RULES None.

NOTES The original value in Qkey\$ is replaced with the next GT
key.

RETURN Errors 202, 204, 219, 222, and DOS.

EXAMPLE stat = RetrieveGT(0,lastkey\$,Qrecno&)
IF stat = 0 THEN
 PRINT "Rec# of first greater key=";Qrecno&

RetrieveGTE

TYPE FUNCTION

SYNTAX stat = RetrieveGTE(kfile,Qkey\$,Qrecno&)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
Qkey\$ - STRING.Get the next key greater or equal to
Qkey\$ in kfile and also place it in this variable.
Qrecno& - LONG.The record number that the next GTE key
indexes is placed in this variable.

USE Get the next key in kfile that is greater or equal to
Qkey\$ and return it in Qkey\$. Also get that key's data
record number and return it in Qrecno&.

RULES None.

NOTES The original value in Qkey\$ is replaced with the next
GTE key.

RETURN Errors 202, 204, 219, 222, and DOS.

EXAMPLE stat = RetrieveGTE(0,lastkey\$,Qrecno&)
IF stat = 0 THEN
 PRINT "Rec# for first GTE=";Qrecno&

RetrieveLast

TYPE FUNCTION

SYNTAX stat = RetrieveLast(kfile,Qkey\$,Qrecno&)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
Qkey\$ - STRING.The last key in kfile is placed in this variable.
Qrecno& - LONG.The record number that the last key indexes is placed in this variable.

USE Get the last key in kfile and return it in Qkey\$. Also get that key's record number and return it in Qrecno&. Since QBTree sorts on ASCII values, the last key will be the key with the highest ASCII value.

RULES None.

NOTES For sequential reverse-order processing of the entire key file, you would want to start at the very last key; this function does just that. After processing this last key, use RetrievePrev() and continue processing until top of file (error 203).

RETURN Errors 204, 219, 222, and DOS.

EXAMPLE stat = RetrieveLast(0,Qkey\$,Qrecno&)
IF stat = 0 THEN
 PRINT "Last key "; Qkey\$
 PRINT "indexes record number"; Qrecno&

RetrieveLT

TYPE FUNCTION

SYNTAX stat = RetrieveLT(kfile,Qkey\$,Qrecno&)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
Qkey\$ - STRING.Get the previous key less than Qkey\$ in
kfile and also place it in this variable.
Qrecno& - LONG.The record number that the previous LT
key indexes is placed in this variable.

USE Get the previous key in kfile that is less than Qkey\$
and return it in Qkey\$. Also get that key's data record
number and return it in Qrecno&.

RULES None.

NOTES The original value in Qkey\$ is replaced with the
previous LT key.

RETURN Errors 203, 204, 219, 222, and DOS.

EXAMPLE stat = RetrieveLT(0,lastkey\$,Qrecno&)
IF stat = 0 THEN
 PRINT "Rec# to last less than key=";Qrecno&

RetrieveLTE

TYPE FUNCTION

SYNTAX stat = RetrieveLTE(kfile,Qkey\$,Qrecno&)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
Qkey\$ - STRING.Get the previous key less or equal to Qkey\$ in kfile and also place it in this variable.
Qrecno& - LONG.The record number that the previous LTE key indexes is placed in this variable.

USE Get the previous key in kfile that is less or equal to Qkey\$ and return it in Qkey\$. Also get that key's data record number and return it in Qrecno&.

RULES None.

NOTES The original value in Qkey\$ is replaced with the previous LTE key.

RETURN Errors 203, 204, 219, 222, and DOS.

EXAMPLE stat = RetrieveLTE(0,lastkey\$,Qrecno&)
 IF stat = 0 THEN
 PRINT "Rec# to last LTE key=";Qrecno&

RetrieveNext

TYPE FUNCTION

SYNTAX stat = RetrieveNext(kfile,Qkey\$,Qrecno&)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
Qkey\$ - STRING.The next key in kfile is placed in this variable.
Qrecno& - LONG.The record number that the next key indexes is placed in this variable.

USE Get the next key in kfile and return it in Qkey\$. Also get that key's record number and return it in Qrecno&. Use this function to process the key file sequentially in-order.

RULES None.

NOTES This function is usually used after a RetrieveFirst() or RetrieveEqual(). If you are searching for a key based on a partial key, use RetrieveEqual() with the partial key and continue searching with RetrieveNext() until you either find the key you want, determine that it's not in the key file, or reach the end of file.

RETURN Errors 202, 204, 219, 222, and DOS.

EXAMPLE partkey\$ = "73ST"
stat = RetrieveEqual(0,partkey\$,Qrecno&)
IF stat = 200 THEN
 DO
 stat = RetrieveNext(0,Qkey\$,Qrecno&)
 IF stat = 0 THEN
 IsKey = AskIfWantedKey(Qkey\$)
 IF IsKey THEN EXIT DO
 ENDIF
 LOOP WHILE stat = 0

RetrievePrev

TYPE FUNCTION

SYNTAX stat = RetrievePrev(kfile,Qkey\$,Qrecno&)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
Qkey\$ - STRING.The previous key in kfile is placed in this variable.
Qrecno& - LONG.The record number that the previous key indexes is placed in this variable.

