PowerTREE for Windows

Table of contents

Home

PowerTree
BTree Manager for Windows

http://www.powerbasic.com

About PowerTree for Windows

About PowerTree for Windows

If you like "smaller and faster", PowerTree is just what you need! PowerTree offers a concise, proprietary implementation of the highly efficient B+Tree algorithm for indexed data management. Create one index or many. Use one data file, or several, always in the format of your choice. There are no arbitrary limitations. Retrieve data sequentially, by exact match, or approximation. Support a single user, or hundreds, safely and reliably.

PowerTree is FAST! Index and search thousands of records in seconds. PowerTree is SMALL! But don't let that fool you... The amazing module size just means that you won't need a stack of CD's to distribute your application. The days of fat database BloatWare are over!

PowerTree creates compatible indexes for DOS, 16-bit Windows, 32-bit Windows, and 64-bit Windows. With no limitation on file format, any type of database can be indexed. With no limitation on keys, any number can be utilized, simple or compound. And with easy locking capability, multi-user and multi-threaded applications are a breeze.

Best of all, PowerTree is easy to implement. There are only fifteen functions to learn, not hundreds, like the others. Create or open an index, add new entries, delete or search for existing entries. It's as simple as that! And in Windows, the DLL interface is industry standard, for immediate, efficient use with almost any

programming language. PowerTree is for you!

See Also

Getting Started

Technical Support

Getting Started

Getting Started

Getting Started with PowerTree for Windows

PowerTree is an BTree Manager. You pass it a key value to index, and the record number where your data is stored in the data file. PowerTree then stores that key and record number in an index file. It does not manage the data itself. Your code must open and manage the data file. This allows for maximum flexibility, as you can index any type of data, whether it's stored in a file or an array, or is based on fixed-length or variable-length records.

Accessing the index is done through a set of easy-to-use functions which allow you to create a new index, open an existing index, add new index records, delete existing records, and search the index in various ways. In addition to the index functions, two functions are also supplied for creating a "soundex" code for a key value. Soundex codes are based on what a word sounds like, giving you a flexible way of searching for similar keys.

The following example code assumes you are using PowerBASIC For Windows (PB/Win) or the PowerBASIC Console Compiler for Windows (PB/CC). Accessing the PowerTree DLLs from other languages, such as Visual Basic, Access, Delphi or C/C++, would be done in a similar manner. Note that these examples lack the error checking you would expect of production code, so as to better highlight the fundamentals of PowerTree. See the samples on disk for a more robust approach.

IMPORTANT: When passing string values to PowerTree functions you must NOT pass a dynamic string. You must always pass a fixed-length or ASCIIZ string:

```
DIM s AS STRING * 30
DIM a AS ASCIIZ * 30
```

When using the search routines, you may get different results when passing a fixed-length string versus an ASCIIZ string. See each function reference for details. When using a string constant (eg. "SMITH") PowerBASIC passes the string as an ASCIIZ string. If you need to use a dynamic string, pass the address of the string BYVAL to pass it as an ASCIIZ string:

```
ptFF BYVAL STRPTR(KeyValue$), KeyBlock
```

See Also

Creating an index

Finding a key

Managing the Data File

Distributing Applications

Creating an index

Creating an index

The PTSAMPLE.DAT file contains 10,000 sample records with names and addresses (Important: The names and addresses were computer generated and are not real). Our first sample program, CREATE.BAS, will create three indexes. The first index file is built on the last name in each record, the second index file on the zip code for each record, and the third index file as a compound key with both the zip code and last name combined.

For a program to use PowerTree, we need to tell the compiler how to access the functions in the DLL. In PowerBASIC, we use the $INCLUDE meta-statement to include declarations for each function, and to define the structures (or user-defined types) used by the index manager:

```
$INCLUDE "PTREE.INC"
```

Next, we need to define a structure for the records in PTSAMPLE.DAT and create a variable which will be used to access each record in the file:

```
TYPE PTSampleREC
  Title         AS STRING * 6
  Lastname      AS STRING * 25
  Firstname     AS STRING * 20
  Middleinitial AS STRING * 1
  Address1      AS STRING * 30
  Address2      AS STRING * 30
  City          AS STRING * 25
  State         AS STRING * 2
  Zip           AS STRING * 9
  Comments      AS STRING * 60
END TYPE

DIM rec AS PTSampleREC
```

Now, we can open the file for random access:

```
hFile& = FREEFILE
OPEN "PTSAMPLE.DAT" FOR RANDOM AS hFile& LEN = SIZEOF(rec)
```

Remember, the first record number in a PowerBASIC random access file is 1 by default. Now we need to create an AccessBlock structure for PowerTree and create a new empty index file for the last name:

```
DIM KeyBlock AS AccessBlock
KeyBlock.MultiUser = 0
KeyBlock.KeyLength = 25              'length of the lastname field
ptCreateIndex "PTLAST.PTX", KeyBlock
ptInit "PTLAST.PTX", KeyBlock
```

You can use any filename you like for the index. In the 32-bit version, you can even use long filenames. The MultiUser member tells PowerTree whether or not the index will be accessed by more than one program at the same time. If it is turned on, there is a slight performance penalty as the index is locked and unlocked each time a new record is added. The KeyLength member tells PowerTree how big each key is. If possible, you should try to keep the key as small as possible, but not so small that you get too many matching records. The smaller the key, the more keys PowerTree can load into memory at one time and quickly search through them.

ptCreateIndex simply creates a new empty index file. If a file already exists, it is overwritten with the new empty file. ptInit opens the index for access by the other PowerTree routines. When you are finished with an index, close it with the ptClose function.