USE Get the previous key in kfile and return it in Qkey\$. Also get that key's record number and return it in Qrecno&. Use this function to process the key file sequentially in reverse-order.

RULES None.

NOTES This function is usually used after a RetrieveLast() or RetrieveEqual(). If you are searching for a key based on a partial key, use RetrieveEqual() with the partial key and continue searching with RetrievePrev() until you either find the key you want, determine that it's not in the key file, or reach the top of file.

RETURN Errors 203, 204, 219, 222, and DOS.

EXAMPLE partkey\$ = "73STZZZZ"
stat = RetrieveEqual(0,partkey\$,Qrecno&)
IF stat = 200 THEN
 DO
 stat = RetrievePrev(0,Qkey\$,Qrecno&)
 IF stat = 0 THEN
 IsKey = AskIfWantedKey(Qkey\$)
 IF IsKey THEN EXIT DO
 ENDIF
 LOOP WHILE stat = 0

QBTreeVer

TYPE FUNCTION

SYNTAX stat = QBTreeVer(version)

PARAMETERS version - INTEGER. The version of QBTree in use is placed in this variable.

USE Return the version of QBTree in use.

RULES None.

NOTES Returns the version X 100. Version 5.50 returns version as 550.

RETURN Always 0.

EXAMPLE nul = QBTreeVer(version)
PRINT USING "QBTree version ##.##"; version/100

StatDataFile

TYPE FUNCTION

SYNTAX stat = StatDataFile(dfile, reclen, recs&, bfileno)

PARAMETERS dfile - INTEGER. Number used as fileno in OpenDataFile().
 reclen - INTEGER. The record length of dfile is returned
 in this variable.
 recs& - LONG. The number of active records in dfile is
 placed in this variable.
 bfileno - INTEGER. The DOS file handle for dfile is
 placed in this variable.

USE Obtain basic information on the opened data file.

RULES None.

NOTES None.

RETURN Errors 219, 222.

EXAMPLE stat = StatDataFile(0, reclen, recs&, bfileno)
 IF stat = 0 THEN PRINT "Rec length="; reclen

StatKeyFile

TYPE FUNCTION

SYNTAX stat = StatKeyFile(kfile,keylen,keys&,bfileno)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
keylen - INTEGER.The key length of kfile is returned in this variable.
keys& - LONG.The number of active keys in kfile is placed in this variable.
bfileno - INTEGER.The DOS file handle for kfile is placed in this variable.

USE Obtain basic information on the opened key file.

RULES None.

NOTES None.

RETURN Errors 219, 222.

EXAMPLE stat = StatKeyFile(0,keylen,keys&,bfileno)
IF stat = 0 THEN PRINT "Key length="; keylen

StoreKey

TYPE FUNCTION

SYNTAX stat=StoreKey(kfile,Qkey\$,Qrecno&)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().
Qkey\$ - STRING.Key to add to kfile.
Qrecno& - LONG.The record number of the data record in
your data file that Qkey\$ indexes.

USE Store the key to the key file and associate with it
Qrecno&.

RULES The key must be unique.
Qkey\$ must not begin with an ASCII 0 or 255.
Qrecno& must be >= 0 and <= 16,777,215.

NOTES 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.
The difference between StoreKey() and AddKey(Record) ()
is that StoreKey() does not do any data file handling.
Whereas the AddKey() routines handle both the key and
data file by automatically inserting keys, assigning
record numbers, and reading and writing the data records,
StoreKey() just handles inserting keys. The record
numbers associated with those keys (and retrieved with
one of the Retrieve() routines) are assigned by you.

RETURN Errors 201, 219, 222, 223, 224 and DOS.

EXAMPLE Qkey\$ = LEFT\$(datarecord\$,keylen)
Qrecno& = GetNextFreeRecord&(userdfile)
stat = StoreKey(0,Qkey\$,Qrecno&)

UnlockDataHeader

TYPE FUNCTION

SYNTAX stat = UnlockDataHeader(dfile)

PARAMETERS dfile - INTEGER.Number used as fileno in OpenDataFile().

USE Network. Flush the internal buffers and unlock the data header of dfile so that other processes may read or write it.

RULES SHARE.EXE must be installed.
After locking the data header you must unlock it when you no longer need exclusive access to it. If your program terminates without unlocking active locks, the result is undefined.

NOTES This function is not needed if dfile's asmode=&H12.
This function first calls FlushDataFile(dfile,0) and then releases the lock.

RETURN Errors 219, 222, and DOS.

EXAMPLE '[see LockDataHeader for lock and open code]
stat = UnlockDataHeader(dfile)
IF stat = 0 THEN
 PRINT "Data header is unlocked"

UnlockKeyFile

TYPE FUNCTION

SYNTAX stat = UnlockKeyFile(kfile)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().