Now all that's left is to read each of the records in the data file and add the Lastname key to the index for each record:

```
recs& = LOF(hFile&) \ SIZEOF(rec) 'get the total number of records in the file
FOR x& = 1 TO recs&
  GET hFile&, x&, rec          'read a data record
  KeyBlock.RecordNumber = x&   'specify the record number
  ptAdd rec.LastName, KeyBlock 'add the key to the index
NEXT x&
ptClose KeyBlock 'close the index file
```

The GET statement in PowerBASIC reads a data record from a random access file. The RecordNumber

member is the record number in the data file. You can also use a physical location if you wish (such as an offset into the file). When you search through the index file, this is the value returned which lets you find the record in the data file. The ptAdd function adds the key to the index file.

PowerTree indexes strictly on ASCII text values, if you need to index on a numeric value you should convert it into a string using the STR$ or FORMAT$ functions in PowerBASIC. PowerTree will support up to 2,147,483,647 records (long integer positive range) per index.

Using the same technique, we can now index on the ZIP code member of the record:

```
KeyBlock.MultiUser = 0
KeyBlock.KeyLength = 9       'length of the zip member
ptCreateIndex "PTZIP.PTX", KeyBlock
ptInit "PTZIP.PTX", KeyBlock
FOR x& = 1 TO recs&
  GET hFile&, x&, rec        'read a data record
  KeyBlock.RecordNumber = x& 'specify the record number
  ptAdd rec.Zip, KeyBlock    'add the key to the index
NEXT x&
ptClose KeyBlock 'close the index file
```

It is also possible to index on more than a single key. This lets you distinguish between SMITH in the 93923 zip code and SMITH in the 90210 zip code, which helps to create a more specific index record and speeds up searches. Since the Lastname member is 25 bytes and the Zip member is 9 bytes, the total key size will be 34 bytes. We then simply combine the two strings together into a single string and pass the new string as the index key:

```
DIM combo AS STRING * 34
KeyBlock.MultiUser = 0
KeyBlock.KeyLength = 34          'length of the lastname + zip members
ptCreateIndex "PTCOMBO.PTX", KeyBlock
ptInit "PTCOMBO.PTX", KeyBlock
FOR x& = 1 TO recs&
  GET hFile&, x&, rec            'read a data record
  KeyBlock.RecordNumber = x&     'specify the record number
  combo = rec.Lastname + rec.Zip 'combine the two members into a single string
  ptAdd combo, KeyBlock          'add the key to the index
NEXT x&
ptClose KeyBlock                 'close the index file
CLOSE hFile&                     'close the data file
```

The "more unique" our index key is, the better chance we'll have of finding it on the first or second search. This can speed up searches dramatically.

See Also

- Getting Started
- Finding a key
- Managing the Data File
- Distributing Applications

Finding a key

Finding a key

The whole point of indexing data is, of course, so that you can quickly find specific information. And you won't find much faster than with PowerTree. FIND.BAS allows you to enter a last name, zip code or both and searches the appropriate index for a record. It then displays all information for the record it found.

Using the PTSAMPLE.DAT and index files created by CREATE.BAS, our next example program will allow

you to look up a record by last name, zip code, or both.

First we need to initialize all of our variables and open the data file and index file:

```
$INCLUDE "PTREE32.INC"

TYPE PTSampleREC
  Title         AS STRING * 6
  Lastname      AS STRING * 25
  Firstname     AS STRING * 20
  Middleinitial AS STRING * 1
  Address1      AS STRING * 30
  Address2      AS STRING * 30
  City          AS STRING * 25
  State         AS STRING * 2
  Zip           AS STRING * 9
  Comments      AS STRING * 60
END TYPE

DIM rec AS PTSampleREC
DIM KeyBlock AS AccessBlock
DIM i AS STRING * 34

' ** Open the data file
hFile& = FREEFILE
OPEN "PTSAMPLE.DAT" FOR RANDOM AS hFile& LEN = SIZEOF(rec)
recs& = LOF(hFile&)\SIZEOF(rec) 'get the total number of records
```

You may have noticed that the string variable i was created with a length of 34 bytes. If we look up a record by last name we only need 25 bytes. And if we look up a record by zip code we only need 9 bytes. But if we look up the complex key which is the last name plus the zip code, we need 34 bytes.

You may be wondering why we don't create three different fixed-length strings. Because, PowerTree knows how large a key is and will only use the number of bytes in a string that match the key size. So, if we are looking up a last name and adding the value "SMITH" to the 34-byte string i, only the first 25 bytes will be used by PowerTree. Any data after that is simply ignored. We use a fixed-length string instead of an ASCIIZ string because PowerBASIC pads fixed-length strings with spaces, making the data easier to read and more compatible with other platforms such as DOS where ASCIIZ strings might not be supported.

Next, your program needs to find out how the data is to be looked up. By last name, zip code, or both? In FIND.BAS we place the letter "L", "Z", or "B" in the variable a$ to determine which index to use. Once this is known, we can open the appropriate index and look up the data.

```
KeyBlock.MultiUser = 0
SELECT CASE a$
  CASE "B", "b"
    KeyFile$ = "PTCOMBO.PTX"
  CASE "L", "l"
    KeyFile$ = "PTLAST.PTX"
  CASE "Z", "z"
    KeyFile$ = "PTZIP.PTX"
END SELECT

' ** Initialize the index
ptInit KeyFile$, KeyBlock

' ** Check for an error
IF ISTRUE KeyBlock.ErrType THEN
  EXIT FUNCTION
END IF
```

After calling ptInit to open the index file, it's a good practice to check the ErrType member to see if an error occurred. This might be due to an invalid filename or path, or the index file itself might be corrupted.