USE Network. Flush the internal buffers and unlock the key file so that other processes may read or write it.

RULES SHARE.EXE must be installed.
After you lock a key file you must unlock it when you no longer need exclusive access to it. If your program terminates without unlocking active locks, the result is undefined.

NOTES This function is not needed if kfile's asmode=&H12.
This function first calls FlushKeyFile(kfile,0) and then releases the lock.

RETURN Errors 219, 222, and DOS.

EXAMPLE '[see LockKeyFile() for lock and open code]
stat = UnlockKeyFile(kfile)
IF stat = 0 THEN
 PRINT "Key file is unlocked"

UpdateRecord

TYPE FUNCTION

SYNTAX stat = UpdateRecord(dfile,Qrec\$)

PARAMETERS dfile - INTEGER.Number used as fileno in OpenDataFile().
 Qrec\$ - STRING.Data record to update the current data
 record with.

USE After finding a key with one of the Get() functions you
 can update that record using this function.

RULES UpdateRecord() does not affect the key file so you must
 not change the key portion of the data record.
 You must UpdateRecord() immediately after finding a key
 with one of the Get() functions (Equal, First, Last,
 Next, Prev, GT, GTE, LT, LTE) otherwise you may
 inadvertently change the current data record and update
 the wrong record.

NOTES If you are changing the key portion of the data record
 you should not use UpdateRecord(). Instead, first
 AddKeyRecord() the newly updated key and data record, and
 if successful, continue with DeleteKeyRecord() of the old
 key and data record.

RETURN Errors 206, 219, 222, and DOS.

EXAMPLE stat = GetEqual(0,0,Qkey\$,Qrec\$)
 IF stat = 0 THEN
 MID\$(Qrec\$,10,3) = "OLD"
 stat = UpdateRecord(0,Qrec\$)

APPENDIX A. SUPPORT FUNCTIONS

These functions are used in support of QBTREE and can also be used by you in your programs. All are FUNCTIONS except GetDiskInfo and MemCopy.

- 1) ExpandDataFile(dfileno,norecs&)
- 2) ExpandKeyFile(kfileno,nokeys&)
- 3) FileExists(filename\$)
- 4) GetDefaultDrive()
- 5) GetDiskInfo(drive\$,ACL,MCL,BPS,SPCL,freebytes&)
- 6) GetDosVersion()
- 7) GetXEInfo(class,action,locus)
- 8) LoadDataHeader(dfileno)
- 9) LoadKeyHeader(kfileno)
- 10) SFTFiles()
- 11) CreateFile(filenameZ\$,attribute)
- 12) OpenDevice(filenameZ\$,mode,handle,flen&)
- 13) ReadDevice(handle,start&,bytes&,vseg,voff)
- 14) WriteDevice(handle,start&,bytes&,vseg,voff)
- 15) CloseDevice(handle)
- 16) DeleteFile(filenameZ\$)
- 17) RenameFile(oldfileZ\$,newfileZ\$)
- 18) MemCopy(seg1,off1,seg2,off2,bytes)

ExpandKeyFile

TYPE FUNCTION

SYNTAX stat=ExpandKeyFile(kfileno,nokeys&)

PARAMETERS kfileno - INTEGER.Number used as fileno in OpenKeyFile().
nokeys& - LONG.The number of keys to expand kfileno by.

USE Expand the physical size of kfileno so that it can contain at least nokeys& more keys without file fragmentation.

RULES None.

NOTES This routine is optional.
Two uses of this routine are: 1) tell DOS to allocate disk space for nokeys& more keys beforehand to minimize file fragmentation, and 2) ensure that your program will have room for nokeys& more keys, now and in the future. In other words, if you know your program will ultimately need room for 60,000 keys then you could reserve the space all at once.
Keys that have been deleted are included when calculating the number of keys to add. For instance, let's say you add 100 keys to a new file and then delete 40 of them. Then, you decide to expand the key file by 100 keys. ExpandKeyFile() knows that approximately 40% of the keys have been deleted so it physically expands the key file by 60 keys, not 100.
QBTree is approximately 67% efficient in using node space. This is factored in when calculating the space required to allocate nokeys& more keys and thus the expansion is an approximation.
Existing data in the newly added key space is not cleared.
The space added will be as contiguous as the current state of fragmentation on the disk. If the disk is highly fragmented then this routine will get highly-fragmented space from DOS.

RETURN Errors 208, 219, 222, 224, and DOS.

EXAMPLE INPUT "Keys to expand key file ";xkeys&
stat=ExpandKeyFile(kfileno,xkeys&)

FileExists

TYPE FUNCTION

SYNTAX stat=FileExists(filename\$)

PARAMETERS filename\$ - STRING.Pathname of file to verify exists.

USE Determine if filename\$ exists and can be accessed.

RULES None.