Now that the proper index file is open, we need the data to look up. Once we have that, we assign it to the i variable discussed earlier and call the ptFF function to perform the search. The data in PTSAMPLE.DAT is all uppercase, so don't forget to convert the data typed by the user into uppercase before we do the search. Note that PowerTree itself does no case conversion and is case-sensitive, so when indexing data, you may prefer to convert the key value to uppercase beforehand.

```
' ** Search for the key
i = UCASE$(i) 'convert it to upper case
ptFF i, KeyBlock
```

Again, checking the ErrType member will tell us if any data was located. If no error, we can read the record from the data file. The record number is found in the RecordNumber member:

```
IF ISFALSE KeyBlock.ErrType THEN
  GET hFile&, KeyBlock.RecordNumber, rec
END IF
```

Now, let's say that you searched for the last name of SMITH. There might be a hundred SMITHs in your database. Once you've located the first SMITH, using the ptFF function, you can call the ptFN function to find the rest:

```
DO
  ptFN i, KeyBlock
  IF ISTRUE KeyBlock.ErrType THEN
    EXIT DO
  END IF

  GET hFile&, KeyBlock.RecordNumber, rec
  ' your code goes here
LOOP
```

When the ErrType member returns nonzero, you're all out of SMITHs. Don't forget to close the index and the data file when you're done:

```
ptClose KeyBlock
CLOSE hFile&
```

See Also

Getting Started

Creating an index

Managing the Data File

Distributing Applications

Managing the Data File

Managing the Data File

As you can see, using PowerTree to index records for you is very easy. But what about the data file? PowerTree doesn't do anything with that, so how do you create one and add records to it? The key is the recs& variable from the previous two examples.

When you need to add a record to a data file, you simply increment (add one to) the recs& variable and use that for your new record number:

```
INCR recs&
PUT hData&, recs&, rec    'add the new record to the data file
KeyBlock.RecordNumber = recs&
ptAdd KeyValue$, KeyBlock 'add the new key to the index file
```

If you are working with a multi-user database, it is important that recs& be up to date or you could be overwriting a new record added by another user. A safer method might be:

```
LOCK hFile& 'for multi-user only
```

```
recs& = (LOF(hFile&) \ SIZEOF(rec)) + 1
UNLOCK hFile&
```

A much more complex issue is deleting records and re-using deleted record entries. In most database programs, when you delete a record, the record isn't actually removed from the data file. Instead, it's marked as deleted (usually by changing the first letter in the last name to a CHR$(255) or something else that can be readily identified), and a separate index is used to track all deleted records so that they can be re-used. One advantage to this method is that deleted records can be recovered as long as they haven't been re-used yet. Actually removing a record from a data file involves shifting all the records that follow down one record number and trimming the last record off the end, then completely re-indexing the entire data file. This is a time-consuming operation, and one that you wouldn't want to do often.

Multi-user Programming

When you set the MultiUser member of the KeyBlock variable to non-zero, PowerTree automatically uses file sharing and locking internally when it accesses the index. This is to prevent two different programs (or threads) from writing data to the index at the same time, which can causes corrupted records. In most cases, this should be sufficient to allow your data and indexes to be used by more than one program. However, there may be times when you want to lock records in your data file to prevent them from being overwritten.

The following function will add a record to a data file and index with full record locking support, including a timeout value. If the function can't lock the record after 10 tries, it aborts with an error:

```
FUNCTION AddRecord&(BYVAL hFile&, KeyBlock AS AccessBlock, rec AS MyRecord, BYVAL
key$)
  timeout& = 10
  DO
    DECR timeout&

    IF timeout& = 0 THEN
      FUNCTION = 0 'not successful
      EXIT FUNCTION
    END IF

    LOCK hFile&, KeyBlock.RecordNumber
  LOOP UNTIL ERRCLEAR = 0

  PUT hFile&, KeyBlock.RecordNumber, rec
  ptAdd Key$, KeyBlock
  UNLOCK hFile&, KeyBlock.RecordNumber
  FUNCTION = -1 'success
END FUNCTION
```

It is important that the record only be locked long enough to write the data to the file and update the index. Otherwise, you'll cause other users to timeout waiting for you to release the record. Remember, once a record is locked it can't be read from or written to by other programs.

See Also

Getting Started

Creating an index

Finding a key

Distributing Applications

Distributing Applications

Distributing Applications

Once your application is complete and you are ready to distribute it to other users, what should you include?

The only file(s) you need to distribute along with your application are the PTREE16.DLL and PTREE32.DLL files. The PTREE16.DLL only needs to be distributed with 16-bit Windows applications, and PTREE32.DLL only needs to be distributed with 32-bit Windows applications.

IMPORTANT: The source code and header/include files for PowerTree are proprietary and may not be distributed in any way without the expressed written consent of PowerBASIC, Inc. Only the PTREE16.DLL and PTREE32.DLL files may be distributed with your products.

If you have any questions or doubts about what you can distribute, please contact our sales department.

See Also

- Getting Started
- Creating an index
- Finding a key
- Managing the Data File

Function Reference

Function Reference

PowerTree Function Reference

AccessBlock	User-Defined Type used to track all information about an index.
ptCreateIndex	Create a new index. If an index with the same name already exists, overwrite it with a new empty index.
ptInit	Open an existing index file.
ptAdd	Add a key value to an open index.
ptDel	Delete a key value from an open index.
ptFF	Find the first occurrence of a key value in an index.
ptFFa	Find the first key value after the search key in an index.
ptFN	Find the next occurrence of a key value in an index.
ptFP	Find the previous occurrence of a key value in an index.
ptFL	Find the last occurrence of a key value in an index.
ptFLb	Find the last occurrence of a key in an index, before the specified key value.
ptFR	Find an exact match of a key value and record number in an index.
ptKey	Return the key value associated with the current key.
ptClose	Close an open index file.
cSoundex	Return a 4 byte "soundex code" for a string value.
nSoundex	Return an all-numeric 4 byte "soundex code" for a string value.

AccessBlock

AccessBlock User-Defined Type

The AccessBlock UDT is used by PowerTree to store information about the index.

```
TYPE AccessBlock
  MultiUser      AS LONG
  KeyFile        AS LONG
  KeyLength      AS LONG
  Reserved       AS LONG
  NumberOfBlocks AS LONG
  WhichBlock     AS LONG
  WhichIndex     AS LONG
  IndexBlock     AS LONG
  RecordNumber   AS LONG
  NextRecord     AS LONG
  PrevRecord     AS LONG
  WasDeleted     AS LONG
  HoldLocks      AS LONG
  ErrType        AS LONG
END TYPE
```

MultiUser	Set this to 0 if running in single-user mode, and to 1 if running in multi-user (network) mode.
KeyFile	This holds the file handle used internally by PowerTree and must not be used by your code.
KeyLength	In ptCreateIndex you set KeyLength equal to the maximum number of bytes in the key. This must be the maximum number of bytes when the key is in string form.
Reserved	Internal use only! Do not set this yourself.
NumberOfBlocks	Internal use only! Do not set this yourself.
WhichBlock	Internal use only! Do not set this yourself.
WhichIndex	Internal use only! Do not set this yourself.
IndexBlock	Internal use only! Do not set this yourself.
RecordNumber	The number of the current record. This may be set by you, or may return the record number matched by any of the search routines. When inserting a key value into an index file, you should set RecordNumber equal to the current record number being indexed.
NextRecord	Internal use only! Do not set this yourself.
PrevRecord	Internal use only! Do not set this yourself.
WasDeleted	Internal use only! Do not set this yourself.
HoldLocks	Internal use only! Do not set this yourself.
ErrType	Returns 0 if the operation succeeded, or a non-zero value if it failed. If the value is 70, a record-locking operating failed in the index. If the value is -1, the requested operation failed (find next and there was no matching record, find first and the record did not exist, etc.) The value returned by any of the indexing functions is the same as ErrType.

See Also

Function Reference

CSoundex

cSoundex

Declare `FUNCTION cSoundex(x AS ASCIIZ) AS STRING`

Purpose Returns a 4 byte "soundex code" for a given string. This code will consist of the first character of the original string, followed by up to three digits.

A soundex code is a string that describes how a word "sounds", and is useful for finding words which are spelled and/or sound alike. For example, "SMITH" and "SMYTHE" both return the same soundex code. There are limitations to this technique, however, and some words that don't actually sound alike will end up having the same soundex code. For this reason, soundex codes are generally used to help narrow the range of most likely selections before presenting a "pick list" to a user.

Example

```
DIM s AS STRING
s = cSoundex("SMITH")
```

nSoundex

nSoundex

Declare

```
FUNCTION nSoundex(x AS ASCIIZ) AS STRING
```

Purpose Return a 4 byte "soundex code" for a given string. This code will consist of up to four digits.

A soundex code is a string that describes how a word "sounds", and is useful for finding words which are spelled and/or sound alike. For example, "SMITH" and "SMYTHE" both return the same soundex code. There are limitations to this technique, however, and some words that don't actually sound alike will end up having the same soundex code. For this reason, soundex codes are generally used to help narrow the range of most likely selections before presenting a "pick list" to a user.

Example

```
DIM s AS STRING
s = nSoundex("SMITH")
```

ptAdd

ptAdd

Declare

```
FUNCTION ptAdd(KeyValue AS ANY, x AS AccessBlock) AS LONG
```

Purpose Add a key value to an open index. When adding a key value you need to specify if the record number in your data file where the data record is stored in the *RecordNumber* member of the AccessBlock UDT. If you pass an ASCIIZ string to ptAdd, it will automatically be converted into a fixed-length string in PowerTree and padded with the proper number of spaces to meet the *KeyLength*.

KeyValue may be a fixed-length string or ASCIIZ string.

If your data file has been opened as a Random Access file with a fixed length record size, *RecordNumber* should reflect the random access record number in the data file. If your data file has been opened for Output (text file) or as a Binary file with variable length data records, *RecordNumber* should reflect the offset within the file where the record is stored (use the SEEK function in PowerBASIC). The important thing is that *RecordNumber* should contain a value you can use to find the data record in your data file.

Returns The ptAdd function returns zero if it was successful, or non-zero if an error occurred.

Example

```
KeyBlock.RecordNumber = 1
ptAdd "SMITH", KeyBlock
```

ptClose

ptClose

Declare `FUNCTION ptClose(x AS AccessBlock) AS LONG`

Purpose Close an open index.

This should be called when you are finished using an index. Note that you CANNOT share an AccessBlock value for one open index with another open index: each open index must have its own AccessBlock.

Returns The ptClose function returns zero if it was successful, or non-zero if an error occurred.

Returns The ptClose function returns zero if it was successful, or non-zero if an error occurred.

Example

```
ptClose KeyBlock
```

ptCreateIndex

ptCreateIndex

Declare `FUNCTION ptCreateIndex(Filename AS ASCIIZ, x AS AccessBlock) AS LONG`

Purpose Create a new index file.