NOTES This function returns -1 if the file exists. Other return values are DOS error codes. For instance, if the file is currently locked, 5 is returned (access denied). To determine if a file exists or not, check for a return value of -1 (exists and is accessible), or 2 (file does not exist).

RETURN -1 if file is accessible,
 2 if file does not exist,
 or other DOS errors.

EXAMPLE stat=FileExists("C:\HIST\AR91.KEY")
 IF stat=2 THEN
 DoCreateNewKeyFile
 ELSEIF stat=-1 THEN
 DoFileAlreadyExistsError
 ELSE
 DoHandleFileAccessError stat

GetDefaultDrive

TYPE FUNCTION

SYNTAX CurrDrive=GetDefaultDrive

PARAMETERS None.

USE Determine the current default DOS drive.

RULES None.

NOTES This function returns the ASCII representation of the
current default DOS drive.
For instance, if the current drive is C:,
GetDefaultDrive returns the ASCII representation of C, or
67. To convert this to a string use CHR\$().

RETURN ASCII representation of current DOS drive letter.

EXAMPLE CurrDrive\$=CHR\$(GetDefaultDrive)
PRINT "Current drive=";CurrDrive\$+":"

GetDiskInfo

TYPE SUB

SYNTAX GetDiskInfo(drive\$,ACL,MCL,BPS,SPCL,freebytes&)

PARAMETERS drive\$ - STRING.Drive letter to get disk info from (first character significant).
ACL - INTEGER.The number of available clusters on drive\$ is placed in this variable.
MCL - INTEGER.The number of total, or maximum, clusters on drive\$ is placed in this variable.
BPS - INTEGER.The number of bytes per sector for drive\$ is placed in this variable.
SPCL - INTEGER.The number of sectors per cluster for drive\$ is placed in this variable.
freebytes& - LONG.The number of free bytes left on drive\$ is placed in this variable.

USE Obtain information for the given DOS drive.

RULES None.

NOTES This function uses DOS function 36h to get information on drive\$.
For information on the default drive use drive\$ = "@".
freebytes& = -1 if drive\$ is not a valid DOS drive.

RETURN None.

EXAMPLE GetDiskInfo("C:",ACL,MCL,BPS,SPCL,freebytes&)
PRINT "Bytes remaining on drive C:=";freebytes&

GetDosVersion

TYPE FUNCTION

SYNTAX DOSver=GetDosVersion

PARAMETERS None.

USE Determine the DOS version in use.

RULES None.

NOTES This function returns the DOS version that your program
 is running on x100.
 For instance, if the system is running on DOS 3.31
 GetDosVersion returns 331.

RETURN DOS version x100.

EXAMPLE DOSver=GetDosVersion
 PRINT USING "DOS version ##.##";DOSver/100

GetXEInfo

TYPE FUNCTION

SYNTAX Xerror=GetXEInfo(class,action,locus)

PARAMETERS class - INTEGER.The error class code is placed in this variable.
 action - INTEGER.The recommended action code is placed in this variable.
 locus - INTEGER.The error locus code is placed in this variable.

USE Obtain detailed error information after a previous DOS error.

RULES Requires DOS 3+.

NOTES Use this function after a QBTree function returns a DOS error.
 See Appendix D, DOS Error Codes, for a listing of all DOS error, class, action, and locus code descriptions.

RETURN Extended DOS error code.

EXAMPLE stat=OpenKeyFile(filename\$,0)
 IF stat>0 AND stat<200 THEN
 Xerr=GetXEInfo(class,action,locus)
 PRINT "Extended error:";ExtendErrMsg\$(Xerr)
 PRINT "Error class:";ClassErrMsg\$(class)
 PRINT "Recommend:";RecommendMsg\$(action)
 PRINT "Error locus:";LocusMsg\$(locus)

LoadDataHeader

TYPE FUNCTION

SYNTAX stat = LoadDataHeader(dfile)

PARAMETERS dfile - INTEGER.Number used as fileno in OpenDataFile().

USE Network. Load from disk the data header of dfile.

RULES If there is any possibility that another process has changed the data file in such a manner as to alter the data header since the open, you must call this function before performing any R/W operation on the data file.

NOTES This function is not needed if dfile's asmode=&H12. This function is needed only if the data file is being used on a network and there's a possibility that another process altered the data header. For instance, let's say you open dfile but don't lock it. Another process (program) on the network also opens it, adds 10 new records to it, and closes it. Now, the data header you have in RAM (loaded when dfile was opened) no longer matches the data header on disk. Before reading or writing to dfile you need to LoadDataHeader() to obtain the valid header data. This function is called by LockDataHeader.

RETURN Errors 219, 222, and DOS.

EXAMPLE stat = LoadDataHeader(0)

LoadKeyHeader

TYPE FUNCTION

SYNTAX stat = LoadKeyHeader(kfile)

PARAMETERS kfile - INTEGER.Number used as fileno in OpenKeyFile().

USE Network. Load from disk the key header of kfile.

RULES If there is any possibility that another process has changed the key file in such a manner as to alter the key header since the open, you must call this function before performing any R/W operation on the key file.

NOTES This function is not needed if kfile's asmode=&H12. This function is needed only if the key file is being used on a network and there's a possibility that another process altered the key header. For instance, let's say you open kfile but don't lock it. Another process (program) on the network also opens it, adds 10 new keys to it, and closes it. Now, the key header you have in RAM (loaded when kfile was opened) no longer matches the key header on disk. Before reading or writing to kfile you need to LoadKeyHeader() to obtain the valid header data. This function is called by LockKeyFile(). If you were sequentially accessing the file using GetNext() or GetPrev() and you UnlockKeyFile() you will need to maintain the last key found so you can restore to that key when you continue processing.

RETURN Errors 219, 222, and DOS.

EXAMPLE stat = LoadKeyHeader(0)

SFTFiles

TYPE FUNCTION

SYNTAX MaxFiles=SFTFiles

PARAMETERS None.

USE Determine the number of file slots allocated in the
System Files Table.

RULES None.

NOTES This function returns the number after FILES= in
CONFIG.SYS.
For instance, if you have FILES=30 in your CONFIG.SYS
file, SFTFiles() returns 30.
This uses an undocumented DOS function. See the BASIC
source for more.

RETURN The maximum number of DOS system-wide handles.

EXAMPLE MinFiles=40+4
Files=SFTFiles
IF Files<MinFiles THEN
 PRINT "Set FILES=44 in CONFIG.SYS"

CreateFile

TYPE FUNCTION

SYNTAX stat>CreateFile(filenameZ\$,attribute)

PARAMETERS filenameZ\$ - STRING.Pathname of file to create.
attribute - INTEGER, BYVAL.DOS directectory attribute to
give the file.

USE Create a new file using DOS function 3Ch.

RULES filenameZ\$ must not already exist.
filenameZ\$ must end with a CHR\$(0).

NOTES By using these file functions you can save 15K in EXE
file size by not needing BASIC's file functions.
attribute can take the following values:
 0 normal
 1 read-only
 2 hidden
 4 system

RETURN -1 if filenameZ\$ either has zero length or is not ASCIIIZ,
Errors DOS.

EXAMPLE filename\$ = "F:\AR91\CONFIG.QBT"
stat>CreateFile(filename\$+CHR\$(0),0)

OpenDevice

TYPE FUNCTION

SYNTAX stat=OpenDevice(filenameZ\$,mode,handle,flen&)

PARAMETERS filenameZ\$ - STRING.Pathname of file/device to open.
mode - INTEGER, BYVAL.Open mode.
handle - INTEGER.The DOS handle for this file is placed
in this variable.
flen& - LONG.The length of the file at time of open is
placed in this variable.

USE Open a file/device using DOS function 3Dh.

RULES filenameZ\$ must exist.
filenameZ\$ must end with a CHR\$(0).

NOTES By using these file functions you can save 15K in EXE
file size by not needing BASIC's file functions.
mode values are (bit-mapped):
bit 7 = inheritance flag (DOS 3+)
 4-6 = sharing mode (DOS 3+)
 3 = reserved (should=0)
 0-2 = access type
For read/write non-network access to a file use mode=2.
For exact uses of mode consult a DOS programmer reference.

RETURN -1 if filenameZ\$ either has zero length or is not ASCIIIZ,
Errors DOS.

EXAMPLE stat=OpenDevice("TEST.DAT"+CHR\$(0),2,han,flen&)
IF stat=0 THEN
 PRINT "DOS handle=";han
 PRINT "File length=";flen&

ReadDevice

TYPE FUNCTION

SYNTAX stat=ReadDevice(handle,start&,bytes&,vseg,voff)

PARAMETERS handle - INTEGER, BYVAL.DOS handle to read from.
 start& - LONG, BYVAL.If block device then byte offset
 into file to start reading from.
 bytes& - LONG, BYVAL.Number of bytes to read from the
 device.
 vseg - INTEGER, BYVAL.Segment of buffer.
 voff - INTEGER, BYVAL.Offset of buffer.

USE Read from a DOS device to RAM using DOS function 3Fh.

RULES handle must be open.

NOTES The first byte is 0.
 To continue reading from the last position+1 use start&
 = -1.

By using these file functions you can save 15K in EXE
file size by not needing BASIC's file functions.
You can read/write to DOS's predefined handles without
first needing to open them (they're already open):

```
0 STDIN  (CON)
1 STDOUT (CON)
2 STDERR (CON)
3 STDAUX (AUX)
4 STDLST (PRN)
```

RETURN Errors DOS.