If an index file of the same name already exists, it is overwritten with the new empty index. When initializing a new index you need to specify the size of the index key in the *KeyLength* member of the AccessBlock UDT. ptCreateIndex does not leave the index open after it is created. You must call the ptInit function to open the index and access it.

Returns The ptCreateIndex function returns zero if it was successful, or non-zero if an error occurred.

Example

```
DIM KeyBlock AS AccessBlock
KeyBlock.KeyLength = 35
ptCreateIndex "MYINDEX.PTX", KeyBlock
```

ptDel

ptDel

Declare `FUNCTION ptDel(KeyValue AS ANY, x AS AccessBlock) AS LONG`

Purpose Delete a key value in an open index.

KeyValue may be a fixed-length string or ASCIIZ string.

Before you can delete a key value, you must first use one of the ptFx functions to find the record, in order to fill in the proper information in the AccessBlock UDT. Once located, the index record specified in the AccessBlock UDT is deleted by the ptDel function. The ptFR function will locate an index record with the specified key value and data record number.

Returns The ptDel function returns zero if it was successful, or non-zero if an error occurred.

Example

```
KeyBlock.RecordNumber = 5 'delete the 5th data record
```

```
ptFR "SMITH", KeyBlock 'locate the index record
ptDel "SMITH", KeyBlock 'now delete it
```

ptFF

ptFF

Declare

```
FUNCTION ptFF(KeyValue AS ANY, x AS AccessBlock) AS LONG
```

Purpose

Find the first occurrence of a key value in an index. If you specify a null string (zero length) as the key value, *RecordNumber* will return the first key value in the index.

KeyValue may be a fixed-length string or ASCIIZ string.

If successful, the record number for your data file can be found in the *RecordNumber* member of the AccessBlock UDT.

If the key value is not located in the index, the *RecordNumber* member of AccessBlock will indicate the position of the next higher key (or, if there is no higher key, the next lower key).

If you want to search for an exact match, pass a fixed-length string that is the same length as the *KeyLength* member:

```
DIM x AS STRING * 25 ' assuming KeyBlock.KeyLength = 25
x = "SMITH"
ptFF x, KeyBlock
```

If you want to search for a partial string, pass the value as an ASCIIZ (NUL-terminated) string:

```
DIM x AS ASCIIZ * 25
x = "SMITH"
ptFF x, KeyBlock
```

Returns

The ptFF function returns zero if it was successful, or non-zero if an error occurred or if no matching key was found.

Example

```
ptFF "SMITH", KeyBlock 'locate SMITH in the index
IF ISFALSE KeyBlock.ErrType THEN
  GET hFile&, KeyBlock.RecordNumber, rec
END IF
```

ptFFa

ptFFa

Declare

```
FUNCTION ptFFa(KeyValue AS ANY, x AS AccessBlock) AS LONG
```

Purpose

Find the first occurrence of a key after the specified key value in an index.

KeyValue may be a fixed-length string or ASCIIZ string.

If there are three records in your index:

```
SMITH
SMITH
THOMPSON
```

And you search for "SMITH" using ptFFa, it will return the record number for "THOMPSON" which is the very next index record after the end of the SMITH's.

Returns

The ptFFa function returns zero if it was successful, or non-zero if an error occurred or no matching key was found.

Example

```
ptFFa "SMITH", KeyBlock
IF ISFALSE KeyBlock.ErrType THEN
  GET hFile&, KeyBlock.RecordNumber, rec
END IF
```

ptFL

ptFL

Declare

```
FUNCTION ptFL(KeyValue AS ANY, x AS AccessBlock) AS LONG
```

Purpose

Find the last occurrence of a key value in an index.

KeyValue may be a fixed-length string or ASCIIZ string.

If your index contains three key values:

```
SMITH
SMITH
THOMPSON
```

Passing "SMITH" to ptFL will find the second record since it is the last "SMITH". If the key value passed is a null (zero-length) string the last record in the index is returned.

If successful, the record number for your data file can be found in the *RecordNumber* member of the AccessBlock UDT.

If the key value is not located in the index, the *RecordNumber* member of AccessBlock will indicate the position of the next higher key (or, if there is no higher key, the next lower key).

Returns

The ptFL function returns zero if it was successful, or non-zero if an error occurred or if no matching key was found.

Example

```
ptFL "SMITH", KeyBlock
IF ISFALSE KeyBlock.ErrType THEN
  GET hFile&, KeyBlock.RecordNumber, rec
END IF
```

ptFLb

ptFLb

Declare

```
FUNCTION ptFLb(KeyValue AS ANY, x AS AccessBlock) AS LONG
```

Purpose

Find the record before the last occurrence of a key value in an index.

KeyValue may be a fixed-length string or ASCIIZ string.

If your index contains three key values:

```
SMITH
SMITH
THOMPSON
```

Passing "THOMPSON" to ptFLb will find the second record since it is the record before "THOMPSON".

If successful, the record number for your data file can be found in the *RecordNumber* member of the AccessBlock UDT.

Returns

The ptFLb function returns zero if it was successful, or non-zero if an error occurred or no matching key was found.

Example

```
ptFLb "THOMPSON", KeyBlock
IF ISFALSE KeyBlock.ErrType THEN
  GET hFile&, KeyBlock.RecordNumber, rec
END IF
```

ptFN

ptFN

Declare

```
FUNCTION ptFN(KeyValue AS ANY, x AS AccessBlock) AS LONG
```

Purpose

Find the next occurrence of a key value in an index.

KeyValue may be a fixed-length string or ASCIIZ string.