EXAMPLE REDIM buffer(1 TO 16384) AS INTEGER
 vs=VARSEG(buffer(1))
 vo=VARPTR(buffer(1))
 stat=OpenDevice("TEST.DAT"+CHR\$(0),2,han,flen&)
 'read 32K of the file at a time
 IF stat=0 THEN
 stat=ReadDevice(han,0,32768,vs,vo)
 DoProcessBuffer vs,vo
 DO WHILE stat=0
 vs=vs+(32768\16)
 stat=ReadDevice(han,-1,32768&,vs,vo)
 DoProcessBuffer vs,vo
 LOOP

WriteDevice

TYPE FUNCTION

SYNTAX stat=WriteDevice(han,start&,bytes&,vseg,voff)

PARAMETERS handle - INTEGER, BYVAL.DOS handle to write to.
 start& - LONG, BYVAL.If block device then byte offset
 into file to start writing to.
 bytes& - LONG, BYVAL.Number of bytes to write to the
 device.
 vseg - INTEGER, BYVAL.Segment of buffer.
 voff - INTEGER, BYVAL.Offset of buffer.

USE Write from RAM to a DOS device using DOS function 40h.

RULES handle must be open.

NOTES The first byte is 0.
 To continue writing to the last position+1 use start& =
 -1.

By using these file functions you can save 15K in EXE
file size by not needing BASIC's file functions.
You can read/write to DOS's predefined handles without
first needing to open them (they're already open):

```
0 STDIN  (CON)
1 STDOUT (CON)
2 STDERR (CON)
3 STDAUX (AUX)
4 STDLST (PRN)
```

RETURN Errors DOS.

EXAMPLE REDIM buffer(1 TO 16384) AS INTEGER
 vs=VARSEG(buffer(1))
 vo=VARPTR(buffer(1))
 stat=OpenDevice("TEST.DAT"+CHR\$(0),2,han,flen&)
 'write 5 32K blocks of the file
 FOR i=1 TO 5
 DoSetupBuffer vs,vo
 stat=WriteDevice(han,-1,32768&,vs,vo)
 IF stat THEN EXIT FOR
 vs=vs+(32768\16)
 NEXT

CloseDevice

TYPE FUNCTION

SYNTAX stat=CloseDevice(handle)

PARAMETERS handle - INTEGER.The DOS handle to close.

USE Close a DOS device using DOS function 3Eh.

RULES Handles 0-4 cannot be closed.

NOTES By using these file functions you can save 15K in EXE file size by not needing BASIC's file functions.

RETURN Errors DOS.

EXAMPLE stat=OpenDevice("TEST.DAT"+CHR\$(0),2,han,flen&)
IF stat=0 THEN
 PRINT "DOS handle=";han
 PRINT "File length=";flen&
 stat=CloseDevice(han)

DeleteFile

TYPE FUNCTION

SYNTAX stat>DeleteFile(filenameZ\$)

PARAMETERS filenameZ\$ - STRING.Pathname of file to delete.

USE Delete a file using DOS function 41h. Similar to BASIC's KILL.

RULES filenameZ\$ must exist.
 filenameZ\$ must end with a CHR\$(0).

NOTES By using these file functions you can save 15K in EXE file size by not needing BASIC's file functions.

RETURN -1 if filenameZ\$ either has zero length or is not ASCIIIZ, Errors DOS.

EXAMPLE IF FileExists(fileZ\$) = -1 THEN
 stat>DeleteFile(fileZ\$)

RenameFile

TYPE FUNCTION

SYNTAX stat=RenameFile(oldfileZ\$,newfileZ\$)

PARAMETERS oldfileZ\$ - STRING.Pathname of original file.
 newfileZ\$ - STRING.Filename to replace original with.

USE Rename a file using DOS function 56h. Similar to BASIC's
 NAME.

RULES oldfileZ\$ must exist. newfileZ\$ must not exist.
 oldfileZ\$ and newfileZ\$ must end with a CHR\$(0).

NOTES By using these file functions you can save 15K in EXE
 file size by not needing BASIC's file functions.

RETURN -1 if oldfileZ\$ or newfileZ\$ either have zero length or
 aren't ASCIIZ,
 Errors DOS.

EXAMPLE IF FileExists(fileZ\$) = -1 THEN
 stat=RenameFile(fileZ\$,newfileZ\$)

MemCopy

TYPE SUB

SYNTAX MemCopy (fromseg, fromoff, toseg, tooff, bytes)

PARAMETERS FromSeg - INTEGER, BYVAL. The source segment to copy from.
FromOff - INTEGER, BYVAL. The source offset to copy from.
ToSeg - INTEGER, BYVAL. The destination segment to copy to.
ToOff - INTEGER, BYVAL. The destination offset to copy to.
bytes - INTEGER, BYVAL. The number of bytes to copy.

USE Copy memory from/to QB TYPE variable to/from a QBTree variable-length string.

RULES Memory addresses must not overlap.

NOTES None.

RETURN Errors DOS.

EXAMPLE 'See APPENDIX C. Using TYPE Variables.

APPENDIX B. TECHNICAL SPECIFICATIONS

Technical Specifications:

Key length : 1-64 bytes (ASCII sort), fixed-length
Record length: 3-2048 bytes, fixed-length (to 32767 with source)
Node size : 512 bytes
Keys per node: 7-84 keys, $(512-3) \backslash (\text{key length}+5)$
Max keys/file: 5.5 million keys (65534 nodes)
Max recs/file: 16.7 million records
Max key files: 250 (total combined files ≤ 250 DOS 3+)
Max data file: 250 (total combined files ≤ 15 DOS 2)

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

filetype char ; 0 file type, "*"
rootnode int ; 1 sector in file of root node
nokeyslo int ; 3 number of keys (low word)
nokeyshi byte ; 5 - (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 byte ; 12-511 internal use

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 - (high byte)
datavlo int ; 6 data available list (low word)
datavhi byte ; 8 - (high byte)
nxdalo int ; 9 next data record avail (low word)
nxdahi byte ; 11 - (high byte)
internal byte ; 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:

$\text{DataAvailRec} = 1 \& * \text{datavlo} + (\text{datavhi} * 65536)$

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 is a count key byte. This is the count of valid keys on that node. 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
4. The 24-bit data record pointer (low word/hi byte)
5. Forward ptr/back ptr (for non-leaf nodes)
6. First logical key (here 'KEY001')
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 points 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.

Resource Allocation:

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 QBTtree. 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 64 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 QBTtree low-level, 34 bytes.
- ... and other miscellaneous variables totalling about 2500 bytes.

APPENDIX C. WORK AROUNDS.

Using TYPE Variables:

QBTree uses a simple variable, namely a variable-length string, to get and put data in the data file. To use QB TYPE variables, you may want to use this method: Setup your TYPE variable, then allocate enough space in a variable-length 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$(LEN(SSNage))
FromSeg = VARSEG(SSNage)
FromOff = VARPTR(SSNage)
ToSeg = VARSEG(work$)      'QBX/Fs use SSEG
ToOff = SADD(work$)       'string address
count = LEN(work$)
MemCopy FromSeg,FromOff,ToSeg,ToOff,count
```

'You now have the TYPE variable data in a var-len
'string that can be used as the Qrec\$.

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

'To put Qrec\$ in the typed variable,
'reverse the From/To assignments:

```
stat = GetEqual(0,0,Qkey$,Qrec$)
FromSeg = VARSEG(Qrec$) 'QBX/Fs use SSEG
FromOff = SADD(Qrec$)
ToSeg = VARSEG(SSNage)
ToOff = VARPTR(SSNage)
count = LEN(SSNage)
MemCopy FromSeg,FromOff,ToSeg,ToOff,count
PRINT SSNage.SSN; SSNage.age
```

'MemCopy is included in the QBTreeEx.LIB and QBTree.BI.
'The copy goes from low memory to high.

Non-unique Keys:

QBTREE doesn't allow duplicate keys (this may be an inconvenience in some database 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. A well-designed relational database would not have this problem by definition, but this work-around is presented just in case you need it.

When retrieving 'identical' enumerated keys you normally would need to start at the first enumerated key with `GetEqual()` using an enumerator of 0 and continue searching using `GetNext()` until the desired record is found.

Note: For proper sorting, you must have the most significant byte of the enumerator first and the least significant last.

Note: Retrieved keys and records will be returned with their full length retained. Be sure the enumerator occupies the last 2 bytes. Don't just append the enumerator onto a key.

This code accesses an existing key/record (`Qkey$/Qrec$`) from `kfile=0` `dfile=0` and adds a new key to `kfile=1`. This new key in `kfile=1` will index the same data record in `dfile=0` as `Qkey$` in `kfile=0`. Similar code could be used when using `AddKeyRecord()`.