Once you have found a record using ptFF or ptFFa, you can call ptFN to find the next index record in sequence. If *KeyValue* is not found, the *ErrType* member of the AccessBlock UDT is set to -1, and the *RecordNumber* member is not changed. If *KeyValue* is a null (zero-length) string, PowerTree does not attempt to match the *KeyValue*, and simply returns the next record number in sequence.

If successful, the record number for your data file can be found in the *RecordNumber* member of the AccessBlock UDT.

Returns

The ptFN function returns zero if it was successful, or non-zero if an error occurred or if no matching key was found.

Example

```
ptFN BYVAL 0&, KeyBlock
IF ISFALSE KeyBlock.ErrType THEN
  GET hFile&, KeyBlock.RecordNumber, rec
END IF
```

ptFP

ptFP

Declare

```
FUNCTION ptFP(KeyValue AS ANY, x AS AccessBlock) AS LONG
```

Purpose

Find the previous occurrence of a key value in an index.

KeyValue may be a fixed-length string or ASCIIZ string.

Once you have found a record using ptFF or ptFFa, you can call ptFP to find the previous index record in sequence. If *KeyValue* is not found, the *ErrType* member of the AccessBlock UDT is set to -1, and the *RecordNumber* member is not changed. If *KeyValue* is a null (zero-length) string, PowerTree does not attempt to match the *KeyValue*, and simply returns the previous record number in sequence.

If successful, the record number for your data file can be found in the *RecordNumber* member of the AccessBlock UDT.

Returns

The ptFP function returns zero if it was successful, or non-zero if an error occurred or no matching key was found.

Example

```
ptFP BYVAL 0&, KeyBlock
IF ISFALSE KeyBlock.ErrType THEN
  GET hFile&, KeyBlock.RecordNumber, rec
END IF
```

ptFR

ptFR

Declare

```
FUNCTION ptFR(KeyValue AS ANY, x AS AccessBlock) AS LONG
```

Purpose

Find the index record which exactly matches the *KeyValue* and *RecordNumber* member of the AccessBlock UDT.

KeyValue may be a fixed-length string or ASCIIZ string.

This is primarily used when you need to delete a record from an index. You pass the key value and the record number from the data file. You can then call ptDel to delete the found index record, if any. If you pass a null (zero-length) string for *KeyValue*, ptFR will find the index record that matches only the *RecordNumber* value.

Returns

The ptFR function returns zero if it was successful, or non-zero if an error occurred or if no matching key was found.

Example

```
KeyBlock.RecordNumber = 5
ptFR "SMITH", KeyBlock
IF ISFALSE KeyBlock.ErrType THEN
  ptDel "", KeyBlock
END IF
```

ptInit

ptInit

Declare

```
FUNCTION ptInit(Filename AS ASCIIZ, x AS AccessBlock) AS LONG
```

Purpose

Open an existing index file. When opening a new index you need to specify if the index will be accessed by multiple users or threads in the MultiUser member of the AccessBlock UDT.

Once the index has been opened, it can be accessed by the other PowerTree functions. You should close the index using the ptClose statement before your program ends.

Returns

The ptInit function returns zero if it was successful, or non-zero if an error occurred.

Example

```
DIM KeyBlock AS AccessBlock
KeyBlock.MultiUser = 0 'single user
ptInit "MYINDEX.PTX", KeyBlock
```

ptKey

ptKey

Declare

```
FUNCTION ptKey(KeyValue AS ANY, x AS AccessBlock) AS LONG
```

Purpose

Return the value of a key at a given index position.

KeyValue may be a fixed-length string or ASCIIZ string.

This should be called only after successfully looking up a record directly or indirectly, using ptFF, ptFL, ptFFa, ptFLB, ptFN or ptFP. It returns the actual key value associated with the current record. This is particularly useful for resolving searches on partial keywords. For example, if the search was on "Smith", and ptKey tells you that the actual key found is "Smithson", you can decide whether the search should

be stopped or whether any key starting with "Smith" is appropriate for your purposes.

Before calling ptKey, initialize the *KeyValue* parameter to the number of spaces necessary to hold the returned key value.

Returns The ptKey function returns zero if it was successful, or non-zero if an error occurred. The resulting key value is returned in the *KeyValue* parameter.

Example

```
DIM KeyValue AS ASCIIZ * 9 ' key length 9, plus NUL terminator
' ** your index search code goes here
KeyValue = SPACE$(LEN(KeyValue))
ptKey KeyValue, x
```

Technical Support

Technical Support

Technical Support

Visit the Peer Support forums on the PowerBASIC web site or contact us via email at support@powerbasic.com.

Be sure to visit our home page for the latest news, information on upgrades, and programming tips.

Return Policy

Return Policy

Availability of product returns depends on the method of purchase and delivery, so please review the terms below carefully prior to making a purchase decision. The License Agreement also has information about Refunds.

Purchase from PowerBASIC Online Store

Software products purchased through the PowerBASIC Online Store provide electronic delivery only and are considered final. No adjustments or returns are provided outside of the terms of the License Agreement.

Purchase From A Dealer

Currently there are no active Dealers of PowerBASIC. However, if in the past you have purchased a PowerBASIC product from a dealer, distributor, or other third party, any return privileges are based upon the policy and discretion of the seller. PowerBASIC and Drake Enterprises, Ltd,. cannot accept returns on their behalf, so any adjustment or refund must come from the seller alone. Be sure you understand their policy before you purchase.

License Agreement

License Agreement

PowerBASIC License Agreement

LICENSE AGREEMENT IN PLAIN ENGLISH (click here for legal version)

This agreement is between you and Drake Enterprises, Ltd, which is also doing business as PowerBASIC. We're going to refer to Drake as PowerBASIC in this Agreement. Your use of the software will be governed by this Agreement.

Drake legally owns the Software, tools, etc… associated with PowerBASIC, and it's protected by copyrights and trademarks.

This software is leased to you, you don't own it.

The license is good for one person using one computer at a time. You can create your own products using this Software without paying us anything extra, but you can't distribute the source code. Target programs which use PowerBASIC code aren't allowed. An example of this would be, you may write and publish your own target program to sort data in an array, but you can't publish a target program which exports the PowerBASIC ARRAY SORT Command for use with a programming language other than PowerBASIC. You can ask us and we can decide if what you're doing (or planning) is ok. We'll let you know in writing if it's ok or not.

We warrant the physical medium of providing the software will not have defects for 60 days. No other warranties are included, in fact, they're specifically excluded.

If you have a warranty claim, you have to let us know and we have 90 days to fix it or refund your money. Our liability will never be more than the amount you paid for the Software. That's it. No additional liability for PowerBASIC.

You agree to defend us against other parties and not hold us responsible for your actions or products you create, even if you used PowerBASIC to create those products. This includes almost any conceivable notion of liability.

We'll do the same for you if someone else claims we don't own the Software.

Since we're in North Carolina, but have customers all over the world, we're going to use North Carolina law to define and decide on any disagreements.

License Agreement - Legal Version

PowerBASIC License Agreement (Legal Version)

This License Agreement is an agreement between you (referred to herein as "You" or "Licensee") and Drake Enterprises, Ltd, d/b/a/PowerBASIC (referred to herein as "PowerBASIC").

The PowerBASIC compiler and licensed tools ("Software") are proprietary products PowerBASIC and are protected by United States copyright law and international treaties.

The Software and various trademarks, service marks and trade names ("Intellectual Property") are the sole and exclusive property of PowerBASIC, and may be protected by copyright, trade secret and other

intellectual property laws. Any use of PowerBASIC's Intellectual Property without PowerBASIC's express written consent is prohibited.

The Software is licensed, not sold, only on the condition the Licensee agrees to the terms and conditions of this Agreement. PowerBASIC grants to Licensee a non-exclusive, nontransferable license to use the software and any associated manuals and/or documentation under the terms and conditions of this Agreement.

This license is valid for use by one specific person, whose name will be registered with PowerBASIC on one computer at a time. The Software may be moved from one computer to another as long as there is no possibility of it being used on more than one computer at the same time. If the Software is used on a network, one licensed copy of the Software is required for each person who uses the Software. If the licensed product is a compiler, you may distribute the programs you create royalty free. This License grants Licensee no right to sub-license or in any way provide the Software to a third party. You may not distribute the licensed compiler. If the licensed product includes one or more runtime modules, you may reproduce and distribute them royalty free, provided they are distributed only in conjunction with, and as part of your software program, and provided that they bear your copyright notice or the copyright notice which appears on the PowerBASIC label or PowerBASIC.com website. The runtime modules are those files that are required to execute your software program, and which are specifically designated as "runtime modules" in the accompanying PowerBASIC documentation. Your use of any of the demonstration or sample programs provided with this product are governed by the notices and restrictions of the respective author or copyright holder. Except as stated above, you may not resell, transfer ownership, barter, donate, rent, lease, lend, or share the licensed software to/with another person or entity. By written request to PowerBASIC, you may specify a change of licensed user if the replacement user is your employee or family member.

Restrictions

You (Licensee) may use the licensed Software to create and maintain any form of target computer program for your own use. However, if you publish any target computer program, freeware or commercial, which is a tool for programmers (programming language, compiler, interpreter, programmer's library, etc.), you may not export a wrapper for any PowerBASIC command which allows that command to be used in other programming languages. For example, you may write and publish your own target program to sort data in an array. But you may not publish a target program which exports the PowerBASIC ARRAY SORT Command for use with a programming language other than PowerBASIC.

Limited Warranty

PowerBASIC warrants that the physical disks and physical documentation are free of defects in workmanship and materials for a period of sixty days from the date of purchase. If the disks or documentation are found to be defective within the warranty period, PowerBASIC will replace the defective items at no cost to you. The entire liability of this warranty is limited to replacement and shall not, under any circumstances, encompass any other damages.

POWERBASIC MAKES NO OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, ALL SUCH WARRANTIES BEING EXPRESSLY EXCLUDED.

During the Warranty Period, Licensee shall promptly notify PowerBASIC in writing of any claimed deficiency and provide information sufficient to permit PowerBASIC to validate the deficiency. If a deficiency exists which breaches the warranty, PowerBASIC shall, at its sole discretion and within ninety (90) days: (i) correct the deficiency; or (ii) with PowerBASIC's prior written authorization and upon Licensee's de-installation of the Software and return of all copies of the Software to PowerBASIC, refund any License Fee paid to PowerBASIC, whereupon this Agreement shall terminate. Under no circumstances will PowerBASIC's liability exceed amounts paid by the Licensee for use of the Software.

THE REMEDIES SET FORTH ABOVE ARE LICENSEE'S SOLE AND EXCLUSIVE REMEDIES FOR BREACH OF THE WARRANTIES CONTAINED IN THIS AGREEMENT. POWERBASIC SHALL HAVE NO OTHER LIABILITY OR RESPONSIBILITY TO LICENSEE FOR DAMAGES OF ANY KIND, INCLUDING SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES, ARISING OUT OF OR RESULTING FROM THE USE OF THE SOFTWARE OR ANY PROGRAMS, SERVICES OR MATERIALS MADE

AVAILABLE HEREUNDER OR THE USE OR MODIFICATION THEREOF OR UNAUTHORIZED ACCESS TO ANY CONFIDENTIAL DATA.