```
DIM LV AS STRING * 2, HV AS STRING * 2
LV=CHR$(0)+CHR$(0)      'lowest enumerator
HV=CHR$(255)+CHR$(255) 'highest enumerator
Qkey$="TESTKEY-K0"+LV   '10-byte key+2-byte enumerator
NewKey$="NEW_KEY-K1"    '10-byte key, no enumerator yet
stat=GetEqual(0,0,Qkey$,Qrec$)
'accessed test key, now add new key to kfile=1
IF stat = 0 THEN
    stat=AddKey(1,0,NewKey$+LV) 'use LV enumerator
ELSE
    STOP
ENDIF
IF stat = 201 THEN
    'NewKey$+LV exists so reset data pointer
    stat=GetEqual(0,0,Qkey$,Qrec$)
    'get absolute last NewKey$ (won't be found, stat=200)
    stat=GetEqual(1,0,NewKey$+HV,Zrec$)
    'backup and get the last valid one, Zkey$=key and enumerator
    stat=GetPrev(1,0,Zkey$,Zrec$)
    'enumerator is in little-Endian order so reverse bytes,
    'make it a number, add 1, convert back to string,
    'little-Endian it, and add the enumerated key
    enum$ = RIGHT$(ZKey$,2)
    enum$ = RIGHT$(enum$,1)+LEFT$(enum$,1)
    enum = CVI(enum$)
    enum = enum + 1
    enum$ = MKI$(enum)
    enum$ = RIGHT$(enum$,1)+LEFT$(enum$,1)
    stat = AddKey(1,0,NewKey$+enum$)
```

APPENDIX D. ERROR CODES.

QSAM generated error codes (200-218):

- 0 No error - the operation was successful.
- 200 Key not found - the key is not in the index file.
- 201 Key already exists - duplicate keys are not allowed in QBTREE. If this error occurs with AddKey(), you must re-establish a valid data pointer by re-accessing the original key/record before trying again.
- 202 End of file - reached the end of file.
- 203 Top of file - reached the top of file.
- 204 Empty file - there are no keys in the index file.
- 205 Reserved - previously disk full error. This error is now indicated by DOS error -2 - disk full or unexpected end of file.
- 206 Data pointer invalid - a valid QBTREE key access must occur right before a data record can be UpdateRecord() or a key can be AddKey().
- 207 Reserved.
- 208 Node exceeds limit - more than 65534 nodes used by index file.
- 209 Record exceeds limit - more than 16777215 records used by data file.
- 210 Corrupt index - index file is probably corrupt.
- 211 Invalid function - an invalid QSAM function was called.
- 212-218 Reserved.

BTREE.BAS generated error codes (219-239):

- 219 File number invalid - the file number used is greater than the highest allocated in InitQBTREE().
- 220 Data record length invalid - length must be from 3-2048 bytes when creating a QBTREE 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 open.
- 223 Invalid null key assignment - a null key cannot be used. Null is defined in QBTREE as either ASCII 0 or ASCII 255 in the first

character of a key.

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

225 No handles available - no DOS handles are available. Use a greater value in config.sys FILES= (FILES=20). Max FILES=255 with DOS 3+ or FILES=20 with DOS 2.

226 Reserved.

227 File number already open - the fileno is already in use.

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

229 Reserved.

230 File already exists - the CreateFile() filename already exists.

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

232 Reserved.

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

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

235-255 Reserved.

Error numbers are not exclusively generated by the module indicated. It is possible for BTREE to issue error 208, for instance.

DOS Error Codes:

Possible stat return codes in decimal.

- 2 disk full or unexpected end of file
- 1 bad filename
- 0 no error
- 1 function not supported
- 2 file not found
- 3 path not found
- 4 too many open files
- 5 access denied
- 6 handle invalid
- 7 MCBs destroyed
- 8 not enough memory
- 9 memory block address invalid
- 10 environment invalid
- 11 format invalid
- 12 access code invalid
- 13 data invalid
- reserved-0Eh
- 15 disk drive invalid
- 16 cannot remove current directory
- 17 not same device
- 18 no more files
- 19 disk write protected
- 20 unknown unit
- 21 drive not ready
- 22 unknown command
- 23 data error (CRC)
- 24 bad request structure length
- 25 seek error
- 26 unknown medium type
- 27 sector not found
- 28 printer out of paper
- 29 write fault
- 30 read fault
- 31 general failure
- 32 sharing violation
- 33 lock violation
- 34 disk change invalid
- 35 FCB unavailable
- 36 sharing buffer overflow
- reserved-25h
- reserved-26h
- reserved-27h
- reserved-28h
- reserved-29h
- reserved-2Ah
- reserved-2Bh

reserved-2Ch
reserved-2Dh
reserved-2Eh
reserved-2Fh
reserved-30h
reserved-31h
50 network request not supported
51 remote computer not listening
52 duplicate name on network
53 network name not found
54 network busy
55 network device no longer exists
56 NETBIOS command limit exceeded
57 network adapter hardware error
58 incorrect response from network
59 unexpected network error
60 incompatible remote adapter
61 print queue full
62 queue not full
63 not enough space to print file
64 network name deleted
65 access denied
66 network device type incorrect
67 network name not found
68 network name limit exceeded
69 NETBIOS session limit exceeded
70 temporarily paused
71 network request not accepted
72 print/disk redirection paused
reserved-49h
reserved-4Ah
reserved-4Bh
reserved-4Ch
reserved-4Dh
reserved-4Eh
reserved-4Fh
80 file exists
81 reserved-51h
82 cannot make
83 fail on INT24
84 out of structures
85 already assigned
86 invalid password
87 invalid parameter
88 network write fault

DOS Class Codes:

1 out of resources
2 temporary situation
3 authorization

- 4 internal
- 5 hardware failure
- 6 system failure
- 7 application error
- 8 not found
- 9 bad format
- 10 locked
- 11 media failure
- 12 already exists
- 13 unknown

DOS Action Codes:

- 1 retry
- 2 delay retry
- 3 reenter input
- 4 abort
- 5 immediate exit
- 6 ignore
- 7 user intervention

DOS Locus Codes:

- 1 unknown
- 2 block device
- 3 reserved
- 4 serial device
- 5 memory