Indemnification of PowerBasic

LICENSEE HEREBY AGREES TO INDEMNIFY, DEFEND, AND HOLD POWERBASIC HARMLESS FROM AND AGAINST ANY AND ALL LIABILITIES, LOSSES, COSTS, EXPENSE, DAMAGES, AND DEFICIENCIES DURING THE TERM OF THIS AGREEMENT, INCLUDING, WITHOUT LIMITATION, COURT COSTS AND REASONABLE ATTORNEY FEES, WHICH DIRECTLY OR INDIRECTLY ARISE OUT OF, RESULT FROM OR RELATE TO (I) ANY AND ALL LIABILITIES, OBLIGATIONS, OR CLAIMS, WHETHER ACCRUED, ABSOLUTE, CONTINGENT, OR OTHERWISE, WHICH HAVE AS A BASIS THE OPERATION OF LICENSEE, ANY AND ALL ACCOUNTS PAYABLE OF LICENSEE, AND ANY AND ALL TAXES LEVIED OR INCURRED, WHETHER PAYABLE TO A FEDERAL, STATE, LOCAL OR OTHER GOVERNMENTAL AUTHORITY; (II) ANY AND ALL LOSS, CLAIM, CAUSE OF ACTION, LIABILITY, COST, EXPENSES, DAMAGE OR DEFICIENCY DUE TO ANY BREACH BY LICENSEE OF ANY OF ITS REPRESENTATIONS, WARRANTIES, OR COVENANTS CONTAINED IN THIS AGREEMENT; (III) ALL ACTIONS, SUITS, PROCEEDINGS, DEMANDS, ASSESSMENTS, JUDGMENT COSTS AND EXPENSES, INCLUDING THE COST AND EXPENSE OF SUCCESSFUL COLLECTION FROM LICENSEE OR ITS LEGAL REPRESENTATIVE, SUCCESSORS, OR ASSIGNS OF ANY AMOUNT DUE POWERBASIC HEREUNDER OR RESULTING THEREFROM; (IV) ANY HARMFUL SOFTWARE TRANSMITTED BY LICENSEE OR ON BEHALF OF LICENSEE; AND (V) UNAUTHORIZED ACCESS TO ANY PERSONALLY IDENTIFIABLE OR CONFIDENTIAL DATA ATTRIBUTABLE TO THE ACTS OR INACTION OR OMISSIONS OF THE LICENSEE (VI) LICENSEE SHALL INDEMNIFY AND HOLD POWERBASIC HARMLESS AGAINST ANY CLAIM BY A THIRD PARTY RELATING TO LICENSEE'S USE OF THE SOFTWARE OR THE RESULTS THEREOF. The obligations set forth in this section shall survive the termination or expiration of this Agreement.

Indemnification of Licensee

PowerBASIC hereby agrees to indemnify and hold Licensee harmless from and against any and all liabilities, losses, costs, expense, damages, and deficiencies during the term of this Agreement, including, without limitation, court costs and reasonable attorney fees, which directly or indirectly arise out of, result from or relate to any and all liabilities, obligations, or claims, whether accrued, absolute, contingent, or otherwise, which have as a basis the intellectual property ownership of the Software. Licensee agrees to notify PowerBASIC of such claims in writing within 30 days of becoming aware of said claim.

Governing Law

This Agreement and limited warranty shall be construed, interpreted, and governed by the laws of the State of North Carolina, USA, and any action hereunder shall be brought only in North Carolina. If any provision is found invalid or unenforceable, the balance of this Agreement and limited warranty shall remain valid and enforceable. Use, duplication, or disclosure by the U.S. Government of the computer software and documentation in this product shall be subject to the restricted rights under DFARS 52.227-7013 applicable to commercial computer software. All rights not specifically granted herein are reserved by PowerBASIC.

Privacy Policy

Privacy Policy

PowerBASIC is owned by Drake Enterprises, Ltd. At PowerBASIC we are committed to protecting your privacy. We use the information we collect about you to process orders and to provide a more personalized Internet experience. Please read on for more details about our privacy policy.

What information do we collect? How do we use it?

When you order at our online store, we will need to know basic identification and payment method information - name, email address, mailing address, phone number, credit card number/expiration, PayPal account, etc. This allows us to process and fulfill your order and to notify you of your order status.

When you sign up for the Electronic Gazette and new product notices, we need only an email address, which we use to send the information you requested.

We personalize your shopping experience by using your purchases to shape our recommendations about the books, add-ons, and other merchandise that might be of interest to you. We also monitor customer traffic patterns and site usage to help us develop the design and layout of the web site.

We may also use the information we collect to occasionally notify you about important functionality changes to the Web site, new PowerBASIC.Com services, and special offers we think you'll find valuable.

Will PowerBASIC disclose the information it collects to outside parties?

No. PowerBASIC does not sell, trade, or rent your personal information to others. We may offer advertising space to third parties, but no personal information is given to the third party without asking you first. If you do not choose to contact them, they will not have any information about you. We will respond to valid subpoenas in an appropriate fashion (we decide what’s appropriate).

What about "cookies"?

"Cookies" are small pieces of information that are stored by your browser on your computer. Most Web browsers automatically accept cookies, but you can usually change your browser to prevent that. Even without a cookie, you can still use most of the features in our web site, including placing items in your shopping cart and purchasing them. However, without cookies, the Peer Support Forums will not be able to show you which messages are new since your last visit.

Your consent

By using our Web site, you consent to the collection and use of this information by Drake Enterprises, Ltd.